Proposed Solution for Challenge Problem 1

Yuriy Polyakov, Kurt Rohloff, and Vinod Vaikuntanathan

✦

1 Formulation of the Linear Classifier Problem

We are given 5 attributes. The input for each attribute is 8 bits. We need to build an obfuscated classifier that returns true if the input values of all 5 attributes are within certain attribute-specific ranges (specified using thresholds). Thresholds are used in the original problem formulation but we will consider a more general case where the value of an attribute input can match an arbitrary set of integer values $\in [0, 256)$ for each attribute.

1.1 Plaintext Program Compute Model

For each attribute, generate a vector of size 256. Set the components at the indices within the ranges to 0, and the components at other indices to some random values between 1 and $p-1$. Concatenate the vectors for 5 attributes into a "weight" vector $(w_1, \ldots, w_N)$, where $N = 1280$.

Represent each data input as a vector of size 256. Set the component at the index corresponding to the input value to 1 and all other components to 0. Concatenate the input vectors for 5 attributes into a "data" vector $(x_1, \ldots, x_N)$.

Compute $\sum_{i=1}^{N} w_i x_i \bmod p$. If the input values for all 5 attributes match the corresponding ranges, we will get a sum of zeros, which is zero. If at least one input value does not match, we will get a random positive number with a high probability (which depends on p).

Note that we need to sum only 5 integers w_i because all other values of x_i are set to zero. In other words, the complexity of plaintext computation is 4 additions vs. N multiplications and $N-1$ additions.

1.2 Obfuscated Program Compute Model

We use the LWE-based scheme defined in Section 2.1.

The evaluation computation has two parts: token generation and evaluation using the token.

Token generation is written as

$$\mathbf{t} := \sum_{i=1}^{N} x_i \mathbf{s_i} \in \mathbb{Z}_q^n.$$

Note that we need to sum only 5 vectors because all other values of x_i are set to zero.

The evaluation is written as

$$\sum_{i=1}^{N} w_i x_i = \sum_{i=1}^{N} c_i x_i - \langle \mathbf{a}, \mathbf{t} \rangle \,(\mathrm{mod} p)\,.$$

The first sum requires only 4 integer additions because all other values of x_i are set to zero. The second term has a single inner product of two vectors of size $\mathbb{Z}_q^N$.

Therefore, the total complexity of the obfuscated program evaluation is one inner product of vectors of n integers + 4 vector additions + 4 integer additions.

1.3 Parameter Selection

The ciphertext modulus q is bounded as $q/4 > 5pB_{err}$, where B_{err} is the norm of error polynomials, e.g., 12σ, where $\sigma = 3.2$.

The value of p is determined by the probability that we run into a collision with zero when adding up 5 random values between 1 and $p-1$. We need to compute the number of combinations of a sum of 5 random numbers that result in p, $2p$, $3p$, or $4p$. The detailed analysis will be added later. For now, we assume that $p = 2^{40}$ is sufficient for practical purposes.

The above implies that we can use a ciphertext modulus of less than 59 bits, which gives us more than 128 bits of lattice security for $n = 2048$ (see the first table in Section 5.4 of [1]).

1.4 Precomputed vector of secret keys vs "key expansion" on demand

If the dimension N is relatively small, then the vector of secret keys can be precomputed during the key generation. If a larger dimension N is needed, the secret keys can be computed on the fly using a hashing or AES encryption technique. Our current implementation supports both modes (precomputed and AES encryption).

Note that the current AES-based implementation can be improved both from the security and performance perspectives. Its main purpose is to show that AES-based secret key expansion can be executed on demand while still meeting the runtime requirements of Challenge Problem 1. For instance, the secret key could be generated using a hashing technique. A salt could be used as the intial seed for CTR. AES may also be replaced with a hashing technique.

1.5 Program Size and Runtimes

For the case of precomputed secret keys, the total evalution runtime (for computing one inner product of vectors of n integers + 4 vector additions + 4 integer additions) is below 100 μs.

For the case of on-demand key generation (using AES), we have to also add the time of running $5 \times \frac{n}{2}$ AES encryptions to the token generation. Using our current implementation of AES, this running time is about 0.5 ms (but it can be further improved).

The runtime of key generation is well below 1 second.

The obfuscation runtime corresponds to approximately N inner products of two vectors with 2048 integers and $O(N)$ AES encryptions. The runtime is below 1s (for $N = 1280$).

The obfuscated program size is N integers in $\mathbb{Z}_q$, i.e., 1280*8 = 10KB.

The token size is n integers in $\mathbb{Z}_q$, i.e., 2048*8 = 16KB.

If the precomputed approach is used, the secret key size is $1280 * 2048 * 8$ = 20 MB.

If the AES on-demand generation method is used, the secret key size corresponds to the size of the nonce needed to generate vectors in $\mathbb{Z}_q^n$, i.e., not more than 2048*8 = 16KB.

The "public" key a size is n integers in $\mathbb{Z}_q$, i.e., 2048*8 = 16KB.

1.6 Security Attacks

Since the proposed solution is based on a linear function, we can find all components of the weights vector in $N = 1280$ queries by solving the linear system.

If the ranges corresponding to matches are larger than 1, then the number of queries to solve the classier problem is reduced. This should be studied on a case by case basis, and it is the limitation of the classifier problem itself rather than the proposed solution. For instance, if each attribute has only one threshold (and let's say the input value should be above the threshold), then the the 5-byte classifier can be learned in 40 queries using binary search.

The dimension of the linear system can be increased by assigning joint weights for two attributes. In this case, we would use 2*256^2 + 256 = 131,328 dimensions. The evaluation time would be roughly the same (as it only depends on the number of attributes). The obfuscated program would be larger by a factor of 100 (only 1MB). The number of queries would be bounded by 131,328 (actually a somewhare smaller number not to get close to solving the linear system).

1.7 Scalability w.r.t. dimension N

In the precomputed scenario, the storage requirement for the secret keys is 2048*8*N. This implies that the dimension N can be up to 10^6 to meet the 16GB memory requirement of Challenge Problem 1 (assuming all secret keys are stored in the memory). A larger number of dimensions can be supported if the secret keys are read from a fast persistent storage, such as SSD.

In the AES scenario, the secret size is small, and the main storage limitation is the size of the obfuscated program, which is N*8. This implies that N up to 2×10^9 can be supported within the memory limits of Challenge Problem 1 (assuming the obfuscated program is stored in the memory). A larger number of dimensions can be supported if the obfuscated program is read from a fast persistent storage, such as SSD.

Note that the evaluation runtime only depends on the number of attributes, which is 5 in this case, and does not depend on N.

2 LWE Token-Based Obfuscation of Linear Functions

The purpose of this scheme is to perform evaluation on obfuscated tests for linear functions. The actual evaluation function can be described as $\sum_{i=1}^{N} w_i x_i$, where $(x_1, \ldots, x_N) \in \mathbb{Z}_p^N$ and $(w_1, \ldots, w_N) \in \mathbb{Z}_p^N$ refer to data and weights, respectively. The weights (tests) are encrypted/obfuscated and tokens are generated for cleartext data. The total size of the dataset is N.

2.1 LWE Secret Key Scheme

2.1.1 Parameters

- Prime modulus q.
- Number n, the LWE security parameter of the system.
- Discrete Gaussian distribution χ with standard deviation σ.

2.1.2 Key Generation

Generate N secret vectors $\mathbf{s_i} \in \mathbb{Z}_q^n$, where $N \geq 2$. For example, use a nonce K and define $\mathbf{s_i}$ to be a hash of K concatenated to the index i. Another alternative is to compute it as AES(K,i). More specifically, we can generate a secret key K for AES. Then do encryptions of a counter to generate 128-bit random sequences. Use these sequences for random numbers in $\mathbb{Z}_q$.

Note that we need to store only the nonce and can generate a particular key on the fly. In other words, the space requirements for the secret key vectors are limited by the value of n (negligibly small from the practical perspective).

2.1.3 Obfuscation

Choose a random vector $\mathbf{a} \in \mathbb{Z}_q^n$ and error values (numbers) $e_i \in \mathbb{Z}_q$ generated using χ. Compute

$$c_i := \langle \mathbf{a}, \mathbf{s_i} \rangle + pe_i + w_i \in \mathbb{Z}_q.$$

Then the complete obfuscated program $\mathbf{c}$ is written as

$$\mathbf{c} := \{\langle \mathbf{a}, \mathbf{s_i} \rangle + pe_i + w_i\}_{i=1}^{N} \in \mathbb{Z}_q^N$$

and the obfuscation step is represented as

$$\left[\mathbf{a}, \{\langle \mathbf{a}, \mathbf{s_i} \rangle + pe_i + w_i\}_{i=1}^{N}\right].$$

2.1.4 Generation of a token for a specific input

Generate tokens for data $(x_1, \ldots, x_N) \in \mathbb{Z}_p^N$ as $\mathbf{t} := \sum\limits_{i=1}^{N} x_i \mathbf{s_i} \in \mathbb{Z}_q^n$. For each distinct data input, a separate token needs to be generated.

2.1.5 Evaluation

Compute

$$\sum_{i=1}^{N} w_i x_i = \sum_{i=1}^{N} c_i x_i - \langle \mathbf{a}, \mathbf{t} \rangle \, (\mathrm{mod} p) \, .$$

Indeed,

$$\sum_{i=1}^{N} c_i x_i - \langle \mathbf{a}, \mathbf{t} \rangle = \sum_{i=1}^{N} x_i \cdot \langle \mathbf{a}, \mathbf{s_i} \rangle + p \cdot \sum_{i=1}^{N} x_i e_i + \sum_{i=1}^{N} w_i x_i - \langle \mathbf{a}, \sum_{i=1}^{N} x_i \mathbf{s_i} \rangle = p \cdot \sum_{i=1}^{N} x_i e_i + \sum_{i=1}^{N} w_i x_i,$$

where the first term is eliminated after applying $\mathrm{mod} p$.

2.1.6 Correctness constraint

For the evaluation to be correct, the following correctness constraint has to be satisified:

$$\left\| p \cdot \sum_{i=1}^{N} x_i e_i \right\|_\infty < q/4.$$

References

[1] M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Security of homomorphic encryption," HomomorphicEncryption.org, Redmond WA, Tech. Rep., July 2017.