Zeroizing framework

Shahed Sharif

February 7, 2019

1 Overview

As mentioned earlier, both GGH13 and GGH15 are broken. Miles-Sahai-Zhandry and Bartusek-Guan-Ma-Zhandry developed a framework for viewing all known attacks on these two cryptosystems, known respectively as the *weak multilinear map model* and the *GGH15 zeroizing model*. The two models are similar, reflecting the similarity of the attacks against GGH13 and GGH15. The goal of these notes is to describe the GGH15 zeroizing model, and describe the CLLT attack on GGH15 as an instance of the model.

First, a word on what *model* means in this context. *Model* is short for *threat model*, which is framed as a game played by a single player, the adversary. The model specifies the following data, which may be viewed as the "rules of the game":

- a set of public parameters,
- a set of public oracles and algorithms,
- a bound on computational resources, and
- a win condition.

The bound on computational resources is typically of the form "polynomially many oracle queries," and otherwise access to polynomial time algorithms. A cryptographic scheme is insecure under the threat model if an adversary can win with non-negligible probability.

The real point of a model is to prove *security*. In both cases, the authors of the respective papers are able to revise both GGH13 and GGH15 to obtain security under the model, but with respect to *indistinguishability obfuscation*. To clarify, the cryptographic mappings constructed in GGH13 and GGH15 can be used for a number of purposes. We have focused on multi-party Diffie Hellman key exchange. However, another application is iO, or indistinguishability obfuscation. Roughly, iO means the ability to disguise the source code of a program so that the internals are completely incomprehensible to an observer, but the functionality of the program is unchanged. It is unclear to me if the iO security proofs extend to multi-party Diffie-Hellman.

2 GGH15 Zeroizing model

The public parameters for the GGH15 zeroizing model is a *subset* of the set of public parameters for GGH15. Recall that in GGH15, we have a graph G, and for each edge $u \rightsquigarrow v$, we can encode a plaintext S as $C := [S]_{u \rightsquigarrow v}$. In practice, encoding is done ahead of time—users never directly encode plaintexts. Instead, a set of encoded plaintexts $C_i := [S_i]_{u \rightsquigarrow v}$ is made available. Since GGH15 encodings are linear maps (with noise), we have

$$\left[\sum \alpha_i S_i\right] = \sum \alpha_i C_i$$

where, because of the noise, we need the α_i to be small integers. In GGH15, the C_i are public. In the zeroizing model, they are *not* directly available; instead, the adversary only has access to them as black box representatives. To distinguish between the specific matrix C_i and its black box representative, we write $\hat{C}_i$ for the latter. One should think of the $\hat{C}_i$ as transcendental elements of a (noncommutative) polynomial ring. BGMZ refer to these elements as "handles."

The adversary has access to the following zero-testing oracle. The oracle takes as input a polynomial $p \in \mathbb{Z}\langle\{\hat{C}_i\}\rangle$ of polynomial size in our security parameter. The oracle first checks that the polynomial is *edge-respecting*. This means that every monomial is a well-defined product under the constraints of GGH15, and that the product is an encoding for the source-to-sink path $1 \rightsquigarrow d$. If this holds, the oracle computes $p(\{C_i\})$, and zero-tests the result with output (T, b), where T is the matrix computed for the zero-text, and b is the resulting Boolean. Specifically, $T = A_1 p(\{C_i\}) \pmod q$, and b is `TRUE` when T is small. If b is false, the oracle throws an error. If b is true, the oracle stores T and outputs handles to the entries of T.

The adversary makes polynomially many queries to the oracle; say these are given by the polynomial p_u with non-error results T_u, and corresponding handle $\hat{T}_u$. Recall the plaintexts S_i. Let S_{ijk} be the j,kth entry of S_i, and let $\hat{S}_{ijk}$ be handles for these entries. In order to win, the adversary would like to construct a polynomial $Q \in \mathbb{Z}\langle\{\hat{T}_u, \hat{S}_{ijk}\}\rangle$ such that

- $Q(T_u, S_{ijk}) = 0$,
- $Q(T_u, \hat{S}_{ijk}) \neq 0$, and
- $Q(\hat{T}_u, S_{ijk}) \neq 0$.

Roughly, Q is an algebraic relation satisfied by the plaintexts. The second and third conditions are nontriviality conditions.

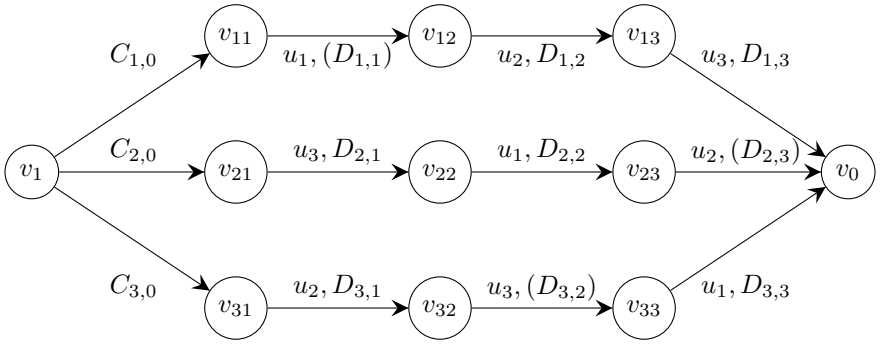
The adversary is allowed to submit polynomially many Q to the oracle in order to win.

Some observations: First notice that the zeroizing model does not explicitly mention the prime q; in fact, the adversary works over $\mathbb{Z}$. This goes to the heart of the term "zeroizing": zero-testing succeeds if and only if the product term—in our notation, T—is small. In that case, the relation $T \equiv A_1 p(C_i) \pmod q$ lifts to an equality over $\mathbb{Z}$.

Also observe that it is unclear how to break, say, Diffie-Hellman in this model. In fact, the model does *not* show how to break any protocols! Rather, existing attacks break Diffie-Hellman by *first* winning in the zeroizing model. That is, insecurity in the zeroizing model is a necessary, but not sufficient condition to break (say) Diffie-Hellman. In an information-theoretic sense, this is good enough—ideal security proofs show that outputs are indistinguishable from noise. In the zeroizing model, the polynomial Q gives an algebraic condition that distinguishes the encoded plaintexts from noise.

2.1 Instantiation of GGH15 zeroizing model

We now show how the model is instantiated in the CLLT attack on GGH15. We first recall the key exchange protocol.



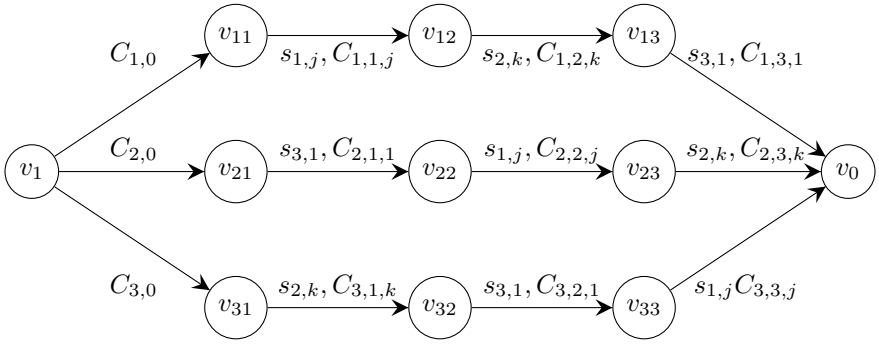
In the above diagram, $D_{i,j}$ is the encoding along the given edge of the secret plaintext u_t appearing next to it. To be precise, let A_{ij} be the matrix associated to node v_{ij}, and A_i associated to v_i. Then for example

$$A_{ij}D_{ij} \equiv u_t A_{i(j+1)} + E_{ij} \pmod{q}$$

for some error matrix E_{ij}.

The $C_{i,0}$ are encodings of the identity, here for convenience. The parenthetical encodings are kept secret, so that only user t knows the corresponding encoding of u_t. The pre-zero-tested key is then the product of the matrices along any path.

Recall that users do not work directly with the u_t. Rather, there are predefined secret plaintexts, which we will call $s_{i,\ell}$ where $i = 1, 2, 3$ and ℓ varies in some large set, and published encodings of these plaintexts along each edge. The first subscript is (I believe) just a record-keeping device: we assume that user i computes the encoding of their key u_i as a linear combination, with small coefficients, of the $s_{i,\ell}$, where the ℓ varies. It is therefore natural to consider the special case $u_1 = s_{1,j}$, $u_2 = s_{2,k}$, $u_3 = s_{3,1}$.



Taking products across the bottom two rows, we obtain

$$p_{j,k}(C) = C_{2,0}C_{2,1,1}C_{2,2,j}C_{2,3,k} - C_{3,0}C_{3,1,k}C_{3,2,1}C_{3,3,j}$$

is a top-level encoding of 0. The adversary feeds these polynomials, over all j, k, to the oracle, obtaining (handles to) zero-test matrices $T_{j,k}$. Let W be the matrix for which $W_{j,k}$ is the first entry of the zero-test matrix $T_{j,k}$. Let M be the matrix consisting of (handles to) the $s_{1,j}$ stacked vertically. We apply some tricks, to be discussed, to concatenate W and M into a single square matrix, then let Q be the determinant. The claim is that with non-negligible probability, Q wins in the model.

To prove the claim, we reveal the aforementioned tricks. We first guess the ranks of W and M; these guesses will be correct with non-negligible probability. Based on these guesses, we choose appropriately sized random matrices U, V, U', V', and replace W, M with UWV and $U'MV$. BGMZ show that if our rank guesses were correct, with high probability W and M will be full rank. Notice that W depends only on the T_u and M depends only on the S_{ijk}, and hence the second two win conditions are satisfied.

The remaining win condition is the hardest to deal with. To prove that it holds, it suffices to find a vector X which lies in the column space of both W and M. Recall that $T_{j,k}$ is the zero-tested value for

$$p_{j,k}(C) = C_{2,0}C_{2,1,1}C_{2,2,j}C_{2,3,k} - C_{3,0}C_{3,1,k}C_{3,2,1}C_{3,3,j}.$$

This works out to

$$s_{1,j}s_{3,1}E_{2,3,k} + \cdots$$

where $E_{2,3,k}$ is one of the error matrices. One can rewrite the sum as a sort of dot product

$$\begin{bmatrix} s_{1,j} & \cdots \end{bmatrix} \cdot \begin{bmatrix} s_{3,1}E_{2,3,k} \\ \vdots \end{bmatrix}.$$

Essentially, we can arrange things in such a way that the first column of the left matrix will be a column of M. This is the desired vector X.

2.2 Breaking Diffie-Hellman

To complete the CLLT attack, one can explicitly compute these matrices to obtain linear dependence relations. In particular, repeating the attack with the Diffie-Hellman graph diagram allows one to write the u_i as linear combinations of the $s_{i,\ell}$. This isn't by itself enough to break Diffie-Hellman, since one would also like these linear combinations to have small coefficients. Instead, the adversary uses these "fake" linear combinations to create a "fake" Diffie-Hellman key. The last step is a highly nontrivial error reduction step, to deduce an equivalent Diffie-Hellman key from the "fake" key.