Cryptanalysis of Gu's ideal multilinear map

Alice Pellet-Mary and Damien Stehlé

ENS de Lyon, Laboratoire LIP (U. Lyon, CNRS, ENSL, INRIA, UCBL), France.

Abstract In March, 2015 Gu Chunsheng proposed a candidate ideal multilinear map [9]. An ideal multilinear map allows to perform as many multiplications as desired, while in κ-multilinear maps like GGH [5] or CLT [3,4] one we can perform at most a predetermined number κ of multiplications. In this note, we show that the extraction Multilinear Computational Diffie-Hellman problem (ext-MCDH) associated to Gu's map can be solved in polynomial-time: this candidate ideal multilinear map is insecure. We also give intuition on why we think that the two other ideal multilinear maps proposed by Gu in [9] are not secure either.

1 Introduction

Multilinear maps are a very powerful tool in cryptography. Their possible applications include one-round multipartite key exchange [1], attribute based encryption [7], obfuscation [6], *etc.* Multilinear maps were first introduced by Boneh and Silverberg in 2003 [1]. Then, during ten years, there was no candidate multilinear map for enabling promising applications. In 2013, Garg, Gentry and Halevi proposed the first candidate multilinear map [5], which we will refer to as the GGH map. This map was recently shown insecure by Hu and Jia in 2015 [10]. Back in 2013, some months after the publication of the GGH map, Coron, Lepoint and Tibouchi proposed another candidate multilinear map [3] based on integers instead of polynomials. This map was shown insecure by Cheon *et al.* in 2014 [2], but Coron, Lepoint and Tibouchi proposed a new version of their map in 2015 [4] that seems immune to the attack of Cheon *et al.*

Currently, the CLT map proposed by Coron, Lepoint and Tibouchi [4] seems to be the only candidate multilinear map that has not been shown insecure. Note that in [8], Gentry, Gorbunov and Halevi proposed a so-called graph-induced multilinear map, which can be seen as a variant of a multilinear map. All these candidate multilinear maps are in fact only approximations to multilinear maps: the encodings of elements involve a noise term, which grows every time an arithmetic operation on these encodings is performed. For this reason, the number of multiplications that can be performed is bounded: given a desired number of multiplications, one can derive scheme instantiations allowing this number of multiplications, but once the scheme is instantiated, the number of allowed multiplications cannot be increased.

In March 2015, Gu Chunsheng proposed three candidate multilinear maps [9]. We will describe here the first of these multilinear maps ([9, Section 3]), which we will call the Gu basic map. The two other maps described by Gu in [9] follow the same design principle. Gu's goal was to build an ideal, noiseless, multilinear map, making it possible to perform as many multiplications as desired, even after the map has been generated.

However, we show in Section 4 that the Gu basic map is not secure: there exists a polynomial-time algorithm for solving the extraction Multilinear Computational Diffie-Hellman problem (ext-MCDH) associated to that map. In Section 5, we briefly explain why we believe that the two other maps proposed by Gu in [9] are not secure either.

2 Notations and Definitions

The Gu basic map, like the GGH map, is not a genuine multilinear map but only a graded encoding scheme.

2.1 Graded encoding schemes

In a graded encoding scheme, there are different levels of encodings. A level-0 encoding corresponds to a plaintext (i.e., a message that is not encoded) belonging to some ring R. From a level-i encoding with

$i > 0$, it should be hard to recover the corresponding plaintext (or even any level-j encoding of the same plaintext with $j < i$). We will denote by S_m^i the set of all level-i encodings of the plaintext $m \in R$.

The first difference between the GGH map and the Gu basic map is that the GGH map is a κ-graded encoding scheme, meaning that the level of an encoding is always between 0 and a predetermined value κ. The Gu basic map is an ideal multilinear map, meaning that we can have encodings of levels as large as desired.

A graded encoding scheme supports linear operations and multiplications: it should have two procedures `Add` and `Mult` such that, for all $m_1, m_2 \in R$ and $i, j \geq 0$:

- If $c_1 \in S_{m_1}^i$ and $c_2 \in S_{m_2}^i$ then $\texttt{Add}(c_1, c_2) \in S_{m_1+m_2}^i$.
- If $c_1 \in S_{m_1}^i$ and $c_2 \in S_{m_2}^j$ then $\texttt{Mult}(c_1, c_2) \in S_{m_1 m_2}^{i+j}$.

Note that when we multiply encodings the level of the encoding increases. A graded encoding scheme should also have two procedures `ZeroTest` and `Extract` such that for all $m \in R$

- If $c \in S_m^\kappa$ then $\texttt{ZeroTest}(c) = 1$ if and only if $m = 0$. This procedure tests whether a level-κ encoding is an encoding of 0 or not.
- If $c_1, c_2 \in S_m^\kappa$, then $\texttt{Extract}(c_1) = \texttt{Extract}(c_2) \in \{0,1\}^*$. This procedure enables to extract the same string from different encodings of the same plaintext.

In the above, κ is the maximum level of an encoding if we are using a κ-graded encoding scheme, or κ is any positive integer if we are using an ideal graded encoding scheme.

Even though graded encoding schemes are not genuine multilinear maps, they can still be used for many protocols that can be obtained from multilinear maps. This is the case for instance for 1-round $(\kappa + 1)$-party key exchange. Using a κ-graded encoding scheme, the protocol is as follows:

(1) Each user i for $0 \leq i \leq \kappa$ chooses a random plaintext m_i and computes a level-1 encoding c_i of m_i.
(2) Each user i sends c_i to all other users.
(3) Each user i computes $\widetilde{c_i} = \texttt{Mult}(c_0, c_1, \ldots, c_{i-1}, m_i, c_{i+1}, \ldots, c_\kappa)$. Note that user i replaces c_i by its secret m_i so that $\widetilde{c_i}$ is a level-κ encoding of the element $\prod_i m_i$ and not a level-$(\kappa+1)$ encoding. At this stage, all users have a (possibly different) level-κ encoding of $\prod_i m_i$.
(4) Each user i compute $s = \texttt{Extract}(\widetilde{c_i})$. The secret s is the same for all users.

With an ideal graded encoding scheme like the Gu basic map, one could even perform 1-round key exchange between $\kappa + 1$ users for any positive integer κ. However, we will see later that the protocol above should not be used with an ideal graded encoding scheme, as otherwise the naive attack described in Section 3.1 would apply. We describe later how one can perform key exchange between any number of users.

One classical problem related to κ-graded encoding schemes is the Graded Decisional Diffie-Hellman problem, introduced by Garg, Gentry and Halevi in [5].

Definition 1 (The Graded Decisional Diffie-Hellman problem (GDDH)). *Let $(m_i)_{0 \leq i \leq \kappa}$ be $\kappa+1$ uniform messages in the plaintext space, and let c_i be a level-1 encoding of m_i for all i. Let C_1 be a (randomized) level-κ encoding of $\prod_{i=0}^{\kappa} m_i$ and C_2 be a (randomized) level-κ encoding of a uniformly random plaintext. Then the GDDH problem is, given the c_i's for all i, to distinguish between C_1 and C_2.*

2.2 Algebraic background

We work in the ring $R = \mathbb{Z}[X]/(X^n + 1)$ for n a power of 2. This ring is the ring of integers of the number field $K = \mathbb{Q}[X]/(X^n + 1)$, and so R is a Dedekind ring, which means that all non zero ideals of R are invertible. We also take a large prime integer q. When we say that an element is small, then it means small relative to q.

If $x = x_0 + x_1 X + \cdots + x_{n-1} X^{n-1} \in K$, we call the *size* of x its Euclidean norm $|x| := \sqrt{\sum_{i=1}^n x_i^2}$. We say that an element $x \in K$ is *smaller* than $\alpha > 0$ if $|x| < \alpha$. For $x, y \in K$, we have that $|x+y| \leq 2\max(|x|, |y|)$ and $|x \cdot y| \leq n \cdot |x| \cdot |y|$. Most of the time, we will not be interested in constant factors, or even factors polynomial in n.[1] If $x \in R$, we will let $[x]_q$ or $x \pmod q$ denote the element of R congruent to x modulo q and with all its coefficients in $[-q/2, q/2)$.

[1] When we say that an element is small or large, it means small or large relative to q, and q is exponential in n, so a polynomial in n is something very small.

If $F \in R$, we denote by $\langle F \rangle$ the ideal of R generated by F. We also define $\mathcal{N}(F)$ as the cardinality of $R/\langle F \rangle$. In what follows, we will sometimes invert some element $g \in R$ modulo an element $F \in R$. Here, we show that if $\gcd(\mathcal{N}(g), \mathcal{N}(F)) = 1$, then g is invertible modulo $\langle F \rangle$ and its inverse can be computed in polynomial time. The norm $\mathcal{N}(g)$ of $g \in R$ is the product of its canonical embeddings in $\mathbb{C}$. If we identify the variable $X \in R$ with a primitive complex $2n$-th root of unity, these embeddings can be seen as automorphisms of R (sending X to X^{2i+1} for $0 \leq i < n$), and one of them is the identity. So there exists an element $\tilde{g} \in R$ such that $g\tilde{g} = \mathcal{N}(g)$. Similarly, there exists an element $\tilde{F} \in R$ such that $F\tilde{F} = \mathcal{N}(F)$. As we assumed that $\gcd(\mathcal{N}(g), \mathcal{N}(F)) = 1$, there exist integers u and v such that $u\mathcal{N}(g) + v\mathcal{N}(F) = 1$. This can be rewritten as

$$u\tilde{g}g + v\tilde{F}F = 1.$$

This equation implies that $u\tilde{g}g = 1 \pmod F$ and then $u\tilde{g}$ is the inverse of g modulo F. Furthermore, we can compute $u\tilde{g}$ in time polynomial in n, $\log(\mathcal{N}(g))$ and $\log(\mathcal{N}(F))$.

3 The Gu Basic Multilinear Map

In [9], Gu proposed three candidate multilinear maps. We describe here the first of these candidate multilinear maps ([9, Section 3]), the two others following the same design principle. A consequence of the fact that there is no a priori restriction on the level of encodings, one should also be able to zero-test at any level (in the GGH and CLT maps, we can only zero-test for encodings of levels $\leq \kappa$, for some predetermined κ).

3.1 Naive attack

Assume we have a multilinear map such that we can zero-test at any level, and we know a level-1 encoding v together with its level-0 plaintext $m_v \neq 0$. Then we can solve the GDDH problem at any level κ.

Indeed, assume we have $\kappa + 1$ level-1 encodings c_i, that encode the level-0 plaintexts m_i, and we want to decide whether c is a level-κ encoding of $\prod_{i=0}^{\kappa} m_i$ or a level-κ encoding of a random element. We then compute $d_1 = \texttt{Mult}(m_v, c_0, c_1, \ldots, c_\kappa)$, which is a level-$(\kappa+1)$ encoding of $m_v \cdot \prod_{i=0}^{\kappa} m_i$, and $d_2 = \texttt{Mult}(v, c)$ which is either a level-$(\kappa+1)$ encoding of $m_v \cdot \prod_{i=0}^{\kappa} m_i$ or a level-$(\kappa+1)$ encoding of a random multiple of m_v (there are at least two different multiples of m_v since $m_v \neq 0$). As we can zero-test at every level, we can test if $d_1 = d_2$ and then decide whether c is an encoding of $\prod_{i=0}^{\kappa} m_i$ or not.

This shows that if we want a multilinear map for which GDDH is hard and we can zero-test at every level, then we cannot publicly give a level-1 encoding together with its level-0 plaintext.

3.2 Description of the Gu basic map

The design of the Gu basic map [9] is close to that of the GGH map, but, as we have seen, we cannot publicly give a level-1 encoding of 1. In the GGH map, the level-1 encoding of 1 is what enables each user to encode its secret plaintext to get a level-1 encoding. Here, the authority provides some level-1 encodings y_i and then anyone can generate a level-1 encoding by computing a linear combination of the y_i's: $\sum_i r_i y_i$ with small randomizers r_i. However, we have seen that we cannot publicly give the level-0 plaintexts corresponding to the y_i's, so the secret related to $\sum_i r_i y_i$ will be the r_i's. From $\sum_i r_i y_i$, it should be hard to recover small r_i's that give the same result.

The Gu basic map works as follows.

Setup. We work in the ring $R = \mathbb{Z}[X]/(X^n + 1)$ for n a power of 2, and we have a large prime integer q which is exponential in n.

(1) Sample elements $(f_i)_{1 \leq i \leq m}$ and g in R with small coefficients (integer m is of the order of the security parameter λ). Let $f = \prod_{i=0}^{m} f_i$.
(2) Compute $F = p_0 f$ for a small random element p_0 in R.
(3) Generate $\tau = O(n^2)$ elements $y_i = (a_i g + t_i f) \bmod F$ with a_i and t_i small random elements of R for all $1 \leq i \leq \tau$.
(4) For all $1 \leq i \leq \tau$, generate $p_{zt,i} = [\sum_{j=0}^{m} h_j (a_i + s_{i,j} f_j) f_j^{-1}]_q$ with h_j and $s_{i,j}$ small random elements of R.

All the random small elements that are used in this initialization step are sampled according to a (discrete) Gaussian distribution with small standard deviation (but not necessarily the same standard deviation for all elements). The exact distribution of the elements is not important for the attack, so we will not explicit it. More details can be found in [9].

Note that so far, everything is small (compared to q), except the $f_j^{-1} \pmod q$ in the $p_{zt,i}$ parameters. Also, the element F is a multiple of f, so if we perform computations modulo F and then reduce the result modulo f, it will give the same result as if we had performed the computations directly modulo f. Giving F allows to perform computations modulo F and so have encodings of bounded size without making f public.

The authority publicly gives F, the y_i's and the $p_{zt,i}$'s (f, g and the a_i's are kept secret).

Level-i encodings. The plaintext space is $\mathcal{P} = R/\langle f\rangle$. A level-$i$ encoding of $m \in \mathcal{P}$ is an element c reduced modulo F such that $c \equiv mg^i \bmod f$. When adding or multiplying encodings, the results are reduced modulo F in order to keep the size of the encodings bounded, and as F is a multiple of f, this will not change the result modulo f. As a consequence, adding two encodings of level i gives a level-i encoding of the sum, and multiplying a level-i encoding by a level-j encoding gives a level-$(i+j)$ encoding of the product.

To generate a level-j encoding, we choose τ small elements $r_i \in R$ and output $c = \sum_i r_i(y_i)^j \bmod F$. We cannot know the level-0 plaintext of the encoding we generate, but the analogue of the level-0 secret are the r_i's, which tell how to generate c with small elements.

Zero-testing. In order to test if a level-j encoding c is an encoding of zero, we take any linear combination $p_{zt} = \sum_i w_i p_{zt,i}$ with small w_i in R (for instance we can simply take $p_{zt} = p_{zt,1}$) and we compute $[c \cdot p_{zt}]_q$. If this is small compared to q, then c is an encoding of zero, otherwise it is not. The idea underlying correctness is that c is an encoding of zero if and only if it is a multiple of f, and in p_{zt}, the large terms come from the $[f_j^{-1}]_q$, so if we multiply p_{zt} by a multiple of f the term $[f_j^{-1}]_q$ cancels out and the result is small, otherwise the result is of the order of q with overwhelming probability.

Extraction. If we just want a way to extract from an encoding a string s that depends only on the hidden plaintext we could compute $s = \mathrm{MSB}([p_{zt,1} \cdot c]_q)$ where MSB denotes the most significant bits of the result, and c is an encoding of any level greater than zero. However, this simple procedure will not enable us to perform key exchange for instance, because we do not have access to the corresponding level-0 plaintext when we encode a message. To circumvent this problem, Gu proposed an extraction procedure that multiplies by a level-0 plaintext at the same time as it extracts. The procedure `Extract` takes as input τ small elements r_i with $1 \le i \le \tau$ in addition to the encoding c. Then, if we denote by m the plaintext corresponding to the encoding $\sum_i r_i y_i$, we can interpret the extraction procedure as "multiply c by the level-0 plaintext m and then extract."

More formally, the extraction procedure works as follow:

$$\texttt{Extract}((r_i)_{1\le i\le\tau}, c) = \mathrm{MSB}([(\sum_i r_i p_{zt,i}) \cdot c]_q).$$

We will now see how to use this extraction procedure to perform key exchange.

Multipartite key exchange. Assume we have $\kappa+1$ users (κ is not fixed here) that generate level-1 encodings c_j for $0 \le j \le \kappa$. User j cannot compute $m_j \cdot \prod_{l\neq j} c_l$ like before, as it does not know the level-0 plaintext associated to its encoding. The idea is that when it performs the extraction, it also multiplies by his level-0 plaintext, obliviously.

For all $1 \le i \le \tau$, the level-0 plaintext of y_i (which is the a_i kept secret by the authority) is hidden in the associated parameter $p_{zt,i}$, so if user j generates $m_j = \sum_i r_{i,j} y_i$ with $r_{i,j}$ small, it knows the $r_{i,j}$'s and can compute $p_{zt}^{(j)} = \sum_i r_{i,j} p_{zt,i}$ that is associated to its encoding. Then, when it multiplies $\prod_{l\neq j} c_l$ by $p_{zt}^{(j)}$, it multiplies by his secret level-0 plaintext at the same time as it extracts the shared secret. The $\kappa+1$ multipartite key exchange is then the same as for a κ-graded encoding scheme, except for the `Extract` procedure:

(1) Each user j for $0 \le j \le \kappa$ chooses random small elements $(r_{i,j})_{1\le i\le\tau}$ in R and computes its level-1 encoding $c_j = \sum_i r_{i,j} y_i$.
(2) Each user j sends c_j to all other users.
(3) Each user j computes $s = \texttt{Extract}((r_{i,j})_{1\le i\le\tau}, \prod_{l\neq j} c_l)$. The secret s is the same for all users.

The following result ensures the correctness of this multipartite key exchange protocol.

Lemma 1 (Adapted from [9, Lemma 3.7]). *If two users j_1 and j_2 compute their respective zero-testing parameters $p_{zt}^{(j_1)} = \sum_i r_{i,j_1} p_{zt,i}$ and $p_{zt}^{(j_2)} = \sum_i r_{i,j_2} p_{zt,i}$, then, with overwhelming probability, the elements $[p_{zt}^{(j_1)} \cdot \prod_{l \neq j_1} c_l]_q$ and $[p_{zt}^{(j_2)} \cdot \prod_{l \neq j_2} c_l]_q$ have the same most significant bits, i.e.,*

$$\texttt{Extract}((r_{i,j_1})_{1 \leq i \leq \tau}, \prod_{l \neq j_1} c_l) = \texttt{Extract}((r_{i,j_2})_{1 \leq i \leq \tau}, \prod_{l \neq j_2} c_l).$$

3.3 Algorithmic problems

As we do not have access to level-0 plaintexts in the Gu basic map, the GDDH problem defined in Section 2.1 does not seem interesting, at least from the perspective of key exchange. Instead of GDDH, we will be interested in two algorithmic problems proposed by Langlois, Stehlé and Steinfeld in [11], that are generalizations of the classical Decisional Diffie-Hellman (DDH) and Computational Diffie-Hellman (CDH) problems.

Definition 2 (The Extraction Graded Computational Diffie-Hellman problem (ext-GCDH)). *Let κ be an arbitrary positive integer. For $0 \leq j \leq \kappa$ and $1 \leq i \leq \tau$, let $r_{i,j}$ be random small elements of R. Then let $c_j = \sum_i r_{i,j} y_i$ for all $0 \leq j \leq \kappa$. The ext-GCDH problem is, given the c_j's for all j, to construct $s = \texttt{Extract}((r_{i,0})_{1 \leq i \leq \tau}, \prod_{j>0} c_j)$.*

Definition 3 (The Extraction κ-Graded Decisional Diffie-Hellman problem (ext-GDDH)). *Let κ be an arbitrary positive integer. For $0 \leq j \leq \kappa$ and $1 \leq i \leq \tau$, let $r_{i,j}$ be random small elements of R. Then let $c_j = \sum_i r_{i,j} y_i$ for all $0 \leq j \leq \kappa$. Let $(u_i)_{1 \leq i \leq \tau}$ be small random elements of R and define $s_1 = \texttt{Extract}((r_{i,0})_{1 \leq i \leq \tau}, \prod_{j>0} c_j)$ (the genuine secret) and $s_2 = \texttt{Extract}((u_i)_{1 \leq i \leq \tau}, \prod_{j>0} c_j)$. The ext-GDDH problem is, given the c_j's for all j, to distinguish between s_1 and s_2.*

Note that ext-GDDH reduces to ext-GCDH. In [9], Gu based the security of his basic map on the presumed hardness of ext-GDDH. We will show that ext-GCDH can be solved in polynomial time for the Gu basic map.

4 Cryptanalysis of the Gu Basic Map

Even though the naive attack does not work for the Gu basic map, we prove that ext-GCDH can be solved in polynomial time for this map, for any positive integer κ.

Let κ be a positive integer and assume we are given $\kappa + 1$ level-1 encodings $c_j \equiv m_j g \pmod{f}$ for $j = 0, \ldots, \kappa$ (the m_j's are kept secret). The attack proceeds as follows:

(1) Compute $c = \prod_j c_j \pmod{F}$. We have that $c \equiv (\prod_j m_j) \cdot g^{\kappa+1} \pmod{f}$. This is a level-$(\kappa + 1)$ encoding of $\prod_{j=0}^{\kappa} m_j$.
(2) Sample small elements $(r_i)_{1 \leq i \leq \tau}$ in R and compute $\widetilde{c_0} = \sum_i r_i y_i \bmod F$. We restart this step until we obtain $\widetilde{c_0}$ invertible modulo F (if one of the y_i is invertible modulo F, it can be taken directly). See Remark 2 below for an estimation of the number of times we have to restart to get a good $\widetilde{c_0}$. We have $\widetilde{c_0} \equiv \widetilde{m_0} g \pmod{f}$ for some $\widetilde{m_0} \in \mathcal{P}$.
(3) Compute $d = c \cdot \widetilde{c_0}^{-1} \pmod{F}$. As F is a multiple of f, we know that computing modulo F and then reducing modulo f is the same as computing directly modulo f, so $d \equiv (\prod_j m_j) \cdot \widetilde{m_0}^{-1} \cdot g^{\kappa} (\bmod f)$. Moreover, d is reduced modulo F, so d is a level-κ encoding of $(\prod_j m_j)\widetilde{m_0}^{-1} \pmod{f}$. Note that d is exactly what user 0 would get by multiplying the other's encodings if he had generated $\widetilde{c_0}$ as level-1 encoding (and user 1 had generated $c_1(c_0/\widetilde{c_0})$ in order to get the same final product). And we know the secret r_i's associated to $\widetilde{c_0}$. So at this point, we are like one of the genuine key-exchange users.
(4) Extract the secret using $p_{zt} = \sum_i r_i p_{zt,i}$ in order to multiply by $\widetilde{m_0}$ at the same time as we extract: return $s = \texttt{Extract}((r_i)_{1 \leq i \leq \tau}, d) = \text{MSB}([p_{zt} d]_q)$, the shared secret.

Remark 1. This attack works well here because we can divide a level-i encoding by a level-j encoding $(i \geq j)$ and obtain a level-$(i - j)$ encoding of the division. This is not the case in the GGH and CLT maps since the numerators of the encodings must be small and, when dividing two small elements modulo q, the result is likely to be of the order of q, i.e., not small at all.

Remark 2. For this attack, we need to find in polynomial time an element $\widetilde{c_0}$ invertible modulo F. We have seen in Section 2.2 that if $\gcd(\mathcal{N}(\widetilde{c_0}), \mathcal{N}(F)) = 1$, then $\widetilde{c_0}$ is invertible modulo F and we can compute its inverse in polynomial time. But are there a lot of $\widetilde{c_0}$ such that $\gcd(\mathcal{N}(\widetilde{c_0}), \mathcal{N}(F)) = 1$? We will assume here that the norm of $\widetilde{c_0}$ is uniform modulo $\mathcal{N}(F) =: N$. We then want to know the proportion of invertible elements in $\mathbb{Z}/N\mathbb{Z}$. This is exactly $\phi(N)/N$, where ϕ denotes Euler's totient function. Using Martens' second formula, we can see that in the worst case, this is asymptotically $\frac{1}{\log \log N}$. So in average, it suffices to pick at most $\log\log(\mathcal{N}(F))$ elements $\widetilde{c_0}$ to get an invertible one. The norm of F is of the order of $|F|^n$, hence $\log\log(\mathcal{N}(F)) \approx \log(n \log(|F|))$, which is polynomial (and even logarithmic) in the bit-size of F.

To conclude, this attack solves ext-MCDH for the Gu basic map in polynomial time.

5 Other Maps Proposed by Gu

Gu also described in [9] two other candidate ideal multilinear maps (i.e., with κ that can be chosen arbitrarily large even after the generation of the map). The second map is very close to the first one we described above. The only difference is that the procedure to extract is more involved. It does not seem immune to the attack described in Section 4, because this attack extracts like a regular user.

The third map proposed by Gu again uses the same principle but the elements are replaced by matrices. However, in this third scheme, we are given level-1 encodings y_i but we are also given the corresponding level-0 plaintexts x_i. So the naive attack described at the beginning of Section 3 to solve the GDDH problem should work against this third map.

To conclude, none of the maps described by Gu in [9] seems secure.

Acknowledgments. This work has been supported in part by ERC Starting Grant ERC-2013-StG-335086-LATTAC.

References

1. D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003.
2. J. H. Cheon, K. Han, C. Lee, H. Ryu, and D. Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, pages 3–12. Springer, 2015.
3. J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493. Springer, 2013.
4. J.-S. Coron, T. Lepoint, and M. Tibouchi. New multilinear maps over the integers. *IACR Cryptology ePrint Archive*, 2015:162, 2015.
5. S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, volume 7881 of *LNCS*, pages 1–17, 2013.
6. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49. IEEE Computer Society Press, 2013.
7. S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO*, pages 479–499. Springer, 2013.
8. C. Gentry, S. Gorbunov, and S. Halevi. Graph-induced multilinear maps from lattices. In *TCC*, pages 498–527. Springer, 2015.
9. C. Gu. Ideal multilinear maps based on ideal lattices. *IACR Cryptology ePrint Archive*, 2015:269, 2015.
10. Y. Hu and H. Jia. Cryptanalysis of GGH map. *IACR Cryptology ePrint Archive*, 2015:269, 2015.
11. A. Langlois, D. Stehlé, and R. Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. In *EUROCRYPT*, LNCS, pages 239–256. Springer, 2014.