CS 198 Codebreaking
Spring 2023 Ryan Cottone, Imran Khaliq-Baporia

Note 10

Homomorphic Encryption

So far we have covered ways of securely encrypting data. Per IND-CPA definitions, this data should look "random" – so the idea of computing *using* encrypted data sounds like nonsense. It turns out that a special class of encryption algorithms allows for this sort of computation – *homomorphic encryption*.

The name "homomorphic" originates from abstract algebra, in which a **homomorphism** f between two groups (or rings/fields/etc) is a *structure-preserving mapping*, such that

$$f(x \cdot_a y) = f(x) \cdot_b f(y)$$

$\cdot_a$ is the group operation of the first group, and $\cdot_b$ is the second group operation. Intuitively, the mapping of the product of two elements is equal to the product of the mapping of two elements. This relates to cryptography when we consider the (group, for example) of plaintexts P and the ciphertext group C. Encryption is a mapping $E : P \to C$, and decryption its inverse. If the mapping is a homomorphism, then $E(a \cdot b) = E(a) \cdot E(b)$.

A consequence of this is that a third party, given only $E(a)$ and $E(b)$, can compute $E(a \cdot b)$!

The motivations for this may be slightly unclear at first. Let's consider a common example in today's world – you want to run a large computation, but lack the compute power to do so. To solve this, you want to outsource to a cloud provider like AWS. By using a homomorphic encryption scheme, you can send them the *encrypted* input data $E(a)$, they can compute $F(E(a)) = E(F(a))$, and you can decrypt the latter term to get $F(a)$.

Types of Homomorphic Encryption

Unfortunately, homomorphic encryption is tricky to build. Lots of schemes are homomorphic in *one* of addition or multiplication, but not both. Such schemes are named **partially homomorphic encryption**.

For example, consider the one-time pad. OTP is perfectly **additively homomorphic** $\mod 2$, since $E(m) = k \oplus m$. To add mod two (XOR), we simply perform the operation as normal: $E(m) \oplus b = k \oplus m \oplus b = k \oplus (m \oplus b) = E(m \oplus b)$. This can be expanded to one-time pad over any additive group, like addition $\mod N$.

Textbook RSA, on the other hand, is **multiplicatively homomorphic**. To multiply some RSA ciphertext $m^e \mod N$ by an integer k, we multiply by $k^e \mod N$: $k^e \cdot m^e \mod N \equiv (km)^e \mod N$. Notice we had to modify our input scalar a bit – this is acceptable in homomorphic encryption (and

often necessarily). Note that we can also find the product of two ciphertexts by multiplying them directly: $m_1^e \cdot m_2^e \equiv (m_1 m_2)^e \mod N$.

What, then, about a scheme that is both additively and multiplicatively homomorphic? This is named **fully homomorphic encryption**, or FHE, and is often considered the *holy grail* of cryptography. Since addition and multiplication $\mod 2$ suffice to describe a universal set of logic gates (namely AND by multiplying by 1 and NOT by adding 1 $\implies$ NAND), we can do **arbitrary computation over encrypted data!** Best of all, homomorphic encryption retains all of its underlying security, so it is still IND-CPA secure!

This extends as far as training machine learning models on encrypted data, like medical records. The applications for FHE are vast, but a key problem remains: FHE is **slow**.

Drawbacks of FHE

A key "downside" of FHE is that security must be preserved, which implies we cannot use programs that have control flow. By control flow, imagine something like binary search on an encrypted array. We are given a value k to search for in array A. At some step, we compare k to some value $A[i]$ – this is possible under FHE, but the result of $k \leq A[i]$ **is also encrypted**. Therefore, the server cannot use that result to decide which branch to take. Informally, we must simulate **all possible execution paths of a program** when working on encrypted data.

Note that we can still return the correct result of a search in an array – simply sum the result of an equality check. Note that this takes linear time (with a sizable constant due to FHE).

Another problem with FHE is the ciphertext size – the keys to encrypt 4 bits of data with a few carry bits can reach multiple gigabytes with modern libraries!

Bootstrapping

Finally, FHE is slow because it uses **bootstrapping**. bootstrapping involves *homomorphiclly evaluating the decryption circuit on itself and re-encrypting the ciphertext.* To understand why this is necessary, we note that current FHE relies on the Learning With Errors problem, which has some amount of error in each ciphertext. Each homomorphic operation increases this error, so we need to reset it by decrypting and re-encrypting. Of course, the server can't do that normally.

To fix this, we provide a *circular encryption* of the decryption key – $Enc(K,K)$. Whether this is secure is technically an open question. In any case, we can homomorphic evaluate the decryption circuit $Dec(K,C)$:

$$Dec(Enc(K),Enc(C)) = Enc(Dec(K,C))$$

This genius idea is what won Craig Gentry a host of awards as part of PhD thesis outlining the very first FHE scheme.

Secure Multiparty Computation

The motivating factor behind FHE is a problem known as **secure multi-party computation**, or SMPC. SMPC is most generally the problem of evaluating a function on multiple private inputs.

Consider the Millionaire's problem – we have Alice, Bob, and Charlie each with a secret amount of money ()m_a, m_b, m_c respectively). Each of them wants to know who has the *most* money ($\max m_a, m_b, m_c$), but do not want to reveal their secret amount to anyone else. Without access to a trusted third party, how might they accomplish this?

FHE provides a straightforward way – this is why SMPC is largely a special case of FHE (indeed, SMPC as a field has been developing since the 1970s, whereas FHE was first invented in 2009).

There are various ways of accomplishing SMPC, such as **oblivious computation**, **oblivious transfer**, **garbled circuits**, and **functional encryption**. All of which are very interesting schemes that are sadly out of scope for this course.