Ethics of Data Science – Part III

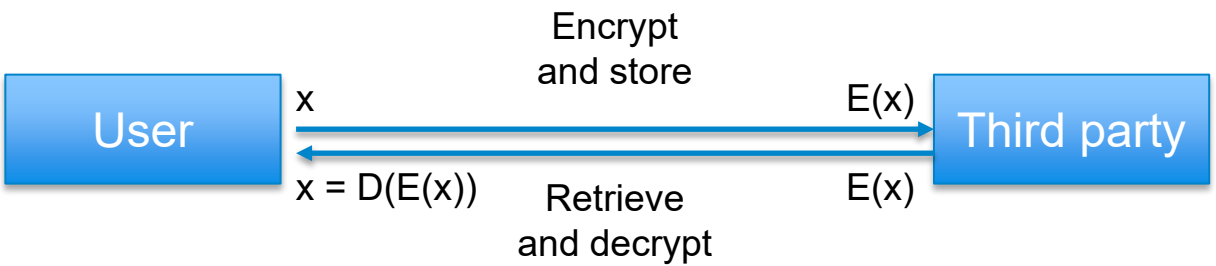
Week 3: Homomorphic Encryption

Polynomial ML - Fisher discriminants

Dr. Chris Anagnostopoulos, Hon. Senior Lecturer

Encryption – why?

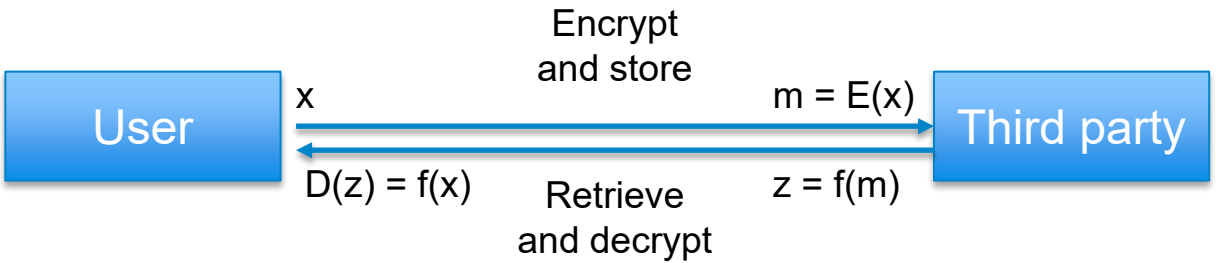
- Encrypting your data means that even if a third party gains access to it, they can't read it.
- This protects communications even when the channel is not secure.
- It also allows for *storage* of your data on non-secure third-party servers.



Malleability in encryption schemes

- For the same reason, that third party cannot perform analytics or machine learning on your data.
- **Malleable encryption schemes** are an exception, in that they allow the third party to compute a function of x, without decrypting it, i.e., given E(x), they can compute E(f(x)) for some function f.

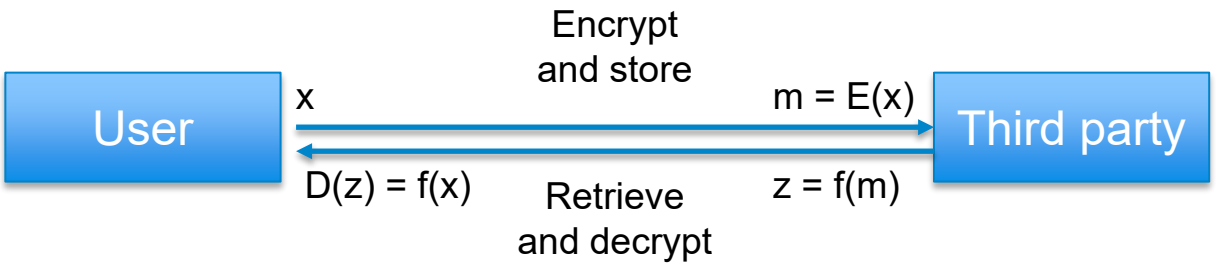
e.g., E(x)+b = E(x+b), E(2*x) = 2*E(x), E(x+y) = E(x) + E(y)



Malleability in encryption schemes

- Historically, malleability was considered a flaw, as it can allow an eavesdropper to modify a secure message in a predictable way, e.g., increase or decrease the amount of cash in a cash transfer.
- However, when there is a reason to out-source computation to a third party, malleability can help.

e.g., $E(x)+b = E(x+b)$, $E(2*x) = 2*E(x)$, $E(x+y) = E(x) + E(y)$



Fully homomorphic encryption

Let E(*x*) be the encryption of (the plaintext) *x*, and D(*m*) be the decrypted version of (the ciphertext) *m*. It is often said that E is homomorphic wrt to addition (resp. multiplication) if:

E(x + y) = E(x) + E(y) resp. E(x*y) = E(x)*E(y)

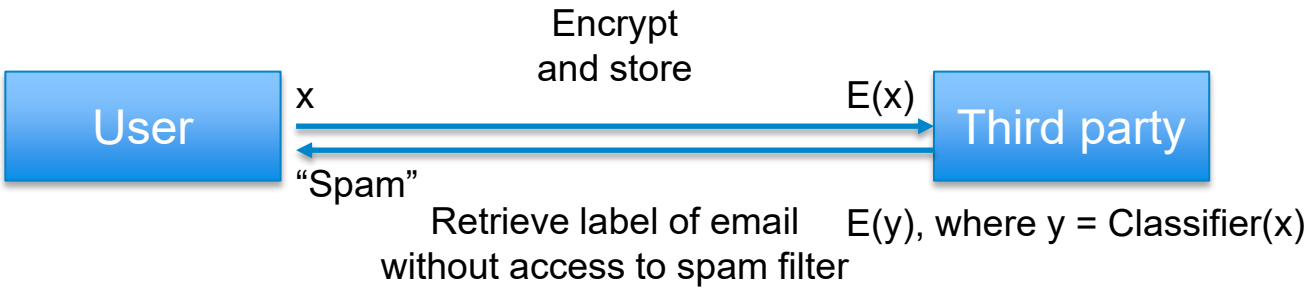
In truth, homomorphy may entail different operations between ciphertexts than between plaintexts. For example, in the *Pailler*[1] cryptosystem, multiplication of ciphertexts translates to addition of plaintexts.

Therefore it is best to think of FHE as a system *where there is a function he.add(x,y) and another function he.multiply(x,y) such that he.add(E(x), E(y)) = E(x+y), and resp. for multiplication*. We abuse the notation a little to also include addition between plaintexts and ciphertexts, e.g., we can write he.add(E(x), 2) = E(x+2), although likely the function would operate differently on the plaintext 2.

1: You don't need to know the details of any given crypto system, but if you have a maths background Pailler and RSA are relatively straightforward.

Fully homomorphic encryption – why?

- It is worth comparing FHE to the federated solution to this problem that also maintains the privacy of the user: decentralize the computation, i.e., send the model f(.) to the user.
- However, the third party might want to keep the model secret, e.g., because it is proprietary ML, or because it may be possible to leak the training data to the user if disclosed.
- FHE hence keeps f(.) secret to the third party, and x secret to the user, but lets the user have f(x). For example, your email provider keeps their spam filter secret, and you keep your emails secret.



Summary

- The basic premise of homomorphic encryption is that you can still apply operations on the ciphertext (= encrypted data), without needing to decrypt it.
- You should care about two sources of resulting complexity when it comes to ML:
  - What parts of the ML pipeline can I protect, and from whose perspective?
  - How do I run the sort of complex algebra ML relies upon on ciphertexts?

What do I need to know about cryptography?

You will need to know high level notions about plaintexts, ciphertexts, semantic security, malleability, homomorphy, public cryptography, and multi-party trusted computation.

You do not need to know how any given scheme works, though if you do have a maths background, it is very useful to check a few schemes out and convince yourselves that they are not magic.