Lecture 15: Fully homomorphic encryption : Introduction and bootstrapping

Boaz Barak

In today's era of "cloud computing", much of individual's and businesses' data is stored and computed on by third parties such as Google, Microsoft, Apple, Amazon, Facebook, Dropbox and many others. Classically, cryptography provided solutions to protecting data in motion from point A to point B. But these are not always sufficient to protect data at rest and particularly data in use. For example, suppose that Alice has some data $x \in \{0,1\}^n$ (where in modern applications x would well be terabytes in length or larger) that she wishes to store with the cloud service Bob, but is afraid that Bob will be hacked, subpoenaed or simply does not completely trust Bob.

Encryption does not seem to immediately solve the problem. Alice could store at Bob an *encrypted* version of the data and keep the secret key for herself. But then she would be at a loss if she wanted to do with the data anything more than retrieving particular blocks of it. If she wanted to outsource computation to Bob as well, and compute $f(x)$ for some function f, then she would need to share the secret key with Bob, thus defeating the purpose of encrypting the data in the first place. For example, after the computing systems of Office of Personell Management (OPM) were discovered to be hacked in June of 2015, revealing sensitive information, including fingerprints and all data gathered during security clearance checks of up to 18 million people, DHS assistant secretary for cybersecurity and communications Andy Ozment said that encryption wouldn't have helped preventing it since "if an adversary has the credentials of a user on the network, then they can access data even if it's encrypted, just as the users on the network have to access data". So, can we encrypt data in a way that still allows some access and computing on it?

Already in 1978, Rivest, Adleman and Dertouzos considered this problem of a business that wishes to use a "commercial time-sharing service" to store some sensitive data. They envisioned a potential solution for this task which they called a privacy homomorphism. This notion later became known as *fully homomorphic encryption* which is an encryption that allows a user that *does not know the secret key* to modify a ciphertext c encrypting x to a ciphertext c' encrypting $f(x)$ for every efficiently computable $f()$. In particular in our scenario above, such a scheme will allow Bob, given an encryption of x, to compute the encryption of $f(x)$ and send it to Alice without ever getting the secret key and so without ever learning anything about x (or $f(x)$ for that matter).

Unlike the case of a trapdoor function, for more than 30 years cryptographers had no constructions achieving this goal. In fact, some people suspected that there is something inherently incompatible between the security of an encryption scheme and the ability of a user to perform all these operations on ciphertexts. Stanford cryptogapher Dan Boneh used to joke to incoming graduate students that he will immediately sign the thesis of anyone who came up with a fully homomorphic encryption. But never expected that he will actually encounter such a thesis, until in 2009, Boneh's student Craig Gentry released a paper doing just that. Gentry's paper shook the world of cryptography, and instigated a flurry of research results making his scheme more efficient, reducing the assumptions it relied on, extending and applying it, and much more. In particular, Brakerski and Vaikuntanathan managed to obtain a fully homomorphic encryption scheme based only on the *Learning with Error (LWE)* assumption we have seen before.

Although there is open source library, as well as other implementations, there is still much work to be done on the implementation front in order to turn FHE from theory to practice. For a comparable level of security, the encryption and decryption operations of a fully homomorphic encryption scheme are several orders of magnitude slower than a conventional public key system, and (depending on its complexity) homomorphically evaluating a circuit can be significantly more taxing. However, this is a fast evolving field, and already since 2009 significant optimizations have been discovered that reduced the computational and storage overhead by many orders of magnitudes. As in public key encryption, one would imagine that for larger data one would use a "hybrid" approach of combining FHE with symmetric encryption, though one might need to come up with tailor-made symmetric encryption schemes that can be efficiently evaluated (see here for the state of art on homomorphically evaluating AES which currently has is about six orders of magnitude slower than non-homomorphic AES computation). In these two lectures we will focus on the homorphically encryption schemes that are easiest to describe, rather than the ones that are most efficient (though the efficient schemes share many similarities with the ones we will talk about). As is generally the case for lattice based encryption, the current most efficient schemes are based on *ideal* lattices and on assumptions such as ring LWE or the security of the NTRU cryptosystem.[1]

> **Aside: verifiying computation:** To take the distance between theory and practice in perspective, it might be useful to consider the case of verifying computation. In the early 1990's researchers (motivated initially by zero knowledge proofs) came up with the no-

[1] As we mentioned before in lecture 12, as a general rule of thumb, the difference between the ideal schemes and the one that we describe is that in the ideal setting one deals with *structured* matrices that have a compact representation as a single vector and also enable fast FFT-like matrix-vector multiplication. This saves a factor of about n in the storage and computation requirements (where n is the dimension of the subspace/lattice). However, there can be some subtle security implications for ideal lattices as well, see e.g., here , here, here, and here.

tion of *probabilistically checkable proofs* which could yield in principle extremely succinct ways to check correctness of computation. These proofs can be thought of as "souped up" versions of NP completeness reductions and like these reductions, have been mostly used for *negative* results, especially since the initial proofs were extremely complicated and also included enormous hidden constants. However, with time people have slowly understood these better and made them more efficient (e.g., see this survey) and it has now reached the point where these results, if not fully practical, are at least nearly practical. Overall, constructions for verifying computation have improved by at least 20 orders of magnitude over the last two decades. (We will talk about some of these constructions later in this course.) If progress on fully homomorphic encryption is similar, then we can expect the road to practical utility to be very long, but there is some hope that it's not a "bridge to nowhere".

Defining fully homomorphic encryption

We start by defining *partially homomorphic* encryption. We focus on encryption for single bits. This is without loss of generality for CPA security (CCA security is anyway ruled out for homomorphic encryption- can you see why?), though there are more efficient constructions that encrypt several bits at a time.

Definition (Partially homomorphic encryption): Let $\mathcal{F} = \cup \mathcal{F}_\ell$ be a class of functions where every $f \in \mathcal{F}_\ell$ maps $\{0,1\}^\ell$ to $\{0,1\}$.
An *$\mathcal{F}$-homomorphic public key encryption scheme* is a CPA secure public key encryption scheme (G, E, D) such that there exists a polynomial-time algorithm $EVAL$ such that for every $(e, d) = G(1^n)$, $\ell = poly(n)$, $x_1, \ldots, x_\ell \in \{0,1\}$, and $f \in \mathcal{F}_\ell$ of description size $|f|$ at most $poly(\ell)$: $c = EVAL_e(1^n, f, E_e(x_1), \ldots, E_e(x_\ell))$ has length at most n and $D_d(c) = f(x_1, \ldots, x_\ell)$.

The requirement that $|c| \leq n$ is to rule out some trivial constructions where the ciphertext grows with the description of f (can you see why it's needed?). Note that by artificially increasing the randomness for the key generation algorithm, this is equivalent to the requiring that $|c| \leq p(n)$ as long as $p(\cdot)$ is a fixed polynomial that does not grow with ℓ or $|f|$.

A *fully homomomorphic encryption* is simply a partially homomorphic encryption scheme for the family $\mathcal{F}$ of *all* functions, where the description of a function is as a circuit (say composed of NAND gates, which are known to be complete).

Example: An XOR homomorphic encryption

It turns out that Regev's LWE-based encryption we saw in Lecture 12 (lattice based crypto) is homomorphic with respect to the class of linear (mod 2) functions.

Let us recall the LWE assumption and the encryption scheme based on it.

Definition: Let $q = q(n)$ be some function mapping the natural numbers to primes. The *$q(n)$-decision learning with error ($q(n)$-dLWE) conjecture* is the following: for every $m = poly(n)$ there is a distribution LWE_q over pairs (A, s) such that:

- A is an $m \times n$ matrix over $\mathbb{Z}_q$ and $s \in \mathbb{Z}_q^n$ satisfies $s_1 = \lfloor \frac{q}{2} \rfloor$ and $|As|_i \leq \sqrt{q}$ for every $i \in \{1, \ldots, m\}$.
- The distribution A where (A, s) is sampled from LWE_q is computationally indistinguishable from the uniform distribution of $m \times n$ matrices over $\mathbb{Z}_q$.

The *LWE conjecture* is that $q(n)$-dLWE holds for every $q(n)$ that is at most $poly(n)$. This is not exactly the same phrasing we used before, but can be shown to be essentially equivalent to it. Before we phrased the conjecture as recovering s from a pair (A', y) where $y = A's' + e$ and $|e_i| \leq \delta q$ for every i. We then showed a *search to decision* reduction demonstrating that this is equivalent to the task of distinguishing between this case and the case that y is a random vector. If we now let $\alpha = \lfloor \frac{q}{2} \rfloor$ and $\beta = \alpha^{-1}(\mod q)$, and consider the matrix $A = (-\beta y | A')$ and the column vector $s = \binom{\alpha}{s'}$ we see that $As = e$. Note that if y is a random vector in $\mathbb{Z}_q^m$ then so is $-\beta y$ and so the current form of the conjecture follows from the previous one. (To reduce the number of free parameters, we fixed δ to equal $1/\sqrt{q}$; in this form the conjecture becomes stronger as q grows.)

A linearly-homomorphic encryption scheme: We can describe the encryption scheme presented in class as:

- Key generation: Choose (A, s) from LWE_q where m satisfies $q^{1/4} \ll m \log q \gg n$.
- To encrypt $b \in \{0, 1\}$, choose $w \in \{0, 1\}^m$ and output $w^\top A + (b, 0, \ldots, 0)$
- To decrypt $c \in \mathbb{Z}_q^n$, output 0 iff $|\langle c, s \rangle| \leq q/10$, where for $x \in \mathbb{Z}_q$ we defined $|x| = \min\{x, q - x\}$.

Note that decryption succeeds since it amounts to $w^\top As + s_1 b$ and $|w^\top As| \leq m\sqrt{q} \ll q$. It turns out that this scheme is scheme above is homomorphic with respect to the class of *linear functions* modulo 2. Specifically we make the following claim:

Claim: For every $\ell \ll q^{1/4}$, there is an algorithm $EVAL_\ell$ that on input $c_1, \ldots, c_\ell$ encrypting bits $b_1, \ldots, b_\ell \in \{0, 1\}$, outputs a ciphertext c encrypying $b_1 \oplus \cdots \oplus b_\ell$.

Proof: The proof is quite simple. $EVAL$ will simply add the ciphertexts as vectors in $\mathbb{Z}_q$. If $c = \sum c_i$ then $\langle c, s \rangle$ will equal $\sum b_i \lfloor \frac{q}{2} \rfloor (\mod q)$ up to noise which is at most $\ell m \sqrt{q} \ll q$. Since $|\lfloor \frac{q}{2} \rfloor - \frac{q}{2}| < 1$ this equals (up to small noise) to $\lfloor (\sum b_i) \frac{q}{2} \rfloor$ which would be 0 if $\sum b_i$ is even and $\lfloor \frac{q}{2} \rfloor$ if $\sum b_i$ is odd. QED

Several other encryption schemes are also homomorphic with respect to linear functions, and over the years people have managed to go above it (e.g., to quadratic functions by Boneh, Goh and Nissim) but not significantly so.

Abstraction: A trapdoor pseudorandom generator.

It is instructive to consider the following abstraction (which we'll use in the next lecture) of the above encryption scheme. On input 1^n key generation algorithm outputs a vector $s \in \mathbb{Z}_q^m$ with $s_1 = \lfloor \frac{q}{2} \rfloor$ and a probabilistic algorithm G_s such that the following holds:

- Any polynomial number of samples from the distribution $G_s(1^n)$ is computationally indistinguishable from independent samples from the uniform distribution over $\mathbb{Z}_q^n$
- If c is output by $G_s(1^n)$ then $|\langle c, s \rangle| \leq n\sqrt{q}$.

Thus s can be thought of a "trapdoor" for the generator that allows to distinguish between a random vector $c \in \mathbb{Z}_q^n$ (that with high probability would satisfy $|\langle c, s \rangle| \geq q/10$) and an output of the generator. We use G_s to encrypt a bit b by letting $c \leftarrow_R G_s(1^n)$ and outputting $c + (b, 0, \ldots, 0)^\top$. In the particular instantiation above we obtain G_s by sampling the matrix A from the LWE assumption and having G_s output $w^\top A$ for a random $w \in \{0,1\}^n$, but we can ignore this particular implementation detail in the forgoing.

Note that this trapdoor generator satisfies the following stronger property: we can generate an alternative generator G' such that the description of G' is indistinguishable from the description of G_s but such that G' actually does produce (up to exponentially small statistical error) the uniform distribution over $\mathbb{Z}_q^n$.

> **Aside: trapdoor generators in real life:** In the above we use the notion of a "trapdoor" in the pseudorandom generator as a mathematical abstraction, but generators with actual trapdoors have arisen in practice. In 2007 the National Institute of Standards (NIST) released standards for pseudorandom generators. Pseudorandom generators are the quintessential private key primitive, typically built out of hash functions, block ciphers, and such and so it was surprising that NIST included in the list a pseudorandom generator based on public key tools - the Dual EC DRBG generator based on elliptic curve cryptography. This was already strange but became even more worrying when Microsoft researchers Dan Shumow and Niels Ferguson showed that this generator *could* have a trapdoor in the sense that it contained some hardwired constants that if generated in a particular way, there would be some information that (just like in G_s above) allows to distinguish the generator from random (see here for a 2007 blog post on this issue). We learned more about this when leaks from the Snowden document showed that the NSA secretly paid \$10 million to RSA to make this generator the default option in their Bsafe software. You'd think that this generator is long dead but it turns out to be the "gift that keeps on giving". In December of 2015, Juniper systems announced that they have discovered a malicious

> code in their system, dating back to at least 2012 (possibly 2008), that would allow an attacker to surreptitiously decrypt all VPN traffic through their firewalls. The issue is that Juniper has been using the Dual EC DRBG and someone has managed to replace the constant they were using with another one, one that they presumably knew the trapdoor for (see here and here for more). Apparently, as surprising as it is, inserting back doors into cryptographic primitives might end up making them less secure.

From linear homomorphism to full homomorphism

Gentry's breakthrough had two components:

- First, he gave a scheme that is homomorphic with respect to arithmetic circuits (involving not just addition but also multiplications) of *logarithmic depth*.
- Second, he showed the amazing "bootstrapping theorem" that if a scheme is homomorphic enough to evaluate its own decryption circuit, then it can be turned into a *fully homomorphic* encryption that can evaluate *any* function.

Combining these two insights led to his fully homomorphic encryption.[2]

In this lecture we will focus on the second component - the bootstrapping theorem. We will show a "partially homomorphic encryption" (based on a later work of Gentry, Sahai and Waters) that can fit that theorem in the next lecture.

Bootstrapping

We now show the following theorem:

Theorem (Bootstrapping Theorem, Gentry '09): Suppose that (G, E, D) is a CPA circular[3] secure partially homomorphic encryption scheme for the family $\mathcal{F}$ and suppose that for every pair of ciphertexts c, c' the map $d \mapsto D_d(c) \; NAND \; D_d(c')$ is in $\mathcal{F}$. Then (G, E, D) can be turned a fully homomorphic encryption scheme.

[2] The story is a bit more complex than that. Frustratingly, the decryption circuit of Gentry's basic scheme was just a little bit too deep for the bootstrapping theorem to apply. A lesser man, such as yours truly, would at this point surmise that fully homomprphic encryption was just not meant to be, and perhaps take up knitting or playing bridge as an alternative hobby. However, Craig persevered and managed to come up with a way to "squash" the decryption circuit so it can fit the bootstrapping parameters. Follow up works, and in particular the paper of Brakerski and Vaikuntanathan, managed to get schemes with much better relation between the homomorphism depth and decryption circuit, and hence avoid the need for squashing and also improve the security assumptions.

[3] You can ignore the condition of circular security in a first read - we will discuss it later.

Proof: The idea behind the proof is simple but ingenious. Recall that the NAND gate $b, b' \mapsto \neg(b \wedge b')$ is a universal gate that allows us to compute any function $f : \{0,1\}^n \rightarrow \{0,1\}$ that can be efficiently computed. Thus, to obtain a fully homomorphic encryption it suffices to obtain a function $NAND - EVAL$ such that $D_d(NAND - EVAL(c, c')) = D_d(c)\ NAND\ D_d(c')$. (Note that this is stronger than the typical notion of homomorphic evaluation since we require that $NAND - EVAL$ outputs an encryption of $bNANDb'$ when given *any* pair of ciphertexts that decrypt to b and b' respectively, regardless whether these ciphertexts were produced by the encryption algorithm or by some other method, including the $NAND - EVAL$ procedure itself.)

Thus to prove the theorem, we need to modify (G, E, D) into an encryption scheme supporting the $NAND - EVAL$ operation. Our new scheme will use the same encryption algorithms E and D but the following modification G' of the key generation algorithm: after running $(d, e) = G(1^n)$, we will append to the public key an encryption $c^* = E_e(d)$ of the secret key. We habe now defined the key generation, encryption and decryption. CPA security follows from the security of the original scheme, where by circular security we refer exactly to the condition that the scheme is secure even if the adversary gets a single encryption of the public key.[4] This latter condition is not known to be implied by standard CPA security but as far as we know is satisfied by all natural public key encryptions, including the LWE-based ones we will plug into this theorem later on.

So, now all that is left is to define the $NAND - EVAL$ operation. On input two ciphertexts c and c', we will construct the function $f : \{0,1\}^n \rightarrow \{0,1\}$ (where n is the length of the secret key) such that $f(d) = D_d(c)\ NAND\ D_d(c')$. It would be useful to pause at this point and make sure you understand what are the inputs to f, what are "hardwired constants" and what is its output. The ciphertexts c and c' are simply treated as fixed strings and are *not* part of the input to f. Rather f is a function (depending on the strings c, c') that maps the secret key into a bit. When running $NAND - EVAL$ we of course do not know the secret key d, but we can still design this function f. Now $NAND - EVAL(c, c')$ will simply equal $EVAL(f, c^*)$. Since $c^* = E_e(d)$, we get that $D_d(EVAL(f, c^*)) = D_d(f(d)) = D_d(D_d(c)\ NAND\ D_d(c'))$. Thus indeed we map *any* pair of ciphertexts c, c' that decrypt to b, b' into a ciphertext c'' that decrypts to $b\ NAND\ b'$. This is all that we needed to prove. QED

Don't let the short proof fool you. This theorem is quite deep and subtle, and requires some reading and re-reading to truly "get" it.

[4] Without this assumption we can still obtained a form of FHE known as a *leveled* FHE where the size of the public key grows with the depth of the circuit to be evaluated. We can do this by having ℓ public keys where ℓ is the depth we want to evaluate, and encrypt the private key of the i^{th} key with the $i + 1^{st}$ public key. However, since circular security seems quite likely to hold, we ignore this extra complication in the rest of the discussion.

Playing with radioactive Legos

Here is one analogy for bootstrapping, inspired by Gentry's survey. Suppose that you need to construct some complicated object from a highly toxic material. You are given a supply of sealed bags that are flexible enough so you can manipulate the object from outside the bag. However, each bag can only hold for 10 seconds of such manipulations before it leaks. The idea is that if you can open one bag inside another within 9 seconds then you can perform the manipulations for arbitrary length. That is, if the object is in the i^{th} bag then you put this bag inside the $i+1^{st}$ bag, spend 9 seconds on opening the i^{th} bag inside the $i+1^{st}$ bag and then spend another second of whatever manipulations you wanted to perform. We then continue this process by putting the $i+1^{st}$ bag inside the $i+2^{nd}$ bag and so on and so forth.