Com Sci 282A: Foundations of Cryptography Prof. Rafail Ostrosky

Lecture 8

Lecture date: Monday, March 2, 2005 Scribe: Brian Chin, Nishali Mehta, Christine Zhang

1 Overview

Definition 1 Homomorphic Encryption *is a semantically-secure public-key encryption which, in addition to standard guarantees has the additional property that given any two encryptions $E(A)$ and $E(B)$, there exists an encryption $E(A * B)$ such that*

$$E(A) * E(B) = E(A * B)$$

where $*$ is either addition or multiplication (in some abelian group).

Homomorphic encryption systems can be very useful because it allows a third party to operate on encrypted values without knowing the plaintext. Thus it can provide a setting for operation on encrypted values by someone else such that only the person who knows the key can decrypt the result.

In this lecture, we will introduce some protocols that were achieved by using homomorphic properities.

2 Homomorphic Encryption Schemes

2.1 EL Gamal Scheme

El Gamal is actually a homomorphic encryption whose binary operation is multiplication. To show how the El Gamal Cryptosystem is homomorphic, we have to first recall the mechanism used to encrypt a message. The public key is the tuple (G, q, g, y) where G is a group, q is the order of the group, g is a generator of that group, and $y = g^x \bmod q$ for some secret key x. To encrypt a message m, the pair (c_1, c_2) is generated, where:

$$c_1 = g^k$$

$$c_2 = y^k \cdot m$$

Fact 2 *Given two encrytions in the El Gamal cryptosystem $E(m_1) = (a_1, b_1)$ and $E(m_2) = (a_2, b_2)$, $(a_1 \cdot a_2, b_1 \cdot b_2)$ is an encryption of the product $m_1 \cdot m_2$.*

Proof

If we multiply two messages componentwise, we get

$$(g^{k_1}, y^{k_1} \cdot m_1) \cdot (g^{k_2}, y^{k_2} \cdot m_2) = (g^{k_1+k_2}, y^{k_1+k_2} \cdot m_1 \cdot m_2) = (g^{r'}, y^{r'} \cdot (m_1 \cdot m_2)).$$

■

2.2 Goldwasser-Micali (GM) Public Key Encryption Scheme

This system is concerned with whether, for a given number x, there is an w with $w^2 \equiv \alpha (\bmod N)$ If an w exists then x is called a *quadratic residue* (QR). Otherwise, x is a *quadratic nonresidue* (QNR).

To generate a key, we pick two large prime numbers p and q of equal length. The public key for this system becomes N, the product of p and q, while the private key is simply (p, q). For the purposes of this encryption scheme, we will only consider values in the set $Z_N^{+1} = \{x | 1 \leq x \leq N, \gcd(N, x) = 1, (\frac{x}{N}) = +1\}$, where $(\frac{x}{N})$ is the Jacobi symbol. The problem of discovering if a given number x is a QR modulo N, where N is the product of two large primes and $(\frac{x}{N}) = 1$, is considered to be a hard problem.

In addition, we constrain the primes p and q such that $p \equiv 3 (\bmod 4)$ and $q \equiv 3 (\bmod 4)$. This property guarantees that -1 will always be a QNR of Z_N^{+1}.

Messages are encrypted bit by bit. For each bit, we want to generate a QR if the bit is zero, and a QNR if the bit is one. To create an arbitrary QR, we simply take some random number $r \in Z_N^{+1}$, and square it modulo N. To generate a QNR, we simply multiply a random QR (calculated the same as above) by a known QNR (as stated above, -1). This gives us our encryption method.

Finding if x is a QR modulo N when the prime factors of N are known is an easy problem. This gives us a mechanism for decryption, given the private key.

Fact 3 *If we multiply the encryptions of two bits A and B the result is an encryption of the exclusive or of A and B.*

$$E(A) \cdot E(B) = E(A \oplus B)$$

where $\oplus$ is the XOR operaton.

This GM scheme is as secure as the assumption that distinguishing Quadratic Residues from Quadratic Non Residues is hard in Z_N^{+1}.

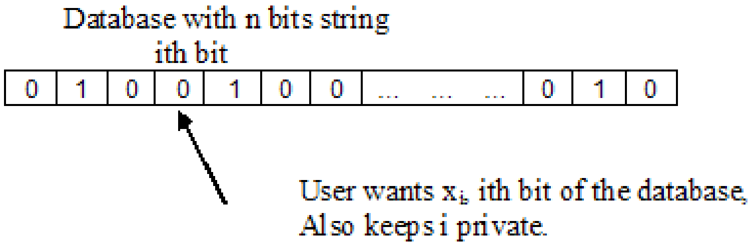

Figure 1: Private information retrival scheme

3 Applications of Homomorphic Encryption

3.1 Private Information Retrieval (PIR) Scheme

Introduction of PIR

Suppose we have a database DB made up of n bits. A user u wants to retrieve the ith bit x_i, but wants i to remain secret from DB with as little communication as possible. In this case, DB should not be able to distinguish between any two indicies i and j.

Naive Solution

There are at several unsatisfactory approaches to the PIR problem. They fail to solve the real-world problem but they point us to the properties, that the practical PIR solution must have.

- *Entire Database Download:* One possible solution is for the database administrator to send the entire database to the user. The client can process queries on the local copy of the database. Thus, the server is unaware of the user queries' content, and consequently, the server is unaware of the user preferences. This certianly hides all information about the query from the database administrator.

  This approach is not practical because of the great cost to the user in storing the database. An additional cost is a communication, which is equal to the size of the database.

- *Request a Subset* The user can request the database to send a random subset that contains the ith bit. While this method requires less costs, it is not secure, as the database can then distinguish between the index i and any index j that was not requested.

- *Anonymization techniques.* Using a traffic anonymization technique, a user can anonymously send queries to a server and anonymously receive the answers. One might think this is a PIR solution. It is not, since the server can still gather some general statistics on the user requests. For example, the server can trace which record has been accessed more than others or which have not been accessed at all. Thus this does not solve our problem either.

Single Database PIR

Theorem 4 *Assume that the GM cryptosystem is secure. There eixsts a PIR protocol with comunication complexity* $\leq \sqrt{n} \cdot k$, *where* n *is the number of bits in the database, and* k *is the security parameter of GM.*

Proof

First, we use the GM public key cryptosystem to generate a public key and secret key. We can use the public key to encrypt individual bits, and we can only decrypt the bits if we have the secret key.

Using the following special homomorphic XOR-properties of GM (or in fact any other XOR homomorphic encryption), we will show how an user can query database while hiding the identity of the data-items she is after.

- $E(0) * E(1) = E(1)$
- $E(1) * E(0) = E(1)$
- $E(1) * E(1) = E(0)$
- $E(0) * E(0) = E(0)$

Partition DB into $\sqrt{n}$ columns of length $\sqrt{n}$ bits each. For each column j, where $1 \leq j \leq \sqrt{n}$, the user sends a pair of values (a_j, b_j). Let i be the column that user is interested in. For column i, the user sends $(E(0), E(1))$ to the server. For every other column, it sends $(E(0), E(0))$.

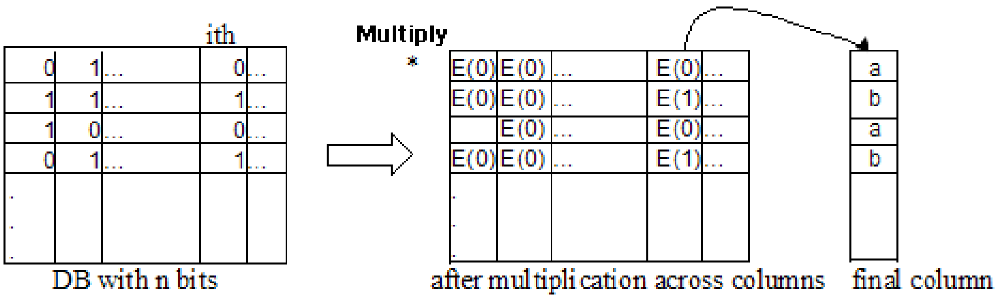

Figure 2: with n bits that contains $\sqrt{n}$ x$\sqrt{n}$ columns

Since the encryption is semantically secure, the DB can not distinguish between $E(0)$ and $E(1)$. Therefore, the DB can not tell the difference between any a_j and b_j. For the each column j of the DB, the DB replaces each 0 with a_j and each 1 with b_j. As a consequence, for every column other than i, the encrypted column is made up entirely of $E(0)$'s.

For each row, multiply the encrypted value in every column together, creating a new column. This results in an encryption of only the ith column due to the above XOR properties of GM. The DB sends this encryption to the user, who can then decode the result, thus obtaining the ith column which contains the bit he was interested in. The DB thus has no idea which column the user was interested in, yet the user still obtains the desired bit.

The procedure is as shown in Figure 2.

■

PIR and Hash Functions

Theorem 5 *Any single-database PIR query of a random location i is a collision resistant hash function.*

Proof

Suppose an adversary can find a collision. Then two different databases, DB_1 and DB_2, must have the same answer at the same location.

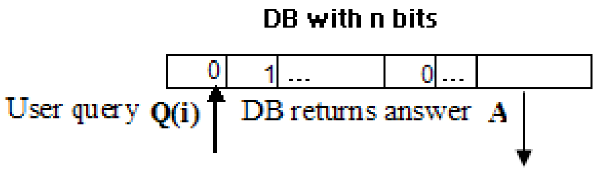

Figure 3: An illustration for a PIR query

Suppose we have the two database, DB_1 and DB_2 such that

$$Q(i)[DB_1] = x \qquad Q(i)[DB_2] = x.$$

they must have the same answer at location i, yet must be different at some other location. Thus any such pair gives partial information about the index i. This contradicts the semantic security of the PIR scheme (i.e. distinguishes i from any other $j \neq i$), and thus disproves the assumption that an adversary can find colliding databases DB_1 and DB_2.

■

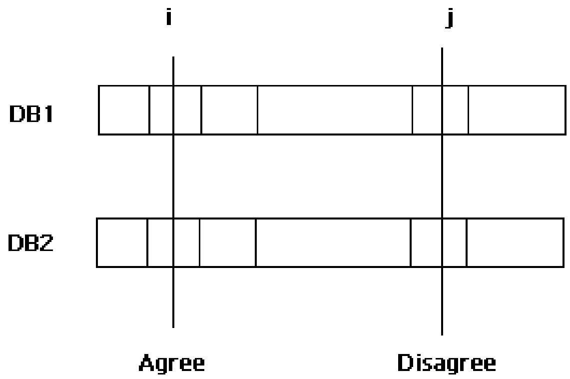

Figure 4: Two difference DB gives the same asnwer at ith position, but disagree at jth position.

3.2 Voting System

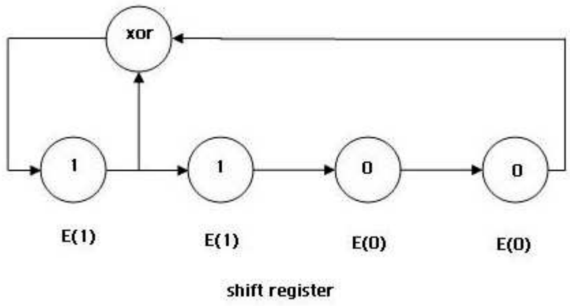

Figure 5: Homomorphic Encryption Application on Voting System with shift register

Another homomorphic encryption application is the electronic voting system. Suppose there is a virtual "whiteboard" where every voter can write in his vote (for simplicity, we assume only one voter at a time). Let's say that we just have yes or no votes, and we want to count all the yes votes Thus every user, one at a time, comes to this white board to write his vote. If his vote is no, he wants to maintain the count. If his vote is yes, he wants to increment the counter by one. We need a way to either add one or leave the value the same after each new vote is cast, without any outsider being able to see what change was made, and without any voter being able to tell the result (until all the votes are cast). We ignore how the counter is decrypted in the end by some trusted party.

We turn to the GM encryption scheme as a basis. We encode the counter as a vector of encrypted bits that is initially set to $E(0)$ at each position. To leave the counter's value the same, we multiply each bit by an encryption $E(0)$. This effectively scrambles the encryption of all of the bits without changing their plaintext values.

To add one using only XORs we turn to a shift register. A shift register is a a simple logical circuit containing k bits which can sequentially cycle through all 2^k possible values by applying the same function over and over. It is implemented using only simple bit movements and the application of XOR gates. Thus a shift register of length k can be used to implement a counter of values from 0 to $2^k - 1$, and can be implemented securely by using GM. This can be done so no one can distinguish between the above scrambling procedure and an application of the shift register. A general picture of a shift register can be found in figure 5.

A person votes negatively by scrambling the values as in the first case, and positively by applying the shift register as in the second case. Since the two operations are indistinguishable, an adversary cannot tell how any individual voted, even if they have the encrypted text before and after their vote. The final tally can be obtained by a person with the private key, by simply decrypting the counter and reading its value.

This particular implementation does not account for other issues that normally come up in voting systems, such as preventing a person from voting multiple times.

4 Open Problem

We do not know how to construct a semantically secure encryption scheme which is fully homomorphic. That is:

$$E(x) \oplus E(y) = E(x + y)$$

$$E(x) \otimes E(y) = E(x \cdot y)$$