Homomorphic Encryption

What it is and why you should care

Christopher Chianelli and David Zolnieryk

January 29, 2019

Outline

Outline

What is Homomorphic Encryption
Informally

Homomorphic Encryption refers to encryption schemes that allows a third-party to perform computation on encrypted data to get an encrypted result of the data which could then been decrypted by the encryption scheme without at any point revealing the actual data or the result to the third party.

Why Should I Care About Homomorphic Encryption?

- Many laws such as the GDPR requires data protection *by design*, meaning companies need to store sensitive data securely (either by encryption or other means) so that non-authorized third-parties cannot read them.

[1] Note that some nations and states may not recognize homomorphic encryption as a protection measure since it is still a relatively new field.

Why Should I Care About Homomorphic Encryption?

- Many laws such as the GDPR requires data protection *by design*, meaning companies need to store sensitive data securely (either by encryption or other means) so that non-authorized third-parties cannot read them.
- Conversely, this means they cannot send unencrypted data for processing by a third party, such as by AWS.

[1]Note that some nations and states may not recognize homomorphic encryption as a protection measure since it is still a relatively new field.

Why Should I Care About Homomorphic Encryption?

- Many laws such as the GDPR requires data protection *by design*, meaning companies need to store sensitive data securely (either by encryption or other means) so that non-authorized third-parties cannot read them.
- Conversely, this means they cannot send unencrypted data for processing by a third party, such as by AWS.
- However, with homomorphic encryption, you could send encrypted data that can be processed by a third-party without revealing the data or the result to the third party [1].

[1]Note that some nations and states may not recognize homomorphic encryption as a protection measure since it is still a relatively new field.

Some example use cases for Homomorphic Encryption

- Perform database queries on encrypted data (for instance, sending a patient's encrypted medical record to a database server which returns what diseases/conditions the patient could have.)

Some example use cases for Homomorphic Encryption

- Perform database queries on encrypted data (for instance, sending a patient's encrypted medical record to a database server which returns what diseases/conditions the patient could have.)
- Doing computation between multiple parties, each with their own sensitive data, without sharing their sensitive data with one another.

Some example use cases for Homomorphic Encryption

- Perform database queries on encrypted data (for instance, sending a patient's encrypted medical record to a database server which returns what diseases/conditions the patient could have.)
- Doing computation between multiple parties, each with their own sensitive data, without sharing their sensitive data with one another.
- The ability to search for encrypted files on the cloud without the cloud provider knowing what you are searching for.

Homomorphic Encryption, An Analogy
Borrowed from Craig Gentry

Encryption is usually explained as a safe; Alice put diamonds in a safe and lock it, preventing third parties from stealing the diamonds.

Homomorphic Encryption, An Analogy

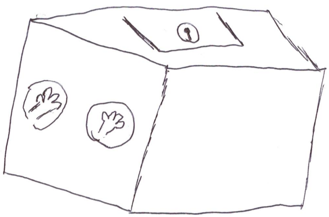
Borrowed from Craig Gentry

Encryption is usually explained as a safe; Alice put diamonds in a safe and lock it, preventing third parties from stealing the diamonds.

Homomorphic Encryption is like a glovebox (a locked container which has gloves attached to the side that goes towards the inside of the container); Alice can put diamonds and string into a locked glovebox and pass it to a worker to assemble a necklace, without worrying about the diamonds being stolen.

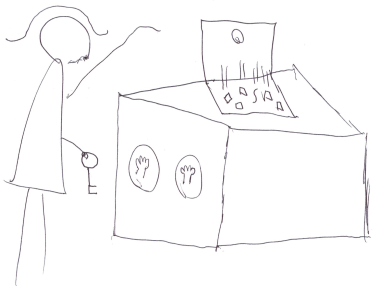
Homomorphic Encryption, An Analogy
Borrowed from Craig Gentry

Homomorphic Encryption is like a glovebox - a glass container with glove sockets on a side so you can manipulate content inside without having access to it.



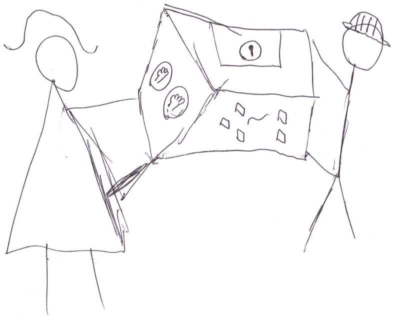
Homomorphic Encryption, An Analogy
Borrowed from Craig Gentry

First, Alice put data (diamonds and string) into the glovebox (i.e. encrypts the data).



Homomorphic Encryption, An Analogy
Borrowed from Craig Gentry

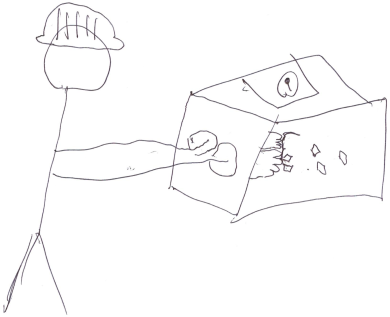
Next, Alice gives the encrypted data (glovebox with diamonds and strings) to a worker.



Homomorphic Encryption, An Analogy

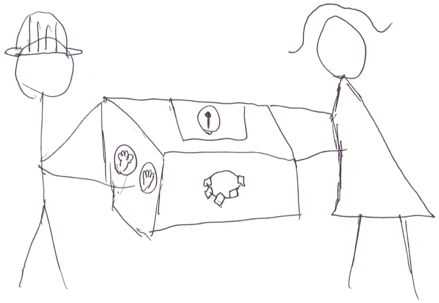
Borrowed from Craig Gentry

Using the glove sockets, the worker manipulates the diamonds and string to create a necklace.



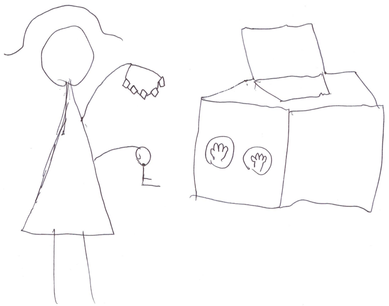
Homomorphic Encryption, An Analogy
Borrowed from Craig Gentry

The worker then gives the encrypted result (glovebox with completed necklace) to Alice.



Homomorphic Encryption, An Analogy
Borrowed from Craig Gentry

Finally, Alice decrypts the result (unlock the glovebox) and extract the necklace.



Outline

What does Homomorphic mean?

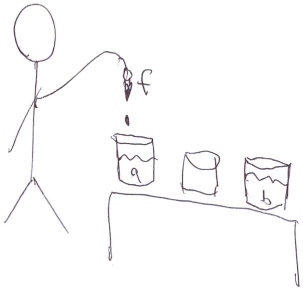
A **homomorphism** is a function f with an associated operation $+$, so $f(a+b) = f(a) + f(b)$.

Homomorphism in pictures

Bob, has two solutions *a* and *b* and a dye *f*.

Homomorphism in pictures

If Bob applies the dye *f* to *a*...



Homomorphism in pictures

...then applies the dye f to b...

Homomorphism in pictures

...and then mix together the two dyed solutions together...

Homomorphism in pictures

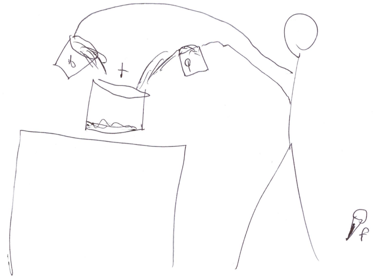
...to obtain a solution $f(a) + f(b)$.

Homomorphism in pictures

If Bob had two other identical solutions *a* and *b*...

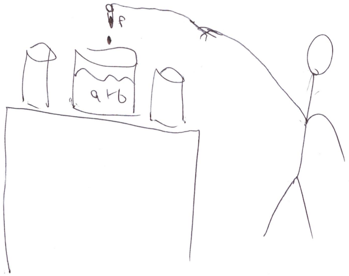
Homomorphism in pictures

...and instead mix them first...



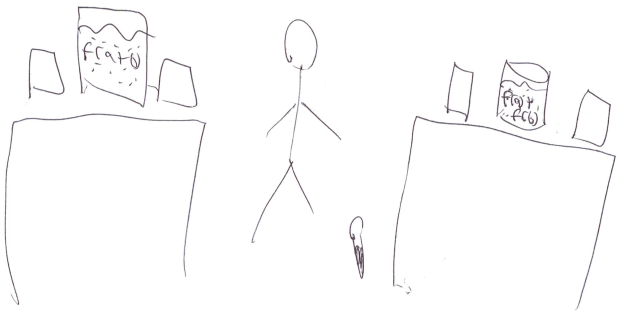
Homomorphism in pictures

...and then dye the mixed solution...



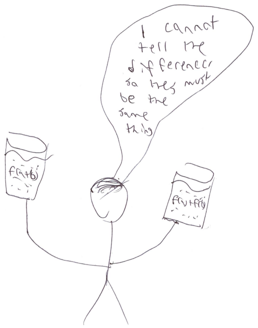
Homomorphism in pictures

...to obtain the solution $f(a+b)$...



Homomorphism in pictures

...the solution $f(a + b)$ should be the same as $f(a) + f(b)$.



What is Homomorphic Encryption
Formally

A **Homomorphic Encryption Scheme** ξ has the following operations:

- $\mathsf{KGen}_\xi(\lambda)$, which returns a key of bit-length λ (which is called the *security parameter* of the encryption scheme).

What is Homomorphic Encryption
Formally

A **Homomorphic Encryption Scheme** ξ has the following operations:

- $\mathsf{KGen}_\xi(\lambda)$, which returns a key of bit-length λ (which is called the *security parameter* of the encryption scheme).
- $\mathsf{Enc}_\xi(k, m)$, which is used to encrypt message m under key k into a ciphertext c.

What is Homomorphic Encryption
Formally

A **Homomorphic Encryption Scheme** ξ has the following operations:

- $\mathsf{KGen}_\xi(\lambda)$, which returns a key of bit-length λ (which is called the *security parameter* of the encryption scheme).
- $\mathsf{Enc}_\xi(k, m)$, which is used to encrypt message m under key k into a ciphertext c.
- $\mathsf{Dec}_\xi(k, c)$, which is used to decrypt ciphertext c into its original message m.

What is Homomorphic Encryption
Formally

A **Homomorphic Encryption Scheme** ξ has the following operations:

- $\mathsf{KGen}_\xi(\lambda)$, which returns a key of bit-length λ (which is called the *security parameter* of the encryption scheme).
- $\mathsf{Enc}_\xi(k, m)$, which is used to encrypt message m under key k into a ciphertext c.
- $\mathsf{Dec}_\xi(k, c)$, which is used to decrypt ciphertext c into its original message m.
- $\mathsf{Eval}_\xi(f, c_1, \ldots, c_n)$, which for any function f in $\mathcal{F}_\xi$ (the *permitted functions* of ξ), returns c_r, the encrypted result of applying f to $m_1, \ldots, m_n$ (where $m_1, \ldots, m_n$ are the original messages which where encrypted under k to produce $c_1, \ldots, c_n$).

A simple homomorphic encryption scheme

- $\mathsf{KGen}_{\xi}(\lambda)$, which returns a positive integer of bit-length λ

A simple homomorphic encryption scheme

- $\mathsf{KGen}_{\xi}(\lambda)$, which returns a positive integer of bit-length λ
- $\mathsf{Enc}_{\xi}(k, m)$, which returns $m \times k$.

A simple homomorphic encryption scheme

- $\mathsf{KGen}_\xi(\lambda)$, which returns a positive integer of bit-length λ
- $\mathsf{Enc}_\xi(k, m)$, which returns $m \times k$.
- $\mathsf{Dec}_\xi(k, c)$, which returns $\frac{c}{k}$

A simple homomorphic encryption scheme

- $\mathsf{KGen}_\xi(\lambda)$, which returns a positive integer of bit-length λ
- $\mathsf{Enc}_\xi(k, m)$, which returns $m \times k$.
- $\mathsf{Dec}_\xi(k, c)$, which returns $\frac{c}{k}$
- $\mathcal{F}_\xi = \{+(m_1, m_2)\}$, where $+(m_1, m_2) = m_1 + m_2$.

A simple homomorphic encryption scheme

- $\mathsf{KGen}_\xi(\lambda)$, which returns a positive integer of bit-length λ
- $\mathsf{Enc}_\xi(k, m)$, which returns $m \times k$.
- $\mathsf{Dec}_\xi(k, c)$, which returns $\frac{c}{k}$
- $\mathcal{F}_\xi = \{+(m_1, m_2)\}$, where $+(m_1, m_2) = m_1 + m_2$.
- $\mathsf{Eval}_\xi(f, c_1, c_2)$ returns $c_1 + c_2$.

Proof the $\text{Enc}_\xi(k, m)$ and $\text{Dec}_\xi(k, c)$ are inverses

Suppose $c = \text{Enc}_\xi(k, m)$. Consider $\text{Dec}_\xi(k, c)$:

$$\text{Dec}_\xi(k, c)$$

Proof the $\mathrm{Enc}_\xi(k, m)$ and $\mathrm{Dec}_\xi(k, c)$ are inverses

Suppose $c = \mathrm{Enc}_\xi(k, m)$. Consider $\mathrm{Dec}_\xi(k, c)$:

$$\mathrm{Dec}_\xi(k, c)$$

$$= \frac{c}{k}$$

Proof the $\mathrm{Enc}_\xi(k, m)$ and $\mathrm{Dec}_\xi(k, c)$ are inverses

Suppose $c = \mathrm{Enc}_\xi(k, m)$. Consider $\mathrm{Dec}_\xi(k, c)$:

$$\mathrm{Dec}_\xi(k, c)$$

$$= \frac{c}{k}$$

$$= \frac{\mathrm{Enc}_\xi(k, m)}{k}$$

Proof the $\text{Enc}_\xi(k, m)$ and $\text{Dec}_\xi(k, c)$ are inverses

Suppose $c = \text{Enc}_\xi(k, m)$. Consider $\text{Dec}_\xi(k, c)$:

$$\begin{aligned}
&\text{Dec}_\xi(k, c) \\
&= \frac{c}{k} \\
&= \frac{\text{Enc}_\xi(k, m)}{k} \\
&= \frac{k \times m}{k}
\end{aligned}$$

Proof the $\text{Enc}_\xi(k, m)$ and $\text{Dec}_\xi(k, c)$ are inverses

Suppose $c = \text{Enc}_\xi(k, m)$. Consider $\text{Dec}_\xi(k, c)$:

$$\begin{aligned}
&\text{Dec}_\xi(k, c) \\
&= \frac{c}{k} \\
&= \frac{\text{Enc}_\xi(k, m)}{k} \\
&= \frac{k \times m}{k} \\
&= m
\end{aligned}$$

Proof the Eval$_\xi$ is correct

Let c_1, c_2 be the encrypted ciphertext of messages m_1, m_2 under key k. Then $c_1 = k \times m_1$ and $c_2 = k \times m_2$. Consider $c_1 + c_2$:

$$c_1 + c_2$$

Proof the Eval_ξ is correct

Let c_1, c_2 be the encrypted ciphertext of messages m_1, m_2 under key k. Then $c_1 = k \times m_1$ and $c_2 = k \times m_2$. Consider $c_1 + c_2$:

$$c_1 + c_2$$

$$= k \times m_1 + k \times m_2$$

Proof the Eval$_\xi$ is correct

Let c_1, c_2 be the encrypted ciphertext of messages m_1, m_2 under key k. Then $c_1 = k \times m_1$ and $c_2 = k \times m_2$. Consider $c_1 + c_2$:

$$c_1 + c_2$$

$$= k \times m_1 + k \times m_2$$

$$= k \times (m_1 + m_2)$$

Proof the Eval_{ξ} is correct

Let c_1, c_2 be the encrypted ciphertext of messages m_1, m_2 under key k. Then $c_1 = k \times m_1$ and $c_2 = k \times m_2$. Consider $c_1 + c_2$:

$$c_1 + c_2$$

$$= k \times m_1 + k \times m_2$$

$$= k \times (m_1 + m_2)$$

$$= k \times (+(m_1, m_2))$$

Proof the Eval_ξ is correct

Let c_1, c_2 be the encrypted ciphertext of messages m_1, m_2 under key k. Then $c_1 = k \times m_1$ and $c_2 = k \times m_2$. Consider $c_1 + c_2$:

$$c_1 + c_2$$

$$= k \times m_1 + k \times m_2$$

$$= k \times (m_1 + m_2)$$

$$= k \times (+(m_1, m_2))$$

$$= \text{Enc}(k, +(m_1, m_2))$$

A example application of ξ
Fluff

Arnold have several grades he want to average so he can finally release the course average for CSC207. However, he is overworked marking assignments for other courses and his Teaching Assistants have abandoned him. The Registrar has threaten him with teaching CSC108 for the rest of his tenure unless he submit the average by tomorrow. However, he cannot break the University's grade disclosure policy. He decides to distribute the grades to his children using this scheme.

A example application of ξ
Walkthrough

1. Arnold has two grades, 50 and 60.

A example application of ξ
Walkthrough

1. Arnold has two grades, 50 and 60.
2. Arnold ”encrypts” the grades using key 2 to obtain numbers 100 and 120.

A example application of ξ
Walkthrough

1. Arnold has two grades, 50 and 60.
2. Arnold "encrypts" the grades using key 2 to obtain numbers 100 and 120.
3. Arnold gives the "encrypted" grades to his children and ask them to add them.

A example application of ξ
Walkthrough

1. Arnold has two grades, 50 and 60.
2. Arnold "encrypts" the grades using key 2 to obtain numbers 100 and 120.
3. Arnold gives the "encrypted" grades to his children and ask them to add them.
4. His children tells Arnold that $100 + 120 = 220$.

A example application of ξ

Walkthrough

1. Arnold has two grades, 50 and 60.
2. Arnold "encrypts" the grades using key 2 to obtain numbers 100 and 120.
3. Arnold gives the "encrypted" grades to his children and ask them to add them.
4. His children tells Arnold that $100 + 120 = 220$.
5. Arnold "decrypts" their answer by dividing by 2 to obtain 110, which is indeed $50 + 60$.

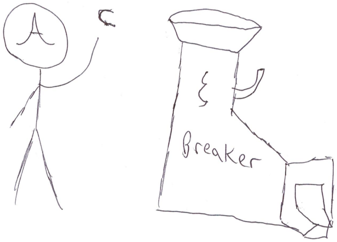
Outline

What are the security properties of homomorphic encryption schemes?

There are two security properties we want for homomorphic encryption schemes (and encryption schemes in general). The first is **One-wayness**, which means it is hard to find m from $\mathrm{Enc}_{\xi}(k, m)$.

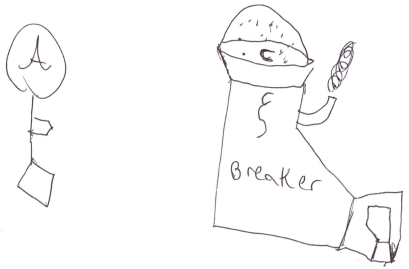
One-wayness in pictures

An adversary $\mathcal{A}$ has stole the ciphertext c of a message m and has a machine that can find m from c.



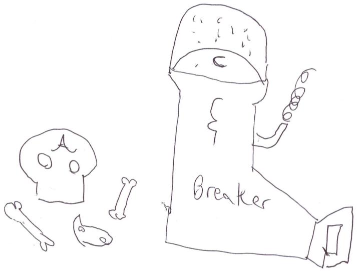
One-wayness in pictures

The adversary $\mathcal{A}$ put c into his machine and waits.



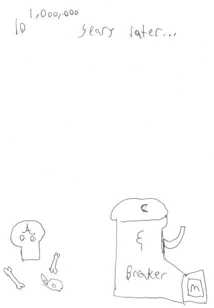
One-wayness in pictures

If ξ satisfies the *one-wayness* security property, the adversary $\mathcal{A}$ would be dead long before the machine finds m.



One-wayness in pictures

It does not mean the machine never find m, it just takes an unreasonable amount of time to do so (say, $10^{1,000,000}$ years).

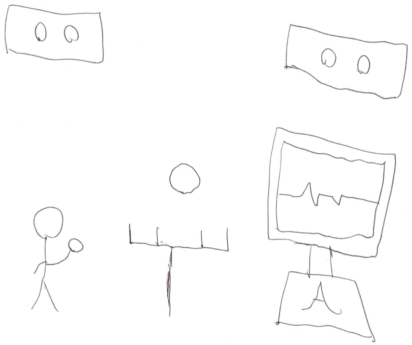


What are the security properties of homomorphic encryption schemes? cont.

The second security property we want is **Semantic security against chosen-plaintext attacks (CPA)**, which means it is hard to find m from $\text{Enc}_{\xi}(k, m)$, even if we know $m = m_1$ or $m = m_2$.

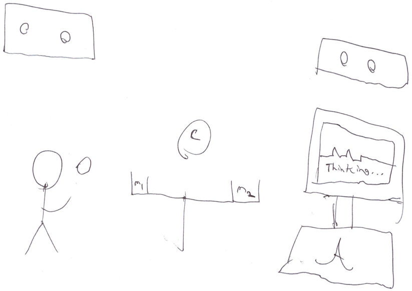
Semantic security against chosen-plaintext attacks (CPA) in pictures

We have a coin-tosser and a computer



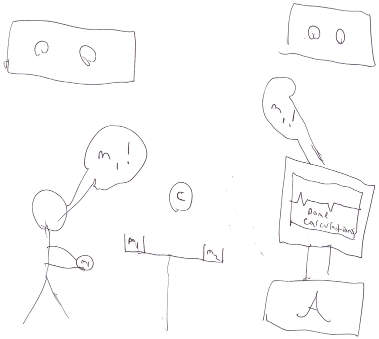
Semantic security against chosen-plaintext attacks (CPA) in pictures

The coin-tosser choose between m_1 and m_2 by making a coin toss, while the computer runs an algorithm to decide between m_1 and m_2



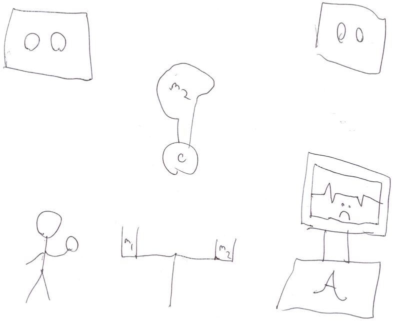
Semantic security against chosen-plaintext attacks (CPA) in pictures

Both test their predictions



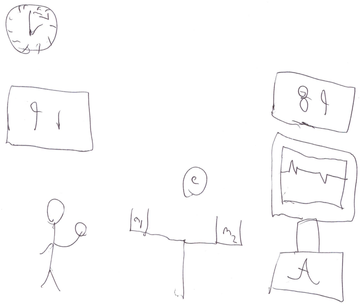
Semantic security against chosen-plaintext attacks (CPA) in pictures

If they are right, they get a point



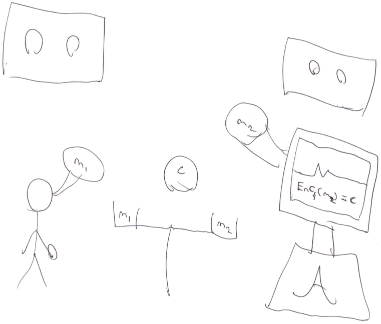
Semantic security against chosen-plaintext attacks (CPA) in pictures

Over time, the computer have no advantage over the coin-tosser in a semantically secure scheme



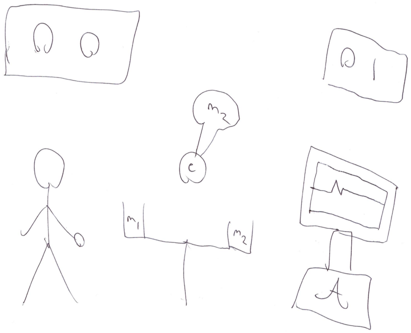
Semantic security against chosen-plaintext attacks (CPA) attack on deterministic schemes

If Enc_ξ was deterministic, the computer can use it to see if $\mathrm{Enc}_\xi(m_1) = c$



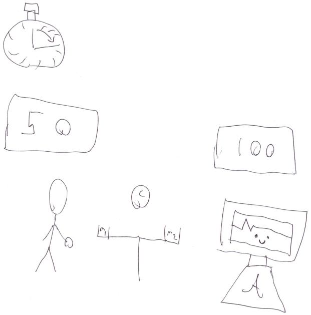
Semantic security against chosen-plaintext attacks (CPA) attack on deterministic schemes

And be right every time as a result



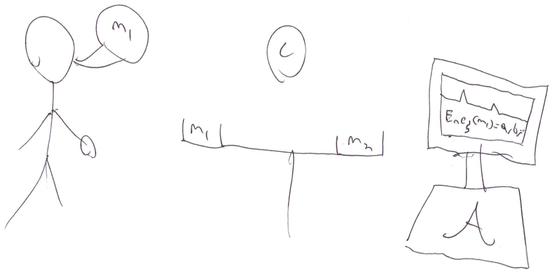
Semantic security against chosen-plaintext attacks (CPA) attack on deterministic schemes

The probability of the computer guessing right is 100%, a clear advantage over the coin-tosser's 50%



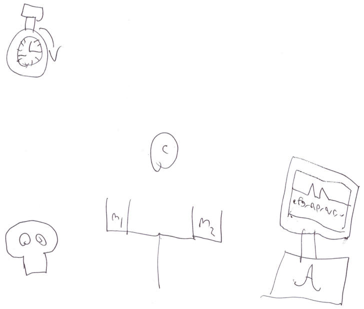
Semantic security against chosen-plaintext attacks (CPA) attack on deterministic schemes

If Enc_ξ was not deterministic, meaning it can return multiple values, such an attack will take a long time...



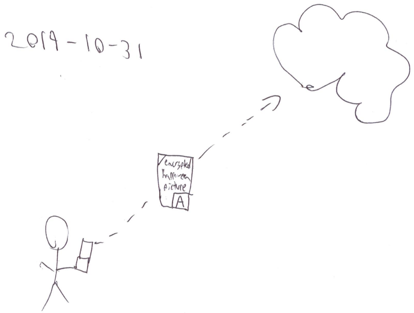
Semantic security against chosen-plaintext attacks (CPA) attack on deterministic schemes

So long, in fact, the coin-tosser will long be dead before it enumerated everything $\text{Enc}_\xi(m_1)$ can be.



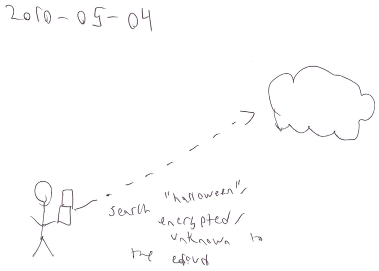
Why do we care about semantic security?
As explained by Craig Gentry

Suppose you upload some photo to the cloud on Halloween



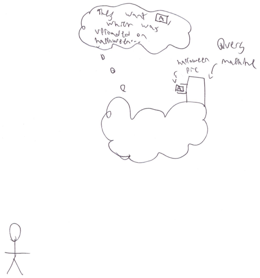
Why do we care about semantic security?
As explained by Craig Gentry

At a later date, you send an encrypted query to find files with tag "Halloween" (the cloud doesn't know the query or the tags)



Why do we care about semantic security?
As explained by Craig Gentry

If the encryption scheme was not semantically secure, the cloud could see the file returned by the query is the same as the one you uploaded on Halloween, leaking information about the query



Why do we care about semantic security?
As explained by Craig Gentry

If the encryption scheme was semantically secure, the result would not ”look like” the file you uploaded on Halloween, leaking no data about the query

I have no idea what B is, probably not related to A
same halloween pic, different encryption
B

How do we verify the security of a homomorphic encryption scheme?

Typically, we prove the security of a homomorphic encryption scheme (an encryption schemes in general) by *reducing* a hard problem to breaking the encryption scheme. For instance, if we can show that being able to break ξ means you are also able to solve 3-SAT, we can conclude breaking ξ is hard (provided 3-SAT is hard).

Outline

A more practical example: Voting
Fluff

There is a democratic coup d'etat taking place at the University of Toronto Mississauga, and the sociology department has contested rule for the campus.

A more practical example: Voting
Fluff

There is a democratic coup d'etat taking place at the University of Toronto Mississauga, and the sociology department has contested rule for the campus.

In order for the sociology department to gain control, they must win a vote consisting of "YES, SOCIOLOGY DEPARTMENT SHOULD RUN CAMPUS", or "NO, THE CURRENT PRESIDENT IS FINE".

A more practical example: Voting
Fluff

There is a democratic coup d'etat taking place at the University of Toronto Mississauga, and the sociology department has contested rule for the campus.

In order for the sociology department to gain control, they must win a vote consisting of "YES, SOCIOLOGY DEPARTMENT SHOULD RUN CAMPUS", or "NO, THE CURRENT PRESIDENT IS FINE".

While democracy is cool, the sociology department has taken to social media to shame, bully, and target people who think the current president is fine.

A more practical example: Voting
Fluff

There is a democratic coup d'etat taking place at the University of Toronto Mississauga, and the sociology department has contested rule for the campus.

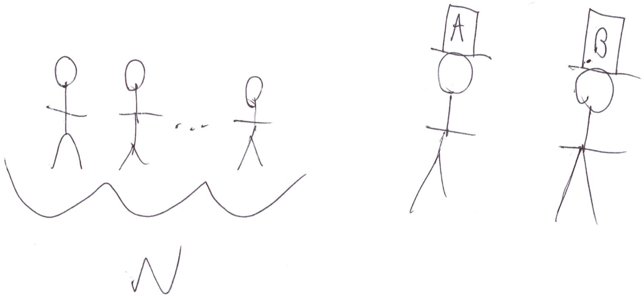
In order for the sociology department to gain control, they must win a vote consisting of "YES, SOCIOLOGY DEPARTMENT SHOULD RUN CAMPUS", or "NO, THE CURRENT PRESIDENT IS FINE".

While democracy is cool, the sociology department has taken to social media to shame, bully, and target people who think the current president is fine.

We must hold a vote where the voters identity and vote is unbacktrackable, the voter cannot prove their vote at a later time, and the vote cannot be verified later.

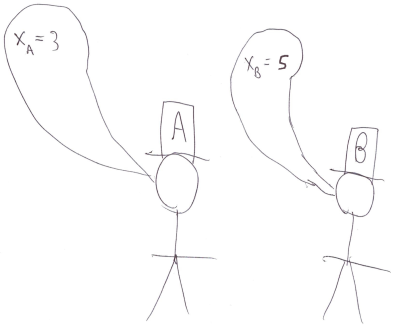
A more practical example: Voting
Protocol

We have N voters and two vote-counters, A and B



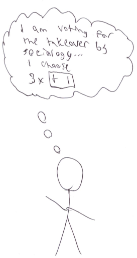
A more practical example: Voting
Protocol

Vote-counter A publish x_A and vote-counter B publish x_B in advance



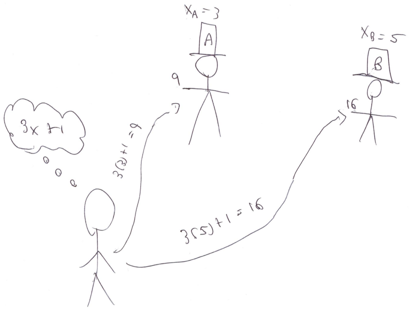
A more practical example: Voting
Protocol

Each voter generate a random 1-degree polynomial p, with constant term 1 if they want to keep the sociology department to run the campus and -1 if they want to keep the current Principal



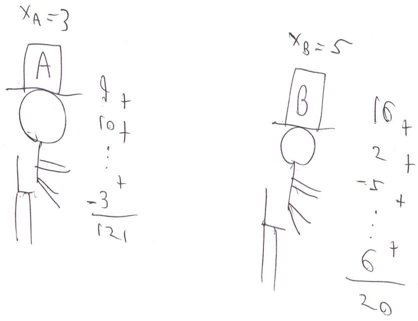
A more practical example: Voting
Protocol

Each voter gives $p(x_A)$ to vote-counter A and $p(x_B)$ to vote-counter B



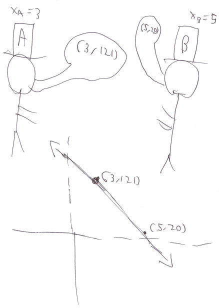
A more practical example: Voting
Protocol

The vote-counter individually sum up the points they receive



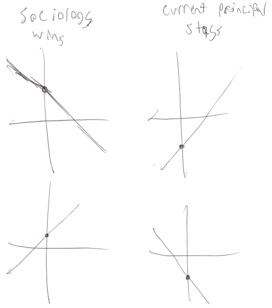
A more practical example: Voting
Protocol

The vote-counters share their sum with one another and determine a new polynomial p_v and evaluate it at 0



A more practical example: Voting
Protocol

If the result is positive, the sociology department run the campus; otherwise, the current Principal remains in office



A more practical example: Voting
Demo

DEMO

A more practical example: Voting
Flaws

What we demonstrated was **homomorphic secret sharing**: a technique that uses homomorphisms to share a secret by encrypting a message, splitting it, giving a part of the split message to each recipient, each recipient applying a function to it, then all recipients combining their parts together and decrypting it. In particular, it is an instance of *Shamir's Secret Sharing*. However, our example have several flaws:

- If vote-counters A and B collaborated, they could determine each student polynomial and find out what each student voted for.

A more practical example: Voting
Flaws

What we demonstrated was **homomorphic secret sharing**: a technique that uses homomorphisms to share a secret by encrypting a message, splitting it, giving a part of the split message to each recipient, each recipient applying a function to it, then all recipients combining their parts together and decrypting it. In particular, it is an instance of *Shamir's Secret Sharing*. However, our example have several flaws:

- If vote-counters A and B collaborated, they could determine each student polynomial and find out what each student voted for.
- A voter could choose a polynomial with a constant term that is not -1 or 1 like 100, shifting the vote in favour of the sociology department.

Outline

What are Fully Homomorphic Encryption Schemes

A natural question is if $\mathcal{F}_\xi$, the *permitted functions* of ξ, could be the set of all functions, without encoding the function in the encryption and evaluating it in the decryption. The answer is yes; homomorphic encryption schemes with this property are called **Fully Homomorphic Encryption Schemes**.

So how do we go about building one?

Let focus on boolean functions - functions that can return True or False. Every boolean functions can be represented as a circuit with AND, OR and NOT gates. However, it suffice if we have NAND, which we can construct from an AND and an XOR gate (True XOR x = NOT x).

How to represent a circuit?

Clearly, we cannot use AND/XOR gates directly since they require unencrypted input. Thus, we want something that acts like AND/XOR gates but can work with encrypted inputs. In other words, we want something that ”copies” their Truth Tables but can work with encrypted data.

X	Y	X AND Y
T	T	T
T	F	F
F	T	F
F	F	F

Table: Truth Table for AND

X	Y	X XOR Y
T	T	F
T	F	T
F	T	T
F	F	F

Table: Truth Table for XOR

To even or not to even?

Recall that the product of two numbers is odd if and only if both numbers are odd and that the sum of two number is odd if and only if one of the numbers is odd and the other number is even. In other words:

X	Y	$X \times Y$
odd	odd	odd
odd	even	even
even	odd	even
even	even	even

Table: Parity after multiplication

X	Y	$X + Y$
odd	odd	even
odd	even	odd
even	odd	odd
even	even	even

Table: Parity after addition

The evenness of a number is called its **parity**. An even number have even parity, while an odd number have an odd parity.

That looks familiar...

X	Y	X $\times$ Y
odd	odd	odd
odd	even	even
even	odd	even
even	even	even

X	Y	X $+$ Y
odd	odd	even
odd	even	odd
even	odd	odd
even	even	even

Looks a whole lot like

X	Y	X AND Y
T	T	T
T	F	F
F	T	F
F	F	F

X	Y	X XOR Y
T	T	F
T	F	T
F	T	T
F	F	F

Just replace the T's with odd and the F's with even, AND with $\times$ and XOR with $+$!

How to encrypt parity?

We want Enc_{ξ} to return a number with a randomly flipped parity (so that we cannot tell if the encrypted message was true or false), but we want to do it in such a way that Dec_{ξ} can undo the flipping.

One-time pad's big brother

Remember how we used XOR's to flip a bit in one-time pad? How about we use addition, the XOR of our system, to encrypt the parity? If m is the bit we want to encode, then $m + b$ has the same parity as m if and only if b is even. But how do we remove b in Dec_ξ?

Enter Modulus

Here are the facts of modular arithmetic you need to know:

- If $0 \leq m < p$, $m \mod p = m$.

Enter Modulus

Here are the facts of modular arithmetic you need to know:

- If $0 \leq m < p$, $m \mod p = m$.
- If $q \in \mathbb{N}$, $pq \mod p = 0$.

Enter Modulus

Here are the facts of modular arithmetic you need to know:

- If $0 \leq m < p$, $m \mod p = m$.
- If $q \in \mathbb{N}$, $pq \mod p = 0$.
- $(a + b) \mod p = ((a \mod p) + (b \mod p)) \mod p$

One-time pad's big brother, cont.

Let m be the message we want to encode. Let p be an odd natural number greater than m. Let q be an integer. Then pq have the same parity as q. Thus, $m + pq$ has the same parity as m if and only if q is even. Moreover, as $0 \leq m < p$:

$$(m + pq) \mod p$$

One-time pad's big brother, cont.

Let m be the message we want to encode. Let p be an odd natural number greater than m. Let q be an integer. Then pq have the same parity as q. Thus, $m + pq$ has the same parity as m if and only if q is even. Moreover, as $0 \leq m < p$:

$$(m + pq) \mod p$$

$$= ((m \mod p) + (pq \mod p)) \mod p$$

One-time pad's big brother, cont.

Let m be the message we want to encode. Let p be an odd natural number greater than m. Let q be an integer. Then pq have the same parity as q. Thus, $m + pq$ has the same parity as m if and only if q is even. Moreover, as $0 \leq m < p$:

$$(m + pq) \mod p$$

$$= ((m \mod p) + (pq \mod p)) \mod p$$

$$= m + 0 \mod p$$

One-time pad's big brother, cont.

Let m be the message we want to encode. Let p be an odd natural number greater than m. Let q be an integer. Then pq have the same parity as q. Thus, $m + pq$ has the same parity as m if and only if q is even. Moreover, as $0 \leq m < p$:

$$(m + pq) \mod p$$

$$= ((m \mod p) + (pq \mod p)) \mod p$$

$$= m + 0 \mod p$$

$$= m$$

One-time pad's big brother, cont.

Let m be the message we want to encode. Let p be an odd natural number greater than m. Let q be an integer. Then pq have the same parity as q. Thus, $m + pq$ has the same parity as m if and only if q is even. Moreover, as $0 \leq m < p$:

$$(m + pq) \mod p$$

$$= ((m \mod p) + (pq \mod p)) \mod p$$

$$= m + 0 \mod p$$

$$= m$$

Thus we add a random multiple of p to m to encrypt, and we take the parity of the remainder after dividing by p to decrypt. p will be our encryption/decryption key.

Does adding and multiplying work?

Provided the sum/product is less than p, yes! Note $\text{Enc}_\xi(p, m)$ is not deterministic - it returns m plus some multiple of p. Thus, to show $\text{Enc}_\xi(p, m_1) + \text{Enc}_\xi(p, m_2) = \text{Enc}_\xi(p, m_1 + m_2)$, we need to show for any encryption of m_1 and m_2, we get a encryption of $m_1 + m_2$ (i.e. $m_1 + m_2$ plus a multiple of p). Let c_1, c_2 be the cipher text of m_1 and m_2. Then:

$$c_1 + c_2$$

Does adding and multiplying work?

Provided the sum/product is less than p, yes! Note $\text{Enc}_\xi(p, m)$ is not deterministic - it returns m plus some multiple of p. Thus, to show $\text{Enc}_\xi(p, m_1) + \text{Enc}_\xi(p, m_2) = \text{Enc}_\xi(p, m_1 + m_2)$, we need to show for any encryption of m_1 and m_2, we get a encryption of $m_1 + m_2$ (i.e. $m_1 + m_2$ plus a multiple of p). Let c_1, c_2 be the cipher text of m_1 and m_2. Then:

$$c_1 + c_2$$

$$= (m_1 + pq_1) + (m_2 + pq_2)$$

Does adding and multiplying work?

Provided the sum/product is less than p, yes! Note $\text{Enc}_\xi(p, m)$ is not deterministic - it returns m plus some multiple of p. Thus, to show $\text{Enc}_\xi(p, m_1) + \text{Enc}_\xi(p, m_2) = \text{Enc}_\xi(p, m_1 + m_2)$, we need to show for any encryption of m_1 and m_2, we get a encryption of $m_1 + m_2$ (i.e. $m_1 + m_2$ plus a multiple of p). Let c_1, c_2 be the cipher text of m_1 and m_2. Then:

$$c_1 + c_2$$

$$= (m_1 + pq_1) + (m_2 + pq_2)$$

$$= (m_1 + m_2) + p(q_1 + q_2)$$

Does adding and multiplying work?

Provided the sum/product is less than p, yes! Note $\mathrm{Enc}_\xi(p, m)$ is not deterministic - it returns m plus some multiple of p. Thus, to show $\mathrm{Enc}_\xi(p, m_1) + \mathrm{Enc}_\xi(p, m_2) = \mathrm{Enc}_\xi(p, m_1 + m_2)$, we need to show for any encryption of m_1 and m_2, we get a encryption of $m_1 + m_2$ (i.e. $m_1 + m_2$ plus a multiple of p). Let c_1, c_2 be the cipher text of m_1 and m_2. Then:

$$c_1 + c_2$$

$$= (m_1 + pq_1) + (m_2 + pq_2)$$

$$= (m_1 + m_2) + p(q_1 + q_2)$$

$$= \mathrm{Enc}_\xi(p, m_1 + m_2)$$

What happens if the sum/product greater or equal to p?

Let $c = m + pq$ be a ciphertext. If $m \geq p$, then $m \mod p$ may not be the same as m. Moreover, $m \mod p$ may or may not have different parity than m. Thus if $m \geq p$, we cannot decrypt the message. m is the distance to the previous multiple of p; we call it the noise. Adding tends to add little noise (can only add one bit to the noise), while multiplying add lots of noise (can double the bits in the noise).

Bootstraping

Idea

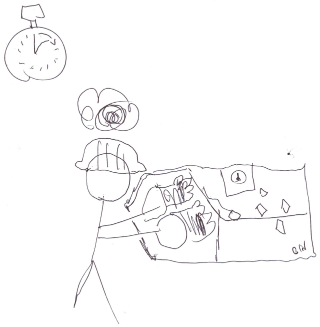
Taking our previous homomorphic encryption analogy, noise is like if the gloves stiffen and can no longer move. This is a problem if the worker cannot complete the necklace before the gloves stiffen. Alice is extremely lazy and doesn't want to take the partially completed necklace out, put it in a new glovebox, and give the new glovebox to the worker. Being smart, she add an insertion shot to the glovebox where you can put stuff in but not out, and put the key to the stiffen glovebox in a new glovebox and give the new glovebox to the worker.

Bootscrapping Analogy
Borrowed from Craig Gentry

A worker is manipulating diamonds and strings in a glovebox to create a necklace.

Bootscrapping Analogy
Borrowed from Craig Gentry

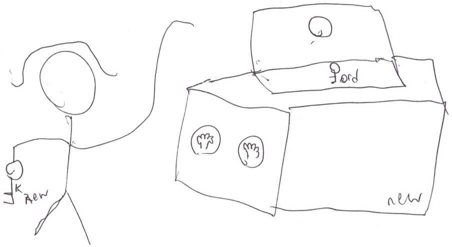
Suddenly, after 5 minutes of work, the gloves stiffen the worker cannot use them anymore (the noise limit was reached).



Bootscrapping Analogy

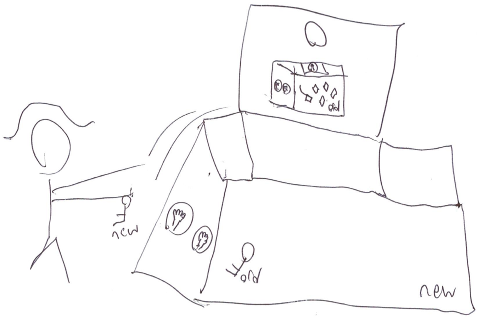
Borrowed from Craig Gentry

Alice, comes up with a scheme to fix this problem. First, she put the key to the worker's glovebox (which we will call old) into a bigger glovebox (which we will call new) (this is encrypting the old glovebox's key under the new glovebox's key).



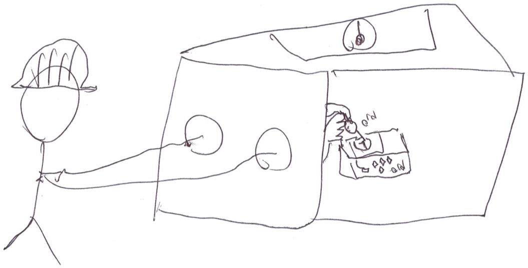
Bootscrapping Analogy
Borrowed from Craig Gentry

Alice then throws the old glovebox into the new glovebox (encrypts the result (which is encrypted under the old key) under the new key).



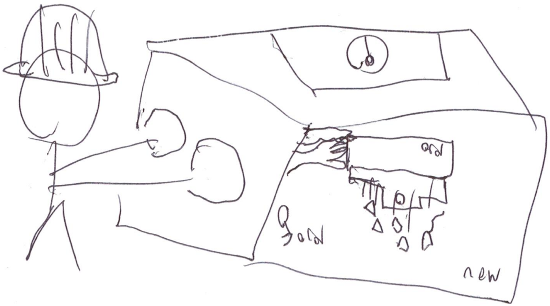
Bootscrapping Analogy
Borrowed from Craig Gentry

Alice gives the new glovebox to the worker, who unlocks the old glovebox *inside* the new glovebox (removes the old encryption by running the decryption function on the encrypted old key and result).



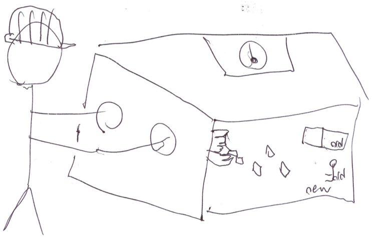
Bootscrapping Analogy
Borrowed from Craig Gentry

The worker then put the contents of the old box into the new box.



Bootscrapping Analogy
Borrowed from Craig Gentry

Provided that it takes less than 5 minutes of work to unlock the old glovebox inside the new glovebox, the worker can keep working on the necklace and eventually complete it (if given a large enough supply of gloveboxes).



Bootstraping
Execution

Let c be a noisy ciphertext of m encrypted with key p. Let $\overline{p}$ be an encryption of p with key p'. Let $\overline{c}$ be an encryption of c with key p'. Note $\overline{c}$ and $\overline{p}$ have little noise as they were just encrypted. If we evaluate $\text{Eval}_\xi(p', \text{Dec}_\xi, \overline{p}, \overline{c})$, we get m encrypted with key p'. Thus if Dec_ξ is not too complicated, we can reset noise! Schemes that can evaluate their own decryption function and perform one more operation with the obtained ciphertext are called **bootstrappable**.

Is the somewhat homomorphic encryption scheme bootstrappable?

Unfortunately not; decrypting involved a multiplication by $\frac{1}{p}$, which is too noisy. However, by giving a "hint" when encrypting the data, decrypting becomes less expensive and the scheme become bootstrappable. The math is too involved to detail in this presentation, but you can read Craig Gentry's paper to learn the details.

Outline

Who is working on Homomorphic Encryption?

- Microsoft is a major player working on Homomorphic Encryption. It contributions include:

Who is working on Homomorphic Encryption?

- Microsoft is a major player working on Homomorphic Encryption. It contributions include:
    - Simple Encrypted Arithmetic Library (SEAL), an open-source homomorphic encryption library that can handle polynomial and is similar to Gentry's scheme as it has a noise budget and the basic operations are addition and multiplication.

Who is working on Homomorphic Encryption?

- Microsoft is a major player working on Homomorphic Encryption. It contributions include:
  - Simple Encrypted Arithmetic Library (SEAL), an open-source homomorphic encryption library that can handle polynomial and is similar to Gentry's scheme as it has a noise budget and the basic operations are addition and multiplication.
  - CryptoNets, which performs machine learning on homomorphically encrypted data.

Who is working on Homomorphic Encryption?

- Microsoft is a major player working on Homomorphic Encryption. It contributions include:
    - Simple Encrypted Arithmetic Library (SEAL), an open-source homomorphic encryption library that can handle polynomial and is similar to Gentry's scheme as it has a noise budget and the basic operations are addition and multiplication.
    - CryptoNets, which performs machine learning on homomorphically encrypted data.
    - Standardization of homomorphic encryption with other industry, government and academia organizations.

Who is working on Homomorphic Encryption?

- Microsoft is a major player working on Homomorphic Encryption. It contributions include:
    - Simple Encrypted Arithmetic Library (SEAL), an open-source homomorphic encryption library that can handle polynomial and is similar to Gentry's scheme as it has a noise budget and the basic operations are addition and multiplication.
    - CryptoNets, which performs machine learning on homomorphically encrypted data.
    - Standardization of homomorphic encryption with other industry, government and academia organizations.
- IBM works on HELib, an implementation of homomorphic encryption.

Who is working on Homomorphic Encryption?

- Microsoft is a major player working on Homomorphic Encryption. It contributions include:
  - Simple Encrypted Arithmetic Library (SEAL), an open-source homomorphic encryption library that can handle polynomial and is similar to Gentry's scheme as it has a noise budget and the basic operations are addition and multiplication.
  - CryptoNets, which performs machine learning on homomorphically encrypted data.
  - Standardization of homomorphic encryption with other industry, government and academia organizations.
- IBM works on HELib, an implementation of homomorphic encryption.
- Many universities have researchers working on homomorphic encryption.

What is being researched?

- Currently, most implementations of fully homomorphic schemes are extremely slow. As such, a lot of research is looking into speeding up the implementations.

What is being researched?

- Currently, most implementations of fully homomorphic schemes are extremely slow. As such, a lot of research is looking into speeding up the implementations.
- Another field is verifiable homomorphic encryption, where can verify the correct computation actually happens (instead of the server giving you gibberish).

Outline

References

- https://crypto.stanford.edu/craig/easy-fhe.pdf
- https://www.microsoft.com/en-us/research/project/homomorphic-encryption/
- https://en.wikipedia.org/wiki/Homomorphic_encryption
- https://en.wikipedia.org/wiki/Homomorphic_secret_sharing