# CSE206A Project: Fully Homomorphic Encryption

Michael Walter James Mouradian Victor Balcer

June 15, 2014

For our project we implemented fully homomorphic encryption, using the scheme described in class and in lecture notes, using the Sage open-source mathematics framework.

## 1 Results

We implemented a proof-of-concept NAND gate, since it is functionally complete. The implementation is highly inefficient and features no algebraic optimizations; our proof-of-concept takes approximately twenty minutes to compute an encrypted NAND over two bits with the following parameters: $n = 4$, $q = 64$, $Q = 2^{25}$, and $\beta = 1$ (i.e. keys $s \in \mathbb{Z}_{64}^4$ and $z \in \mathbb{Z}_{2^{25}}^4$). These are the smallest parameters for which we can compute a NAND gate with error $\beta > 0$. To validate the gate, we computed an encrypted NAND over two randomly generated bits and compared the the decrypted output with the expected output. Decryption was successful across twenty independent trials. The trials are meaningful; with increased error the test reports both positive and negative results.

## 2 Parameter Tuning

Both key switching and homomorphic decryption work on parameters: $n = 8$, $q = 128$, $Q = 2^{17}$, and $\beta = 1$. However, these parameters will fail to compute a homomorphic NAND gate as the additional multiplication adds an error factor of $n \log Q$. Based on the bounds from the formulas, we need to set $Q \geq 2^{35}$. However, based on previous performance, we estimate the time to perform this operation to take over 30 hours.

We did not rely on the formulas to compute our parameters. Because the formulas ignore noise cancellation, they produce larger bounds, with significant time increases, than are necessary for us to reliably calculate an encrypted NAND compared to those found from our experimental tests.

## 3 Limitations and Optimizations

For very small parameters, the limiting factor of our code is matrix binary decomposition. After trying several optimizations within Sage, we migrated this to low-level compiled code called from within Sage; however, the type conversions to and from Sage matrices took longer than performing the decomposition in Sage itself. Ultimately, we used a compiled version of the decomposition code, with pre-allocated space for each matrix. Additionally, we pre-computed all constant decomposition matrices of unit vectors used in convolution for homomorphic decryption.

As parameters increase even slightly, matrix multiplication approaches the decomposition function with respect to running time. With realistic parameters that add any remote degree of security to homomorphic operations, matrix multiplication will clearly dominate the running time of our implementation.

## 4 Future Work

There are three obvious improvements to be made on this implementation. The first is to use CRT to represent elements in $\mathbb{Z}_q$ more efficiently. The second is to use algebraic lattices, whose structure can be leveraged in order to decrease the complexity of our implementation. The third is to focus on implementation, using lower-level languages with hand-optimizations. Without all of these in conjunction, the implementation will likely not ever become fast enough to use for useful circuits in practice.