6.892 Computing on Encrypted Data September 09, 2013

Lecture 1

Lecturer: Vinod Vaikuntanathan *Scribe: Madars Virza*

1 Introduction

Computing environments change: everyone has portable computing devices (in form of mobile phones) and access to large servers (in the cloud). This change presents fundamental challenge of *outsourcing* computation, which is motivated by the asymmetry of the available computing power.

In recent computing scenarios clients are trusted (but weak), while computationally strong servers are untrusted as we do not exhibit full control over them. How to outsource (delegate) the computation? What about privacy of the outsourced computation? For example, how to outsource computing on medical data, which must be kept confidential at all times? Standard solution would be to encrypt the data: this perfectly solves any privacy issues. However, requirements for standard encryption schemes (in particular, non-malleability) also do not let us achieve the wanted functionality: we cannot perform *any* computations on the encrypted data.

2 Homomorphic encryption

A solution to this problem is **homomorphic encryption**, which permits computing on encrypted data. That is, the client can encrypt his data x and send the encryption $\mathsf{Enc}(x)$ to the server. The server can then take the ciphertext $\mathsf{Enc}(x)$ and evaluate a function f on the underlying x obtaining the *encrypted* result $\mathsf{Enc}(f(x))$. The client can decrypt this result achieving the wanted functionality, but the server learns nothing about the data that he computed on.

The notion of homomorphic encryption was first introduced by Rivest, Adleman and Dertouzos in [RAD78] and was first instantiated for restricted classes of functions f, like addition or multiplication. Until recently obtaining homomorphic encryption scheme that would support any functionality — Fully Homomorphic Encryption (FHE) — was a major open problem. In his breakthrough work [Gen09] Gentry gave the first construction of a FHE scheme, however his construction was very complex. Since then fully-homomorphic encryption has been area of very active research and in past few years considerably simpler FHE schemes have been discovered [BV11].

3 Related cryptographic notions

So far we have been concerned about the privacy of client's input x (i.e. **client privacy**). However, there are other security properties that we might also like to ensure. We now proceed to briefly describe some of them:

Verifiable delegation. How can we ensure that the encrypted result we get is really $\mathsf{Enc}(f(x))$ not $\mathsf{Enc}(f'(x))$ for some other function f'? This motivation is very practical, as the cloud has economic incentives to do less work and, presumably, computing the wrong function would be easier for him. Verifiable delegation denotes the property that server evaluated the function f client wanted him to evaluate.
Function privacy. How to protect f? It might be the case that our encryption scheme does not leak anything about x but as a side effect reveals the description of f. This is of concern if the function is proprietary; for example, Google's search algorithm. In this setting we assume that server is honest, but curious.

Server privacy. The computation to be evaluated could take an auxiliary input (e.g. witness w for an **NP** statement) and we would like the $\mathsf{Enc}(f(x, w))$ leaks nothing about this input. Note that functions and inputs are interchangeable and function privacy can be viewed as a special case of server privacy, where f is the universal circuit and the auxiliary input is the real algorithm to be executed.

We have theoretically satisfiable solutions to achieve all of the notions described above, but bridging the gap between theory and practice is an active research area (and motivation for implementation projects in our course!).

4 Functional encryption

A closely related notion to homomorphic encryption is **functional encryption**, where our goal is to reveal the result of the computation to the server, but protect all other information about our encrypted input.

For a motivating example consider the problem of spam filtering for encrypted email without interacting with the client. Then the function f we would like to evaluate would be a classifier that sorts e-mails as either "spam" or "not spam". If we had used homomorphic encryption, the server would learn the encrypted bit (spam/not-spam) $\mathsf{Enc}(f(x))$, but it would be useless for him to actually sort e-mail messages. (We obviously do not want to give the server our decryption key, because it would allow him to read $\mathsf{Enc}(x)$ itself!) Instead we would like to allow the server to evaluate $\mathsf{Enc}(x) \to f(x)$ but just for the particular function f and nothing else (e.g. we don't want the server to compute this for $f(x) = x$).

Functional encryption captures this notion: in the setup phase user generates a master secret key msk, which she can later specialize into secret keys sk_f for individual functions f, using the key generation algorithm KeyGen. Later anyone who possesses sk_f and $\mathsf{Enc}(x)$ can compute $f(x)$ as desired, but cannot learn anything more about x.

5 Program obfuscation

Another problem we might want to solve is how to give out copies of a program P without revealing the algorithms used in P. That is, we would like to get **obfuscated** copies of P^* that are as useful to the user (who can inspect the code of P^*) as black-box access to P. There are many impossibility results for program obfuscation, but we also have constructions for obfuscating restricted classes of programs. We will hopefully cover this topic at the end of the class.

6 Multi-party computation

Consider the problem of secure multiparty computation (MPC): multiple users want to jointly compute the result of a function $f(x_1, \ldots, x_n)$ where each of user would supply the corresponding input x_i. Here we want everyone to learn the result of the function but the computation to reveal nothing else. This problem has been extensively studied and we have many great completeness theorems [Yao86, GMW87, BGW88].

However all of the solutions are very egalitarian: they treat users as having equal computational capabilities. How can we achieve MPC in the world where computational power is asymmetric (e.g. cloud vs mobile device)? We will cover this topic later in the class and introduce the necessary techniques as we go.

7 Public-key encryption and homomorphic encryption

We now proceed to formally define homomorphic encryption. We first recall the definition of public key encryption:

Definition 1. *A public key encryption scheme is a triple of probabilistic polynomial time algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$*, such that:*

- *The key generation algorithm* Gen *takes as an input the security parameter* 1^λ *and outputs a pair of keys* (pk, sk)*, i.e. public key* pk *and private key* pk*.*
- *The encryption algorithm* Enc *takes as an input the public key* pk *and message* μ *and outputs the ciphertext* c*.*
- *The decryption algorithm* Dec *takes as an input the private key* sk *and ciphertext* c *and outputs a message* μ *(or* $\perp$ *indicating failure).*

We require that:

(Correctness.) *For every message* μ *we have:*

$$\Pr\left[\mu^* = \mu \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathsf{Gen}(1^\lambda) \\ c \leftarrow \mathsf{Enc}(pk, \mu) \\ \mu^* \leftarrow \mathsf{Dec}(sk, c) \end{array}\right] = 1 - \mathrm{negl}(\lambda) \ .$$

Where the probability is taken over the randomness used by Gen*,* Enc *and* Dec*.*

(Security.) *For every probabilistic polynomial time adversary* $\mathcal{A}$ *and every sufficiently large security parameter* λ*, the probability of adversary winning in the following security game is at most* $\frac{1}{2} + \mathrm{negl}(\lambda)$*.*

The **(IND-CPA security game.)** *is defined as follows:*

- *The challenger first generates* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ *and sends pk to the adversary;*
- *The adversary* $\mathcal{A}$ *outputs two messages* m_0*,* m_1*.*
- *Challenger chooses a random bit* $b \leftarrow \{0, 1\}$ *and sends* $c^* \leftarrow Enc(sk, m_b)$ *to the adversary;*
- *Adversary outputs his guess* b' *of which message he received and wins if* $b' = b$*.*

As usual, we call a function f *negligible* i.e. $f(x) \in \mathrm{negl}(x)$ if for all polynomials $p(\cdot)$ there exists x_0 such that for all $x \geq x_0$ we have $f(x) < 1/|p(x)|$; this captures the fact that adversary can always win by negligibly small probability (e.g. $1/2^{O(\lambda)}$) by just guessing the secret key, but that (essentially) an efficient adversary cannot do anything more.

We note that our notion of security (ciphertext indistinguishability) implies semantic security: that given pk and ciphertext c the adversary cannot learn any partial information about the message μ itself [GM82].

For homomorphic encryption we add an additional PPT algorithm Eval[1] that performs the homomorphic evaluation. That is, Eval will take as an input the public key pk, a function f and ciphertexts $c_1, \ldots, c_l$ and will return another ciphertext $c^* \leftarrow \mathsf{Eval}(f, c_1, \ldots, c_l)$.

Definition 2. *We say that encryption scheme is* $\mathcal{C}$**-homomorphic** *(where* $\mathcal{C}$ *is the class of functions supported) if for all* $\mu_1, \ldots, \mu_l$ *and* $f \in \mathcal{C}$ *we have:*

$$\Pr\left[\mu^* = f(c_1, \ldots, c_l) \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathsf{Gen}(1^\lambda) \\ c_1 \leftarrow \mathsf{Enc}(pk, \mu_1), c_2 \leftarrow \mathsf{Enc}(pk, \mu_2), \ldots, c_l \leftarrow \mathsf{Enc}(pk, \mu_l) \\ c \leftarrow \mathsf{Eval}(pk, f, c_1, \ldots, c_l) \\ \mu^* \leftarrow \mathsf{Dec}(sk, c) \end{array}\right] = 1 - \mathrm{negl}(\lambda) \ .$$

If $\mathcal{C}$ *is the class of all functions we call the encryption scheme* **fully-homomorphic***.*

Our definition of IND-CPA security does not need to change: as Eval is a public process the adversary can perform homomorphic evaluations himself without the help of the challenger. Looking ahead, we cannot use the IND-CCA game (where the adversary is allowed to make decryption queries on ciphertexts other than c^*): homomorphism is antithetical to access to a decryption oracle. For example, think of $f(x) = x + 1$ — adversary can just mall the challenge ciphertext into one for $\mu_b + 1$ and recover μ_b from the decryption of it. But we will talk more about this later in the class.

[1] Eval needs to be probabilistic if we want function privacy, but can be deterministic if we only care about client privacy.

8 Examples of limited homomorphism

Many popular public key encryption schemes enjoy some sort of homomorphism. In this section we will look at familiar examples of RSA, Goldwasser-Micali, Paillier and El Gamal encryption schemes and see their homomorphic properties.

8.1 RSA encryption scheme

RSA encryption scheme introduced by Rivest, Shamir and Adleman [RSA78] has multiplicative homomorphism. Recall that RSA cryptosystem works like this:

- To generate a public key/secret key pair Gen chooses two large primes p and q and sets $N = p \cdot q$. Gen also chooses an integer e coprime to $\phi(N) = (p-1)(q-1)$. The public key pk is (N, e) and the secret key sk is (p, q). We note that given p and q it is easy to calculate $d = e^{-1} \pmod{\phi(N)}$.
- The set of messages consist of all elements of Z_N^*. An encryption of m is $c = m^e \pmod N$. It is easy to verify that $\mathsf{Dec}(sk, c) = c^d \pmod N$ is a valid decryption algorithm, as $m^{ed} \equiv m \mod N$.

To verify that multiplication of RSA ciphertexts gives the ciphertext that corresponds to the message of multiplied plaintexts we observe that:

$$\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) = m_1^e m_2^e = (m_1 m_2)^e = \mathsf{Enc}(m_1 \cdot m_2)$$

However the "textbook RSA" as described above is not semantically secure, as it is not randomized (so, for example, one can check equality).

To obtain semantically secure RSA for bit encryption we can pick a random $r \in Z_N^*$ and set the ciphertext c to be $(r^e, \mathsf{lsb}(r) \oplus b)$. Then to decrypt, we first decrypt the random r and XOR back the b from the $\mathsf{lsb}(r)$ and the second part of the ciphertext. It can be proved that least-significant bit of RSA encrypted plaintext is unpredictable [CG85] and therefore this scheme achieves semantic security.

Unfortunately, it seems that in the process we have lost the multiplicative homomorphism. Moreover, all the ways that we know how to make RSA semantically secure destroy the homomorphic properties. This motivates the following open problem:

Open Problem 1. *Construct semantically secure and additively (or multiplicatively) homomorphic encryption scheme based on the RSA assumption (that taking roots modulo composite is hard).*

8.2 Goldwasser-Micali encryption scheme

Goldwasser-Micali encryption scheme [GM82] is based on the quadratic residuosity assumption: that it is hard to distinguish quadratic residues with Jacobi symbol 1 from quadratic non-residues with Jacobi symbol 1 (all modulo a composite). Goldwasser-Micali encryption scheme works like this:

- To generate a public key/secret key pair Gen chooses two primes p and q and sets $N = p \cdot q$. Gen also chooses a random y that is a non-square modulo p and non-square modulo q. Gen returns the public key $pk = (N, y)$ and the private key $sk = (p, q)$.
- To encrypt a bit b Enc will choose a random r and return $c = r^2 \cdot y^b \pmod N$. The ciphertext c is non-square modulo p and non-square modulo q if $b = 0$ and square modulo both otherwise and it is easy to distinguish between the two cases given the factorization of N (the secret key). Therefore decryption is well-defined. It can also be proved that Goldwasser-Micali encryption scheme achieves semantic security under the quadratic residuosity assumption.

It is not hard to prove that Goldwasser-Micali scheme has additive homomorphism over $\mathbb{Z}_2$. That is $\mathsf{Enc}(b) \cdot \mathsf{Enc}(b') = (r \cdot r')^2 \cdot y^{b+b'}$, which is identically distributed to $\mathsf{Enc}(b \oplus b')$.

8.3 Additive homomorphism not in $\mathbb{Z}_2$

The RSA and Goldwasser-Micali encryption schemes had multiplicative and additive homomorphisms over $\mathbb{Z}_2$. However, in some applications it would be useful to have homomorphism over a larger ring. For example, homomorphic addition could serve as a tallying procedure for a voting scheme; a similar need also arises in database operations [PRZB11].

There exist schemes that support such homomorphisms, for example, the Paillier encryption scheme uses n-th residuosity assumption and generalizes Goldwasser-Micali encryption scheme, achieving additive homomorphism in $\mathbb{Z}_n$. Similarly El Gamal encryption scheme [EG85] allows one to perform multiplications in $\mathbb{Z}_p^*$.

In general we would like to support arbitrary number of additions and multiplications as we can implement any circuit using just XOR (addition) and AND (multiplication) gates. One idea would be using both representations: one that supports multiplication and one that supports addition, and isomorphisms between them (so we can multiply the addition results (or similar)). It turns out that such idea is workable [GH11]; its security is based on Learning with Errors assumption, which we will survey next.

9 Learning with Errors (LWE) problem

The Learning with Errors (LWE) problem is concerned about recovering a secret $s \in \mathbb{Z}_q^n$ given a collection of "noisy" linear equations on s. Fix size n, modulus q and error distribution χ on $\mathbb{Z}_q$. For a secret vector $s \in \mathbb{Z}_q^n$ consider the two distributions:

- **LWE distribution**, which is sampled by choosing $a \in \mathbb{Z}_q^n$ uniformly at random, drawing e from χ and returning $(a, \langle a, s\rangle + e)$;
- **"Random" distribution**, is sampled by choosing $a \in \mathbb{Z}_q^n$ uniformly at random, $r \in \mathbb{Z}_q$ uniformly at random and returning (a, r)

The **LWE search problem** is the computational problem of recovering s given polynomially many samples from the LWE distribution. The **decisional LWE problem** is the computational problem of distinguishing whether samples come from LWE distribution or are randomly distributed, given that all of them come from the same distribution.

The two problems are formalized via security games. For the LWE search problem the challenger first chooses a random s and then provides the adversary with the oracle access to the corresponding LWE distribution. The adversary wins if he outputs the same s challenger chose at the beginning of the game. We say that class of parameters (n, q, χ) is hard if the winning probability of the adversary is negligible.

For decisional LWE problem a challenger chooses a random s and a random bit b and then answers all oracle queries of the adversary by either sampling the corresponding LWE distribution if $b = 0$ or the "random" distribution if $b = 1$. The challenger wins if his final output is bit b' and $b = b'$. We say that class of parameters (n, q, χ) is hard if the challenger cannot win with probability more than $\frac{1}{2} + \text{negl}$.

The decision problem is easily reducible to the search problem, however it turns out that both are in fact equivalent. We will prove this in the next lecture.

If the error term didn't exist it would be easy to solve the LWE search problem and recover s by just performing the Gaussiam elimination. However Gaussian elimination is not robust to errors and for certain classes of (n, q, χ) we believe that recovering s from the LWE samples is a very hard problem. Assume that the error distribution χ is B-bounded, that is $\Pr_{x \leftarrow \chi}[|x| \leq B] = 1 - \text{negl}$ and consider the ratio q/B. If $q/B = 2^k$, then the best known algorithms for the LWE search problem run in time $2^{n/k} = 2^{n/log(q,B)}$, which is exponential for suitable choice of parameters.

References

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, 1988.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 97–106, 2011.

[CG85] Benny Chor and Oded Goldreich. RSA/Rabin least significant bits are $\frac{1}{2} + 1/\mathrm{poly}(\log n)$ secure. In *Proceedings of CRYPTO '84*, pages 303–313, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[EG85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, 2009.

[GH11] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT '11, pages 129–148, 2011.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, 1987.

[PRZB11] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Symposium on Operating Systems Principles (SOSP)*, Cascais, Portugal, October 2011.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, pages 169–177, 1978.

[RSA78] Ronald L. Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of ACM*, 21(2):120–126, February 1978.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, 1986.