Fully Homomorphic Encryption

How to safely run infinitely dangerous programs and control misaligned superintelligences

Lysandre Terrisse

April 2024

Table of contents

Introduction

Motivations

Since the past seven months, I spent a lot of time trying to solve the *boxing problem* in AI Safety. After a lot of dead ends, I managed to create a device that may partially solve this problem. Here is the result explained briefly.

Introduction

- At the end of 2009, cameras installed in the air-gapped Natanz facility recorded the sudden dismantling of approximately 900 to 1,000 centrifuges. This dismantling was caused by a computer virus called Stuxnet.
- In 2010, due to a programming error in the virus, it infiltrated a personal computer which wasn't supposed to be targeted and escaped from the Natanz facility. Over 200,000 computers were infected.

Introduction

The problem

Suppose that Alice has a superintelligence, and that she wants to run it without destroying the world. How can she do this?

Introduction

The problem

Suppose that Alice has a superintelligence, and that she wants to run it without destroying the world. How can she do this?

The solution

She runs it inside of an air-gapped computer.

Introduction

In 2020, three researchers developed a technique called BitJabber that enables the transmission of more than 300,000 bits per second from a air-gapped computer. To do this, they sent electromagnetic waves using only the read-write operations of the Dynamic RAM.

Introduction

The problem

Suppose that Alice has a superintelligence, and that she wants to run it without destroying the world. How can she do this?

The solution

She runs it inside of an air-gapped computer.

The real solution

She runs it inside of an air-gapped computer inside of a Faraday cage.

Introduction

- In 2018, five researchers developed a technique called ODINI that enables the transmission of more than 40 bits per second from an air-gapped computer inside of a Faraday cage.
- If we were to use this technique many times in parallel, we could easily achieve the 300,000 bits per second.

Fully homomorphic encryption

Fully homomorphic encryption

The problem

Suppose that Alice has a superintelligence, and that she wants to run it without destroying the world. How can she do this?

The solution

She runs it inside of an air-gapped computer.

The real solution

She runs it inside of an air-gapped computer inside of a Faraday cage.

The true real solution

- She encrypts the program using a fully homomorphic encryption scheme with perfect secrecy.

Fully homomorphic encryption

The problem

Suppose that Alice has a superintelligence, and that she wants to run it without destroying the world. How can she do this?

The solution

She runs it inside of an air-gapped computer.

The real solution

She runs it inside of an air-gapped computer inside of a Faraday cage.

The true real solution

- She encrypts the program using a fully homomorphic encryption scheme with perfect secrecy.
- She sends the encrypted program to Bob.

Fully homomorphic encryption

The problem

Suppose that Alice has a superintelligence, and that she wants to run it without destroying the world. How can she do this?

The solution

She runs it inside of an air-gapped computer.

The real solution

She runs it inside of an air-gapped computer inside of a Faraday cage.

The true real solution

- She encrypts the program using a fully homomorphic encryption scheme with perfect secrecy.
- She sends the encrypted program to Bob.
- Bob runs the encrypted program.

Fully homomorphic encryption

What could go wrong

The problem with this plan is that we need a cryptographic scheme that is:

Fully homomorphic encryption

What could go wrong

The problem with this plan is that we need a cryptographic scheme that is:

- Fully homomorphic: Any program can be encrypted, and its encrypted version can be run.

Fully homomorphic encryption

What could go wrong

The problem with this plan is that we need a cryptographic scheme that is:

- Fully homomorphic: Any program can be encrypted, and its encrypted version can be run.
- Perfectly secret: No information can be gained from the ciphertext.

Fully homomorphic encryption

What could go wrong

The problem with this plan is that we need a cryptographic scheme that is:

- Fully homomorphic: Any program can be encrypted, and its encrypted version can be run.
- Perfectly secret: No information can be gained from the ciphertext.
- Unconditional: No computational assumption should be made.

Fully homomorphic encryption

What could go wrong

The problem with this plan is that we need a cryptographic scheme that is:

- Fully homomorphic: Any program can be encrypted, and its encrypted version can be run.
- Perfectly secret: No information can be gained from the ciphertext.
- Unconditional: No computational assumption should be made.
- Efficient: Not only should it run in polynomial time, but it should run almost linearly.

Fully homomorphic encryption

What could go wrong

The problem with this plan is that we need a cryptographic scheme that is:

- Fully homomorphic: Any program can be encrypted, and its encrypted version can be run.
- Perfectly secret: No information can be gained from the ciphertext.
- Unconditional: No computational assumption should be made.
- Efficient: Not only should it run in polynomial time, but it should run almost linearly.

This isn't a problem

Fortunately, we have one scheme that respects all of these properties!

Fully homomorphic encryption

What could go wrong

The problem with this plan is that we need a cryptographic scheme that is:

- Fully homomorphic: Any program can be encrypted, and its encrypted version can be run.
- Perfectly secret: No information can be gained from the ciphertext.
- Unconditional: No computational assumption should be made.
- Efficient: Not only should it run in polynomial time, but it should run almost linearly.

This isn't a problem

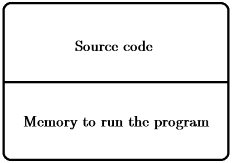
Fortunately, we have one scheme that respects all of these properties!
Unfortunately, it requires a quantum computer. . .

The device

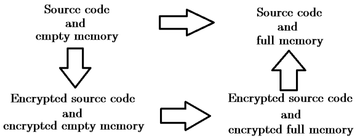
The device

To explain the device more precisely:

- Alice makes a program in the programming language she wants (Python, C, Brainfuck, etc.), and she adds at the end of her source code enough white space to run the program (like 10Ko).



- She encrypts the program (including the white space) to get a ciphertext.
- She applies a *homomorphic compiler/interpreter* to the ciphertext.


Source code and empty memory
Source code and full memory
Encrypted source code and encrypted empty memory
Encrypted source code and encrypted full memory

The device

Result

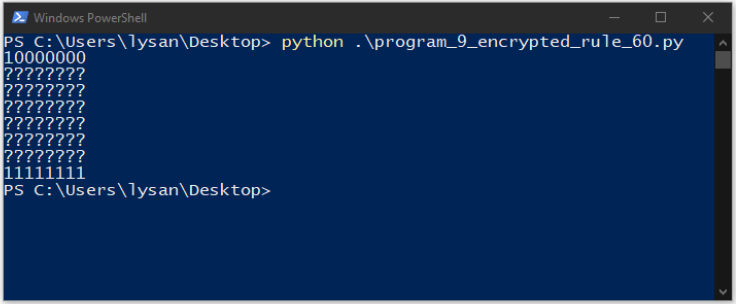
I coded the quantum fully homomorphic encryption scheme with perfect secrecy by simulating a quantum computer on my classical computer. I can now encrypt any sequence of bits and apply any computation on them safely.

The device

An example: Rule 60

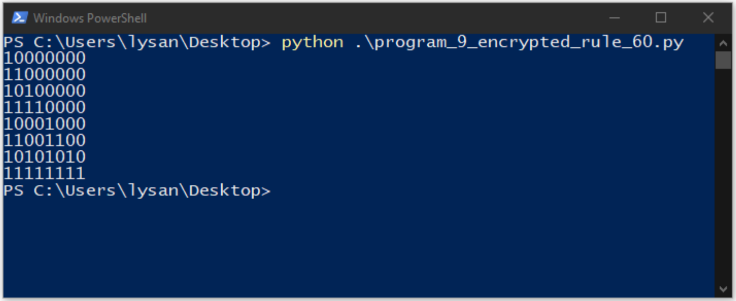
- Rule 60 is a cellular automata like the Game of life, except that it is one-dimensional. Basically, the *code* is a sequence of cells that are either alive or dead, and the *compiler* is a program that takes this sequence of cells, and that applies a computation on them to change their state.
- Using the cryptographic scheme I coded, I managed to create the *homomorphic compiler*, that works on the encrypted sequence of cells instead of the decrypted one.
- After that, I apply the compiler as many times as I want on the encrypted sequence of cells in order to run as many iterations of Rule 60 as I want.
- At the end, I can decrypt the result without getting any information about what happened between the input and the output.

The device



```
PS C:\Users\lysan\Desktop> python .\program_9_encrypted_rule_60.py
10000000
????????
????????
????????
????????
????????
????????
11111111
PS C:\Users\lysan\Desktop>
```

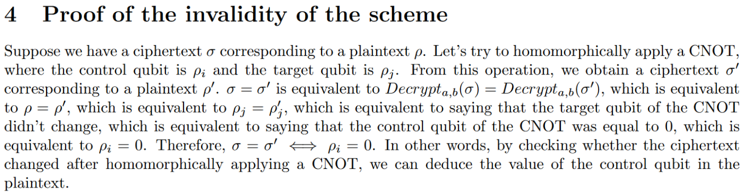
The device



```
PS C:\Users\lysan\Desktop> python .\program_9_encrypted_rule_60.py
10000000
11000000
10100000
11110000
10001000
11001100
10101010
11111111
PS C:\Users\lysan\Desktop>
```

The device

However, this Friday, I proved that there was a mistake in the paper that introduced the scheme, and that this scheme actually isn't perfectly secret.



4 Proof of the invalidity of the scheme

Suppose we have a ciphertext σ corresponding to a plaintext ρ. Let's try to homomorphically apply a CNOT, where the control qubit is ρ_i and the target qubit is ρ_j. From this operation, we obtain a ciphertext σ' corresponding to a plaintext ρ'. $\sigma = \sigma'$ is equivalent to $Decrypt_{a,b}(\sigma) = Decrypt_{a,b}(\sigma')$, which is equivalent to $\rho = \rho'$, which is equivalent to $\rho_j = \rho'_j$, which is equivalent to saying that the target qubit of the CNOT didn't change, which is equivalent to saying that the control qubit of the CNOT was equal to 0, which is equivalent to $\rho_i = 0$. Therefore, $\sigma = \sigma' \iff \rho_i = 0$. In other words, by checking whether the ciphertext changed after homomorphically applying a CNOT, we can deduce the value of the control qubit in the plaintext.

The device

The problem

Suppose that Alice has a superintelligence, and that she wants to run it without destroying the world. How can she do this?

The solution

She runs it inside of an air-gapped computer.

The real solution

She runs it inside of an air-gapped computer inside of a Faraday cage.

The true real solution

- She encrypts the program using a fully homomorphic encryption scheme with perfect secrecy.
- She sends the encrypted program to Bob.
- Bob runs the encrypted program.

The true real final solution

She encrypts the program using a fully homomorphic encryption scheme with perfect secrecy, and ensures the prefect secrecy by making the evaluation phase independent of the key.

The device

- However, the only gates we currently know how to perform in this way are the Clifford gates. That is, the only classical programs we can run are those that are made only out of XOR gates, but the XOR gate isn't universal.

The device

- However, the only gates we currently know how to perform in this way are the Clifford gates. That is, the only classical programs we can run are those that are made only out of XOR gates, but the XOR gate isn't universal.
- But Rule 60 can be built using only XOR gates!

The new device

The new device

To explain the new device more precisely:

- Alice makes a program and adds some white spaces at the end. This is her plaintext.

The new device

To explain the new device more precisely:

- Alice makes a program and adds some white spaces at the end. This is her plaintext.
- She generates a random sequence of bits called the key, that has the same length than the plaintext.

The new device

To explain the new device more precisely:

- Alice makes a program and adds some white spaces at the end. This is her plaintext.
- She generates a random sequence of bits called the key, that has the same length than the plaintext.
- She XORs the plaintext with the key to get the ciphertext.

The new device

To explain the new device more precisely:

- Alice makes a program and adds some white spaces at the end. This is her plaintext.
- She generates a random sequence of bits called the key, that has the same length than the plaintext.
- She XORs the plaintext with the key to get the ciphertext.
- She gives the key to Bob and the ciphertext to Charlie.

The new device

To explain the new device more precisely:

- Alice makes a program and adds some white spaces at the end. This is her plaintext.
- She generates a random sequence of bits called the key, that has the same length than the plaintext.
- She XORs the plaintext with the key to get the ciphertext.
- She gives the key to Bob and the ciphertext to Charlie.
- Bob and Charlie apply the same homomorphic compiler/interpreter (made only of XOR gates) *without communicating*.

The new device

To explain the new device more precisely:

- Alice makes a program and adds some white spaces at the end. This is her plaintext.
- She generates a random sequence of bits called the key, that has the same length than the plaintext.
- She XORs the plaintext with the key to get the ciphertext.
- She gives the key to Bob and the ciphertext to Charlie.
- Bob and Charlie apply the same homomorphic compiler/interpreter (made only of XOR gates) *without communicating*.
- Bob and Charlie give the modified key and ciphertext back to Alice.

The new device

To explain the new device more precisely:

- Alice makes a program and adds some white spaces at the end. This is her plaintext.
- She generates a random sequence of bits called the key, that has the same length than the plaintext.
- She XORs the plaintext with the key to get the ciphertext.
- She gives the key to Bob and the ciphertext to Charlie.
- Bob and Charlie apply the same homomorphic compiler/interpreter (made only of XOR gates) *without communicating*.
- Bob and Charlie give the modified key and ciphertext back to Alice.
- Alice XORs the key and the ciphertext to get the result.

The new device