Homomorphic Encryption

JOYCE YEO SHUHUI

Why Homomorphic Encryption?

Consumers are increasingly concerned about **data privacy**

How can businesses leverage on big data while being cautious of data privacy?

Why Homomorphic Encryption?

Homomorphic Encryption is a *Privacy-Enhancing Technology* that can achieve Data Privacy

History of Homomorphic Encryption

- Rather new technology; 30+ years since developed
- **Limited commercial applications** due to lack of specialist knowledge and standardisations
- In recent years, tech leaders (IBM, Google, Microsoft) are pushing for their widespread use

Homomorphic Encryption (HE)

- Based on Homomorphisms, a special type of mathematical function
- Fundamentally different from current encryption systems like RSA or AES
- **Quantum-resistant**: RSA is vulnerable is a post-quantum world, but Homomorphic Encryption is not

Homomorphisms

$$\mathbf{Enc(a) + Enc(b) = Enc(a + b)}$$

"property-preserving" function

Homomorphisms

$$\mathrm{Enc}(a) + \mathrm{Enc}(b) = \mathrm{Enc}(a + b)$$

Adding encrypted messages

Adding unencrypted messages

"property-preserving" function

Homomorphic Encryption (HE)

- Property-preserving encryption
- **Able to run algorithms on encrypted data**
- Results will be as if the algorithms were run on the raw, unencrypted data

A Common Scenario

- Client Company A: owns the raw data
- Server Company B: owns the software/ analytics ability

\- Both parties are unwilling to share with each other

Solution: Homomorphic Encryption

1. Client encrypts raw data
2. Client sends encrypted data to server
3. Server **computes on encrypted data**
4. Server returns encrypted results to client
5. Client decrypts results

Data is encrypted for the server

Result is as if server performed computations on raw data

Pros & Cons of HE

Benefits	Disadvantages
Allows computations on encrypted data	**Significantly slower** than non-homomorphic encryption
Resolves the data privacy conflict between data owners and analytics companies	Lack of readily-available toolkits and standardisations (still in development!)
	Inherently not CCA-secure, which limits its applications

Key Applications of HE

Healthcare analytics
- Companies and governments holding sensitive data can outsource the analytics

Encrypted search
- Search can be done by only indexing encrypted data

End-to-end verifiability in **voting systems**
- Elections can be audited without revealing votes

Security Problems of Homomorphic Encryption

HE is not CCA secure

Adding any two ciphertexts A and B together will result in a **valid ciphertext** C. This property-preserving function of HE is known as ciphertext **malleability**.

Given the ciphertexts of A, B and C, attackers can discover that C = A + B. **Attackers know the relationship between the ciphertexts**.

HE is not CCA secure

Actual value of C is protected since actual values of A and B are not known.

However, what if the attacker can know A and B?

Then, the security of the HE scheme is compromised.

HE is not CCA secure

Data owners need to ensure that the raw data cannot be easily accessible.

However, applications can be very complex and include multi-party communication. **Rich data flows could lead to inadvertent data leaks**.

HE is still safe for pure outsourcing (one client one server) scenarios.

HE is not CCA secure

Secondly, HE does not provide integrity checks.

The attacker can simply double every data point using homomorphic operations, changing the entire database.

Currently, the only solution is to use canaries to detect these modifications.

Developing applications using Homomorphic Encryption

Types of HE

- HE is a broad class of encryption systems
- **Partially HE:** Some systems are only homomorphic for **addition only or multiplication only**, but not both
- **Somewhat HE:** Some systems only support a **limited number of operations**
- **Fully HE:** The most theoretically correct but also not practical to use

Common HE schemes

DGHV, BGV, BFV, CKKS (named after the researchers who developed them)

Decide on a scheme based on the data type and operations needed

- E.g. CKKS is an approximated-scheme that supports decimal operations

Brakerski/Fan-Vercauteren (BFV)

One of the more commonly-implemented HE schemes.

Based on the **Ring Learning with Error (RLWE)** problem that is computationally hard to solve.

Supports both public-key and symmetric encryption.

BFV

Encryption adds **noise** to the messages

- Without the key, it is difficult to decrypt the noisy messages

Noise grows during computation process

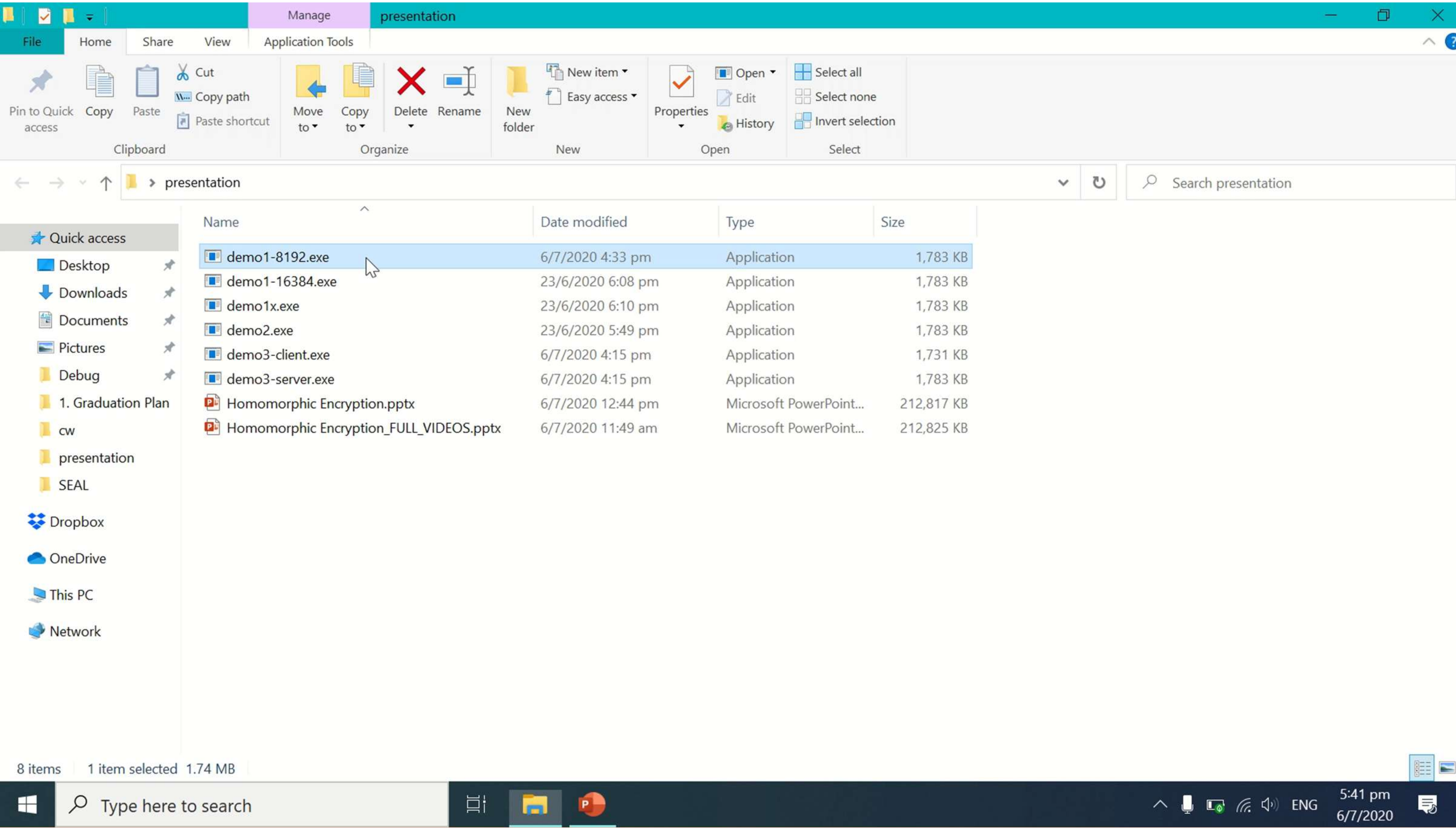
- Adding two ciphertexts: negligible growth
- Multiplying two ciphertexts: noise almost doubles

BFV

Once noise exceeds threshold, it is not possible to decrypt the message correctly

Able to **“reset” the noise** by **bootstrapping**

- Decrypt and re-encrypt the messages
- Very expensive operation (HE is slow)


Manage
presentation
File
Home
Share
View
Application Tools
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
presentation
Search presentation
Name
Date modified
Type
Size
demo1-8192.exe 6/7/2020 4:33 pm Application 1,783 KB
demo1-16384.exe 23/6/2020 6:08 pm Application 1,783 KB
demo1x.exe 23/6/2020 6:10 pm Application 1,783 KB
demo2.exe 23/6/2020 5:49 pm Application 1,783 KB
demo3-client.exe 6/7/2020 4:15 pm Application 1,731 KB
demo3-server.exe 6/7/2020 4:15 pm Application 1,783 KB
Homomorphic Encryption.pptx 6/7/2020 12:44 pm Microsoft PowerPoint... 212,817 KB
Homomorphic Encryption_FULL_VIDEOS.pptx 6/7/2020 11:49 am Microsoft PowerPoint... 212,825 KB
Quick access
Desktop
Downloads
Documents
Pictures
Debug
1. Graduation Plan
cw
presentation
SEAL
Dropbox
OneDrive
This PC
Network
8 items
1 item selected 1.74 MB
Type here to search
ENG
5:41 pm
6/7/2020

C:\Users\Joyce\Desktop\presentation\demo1-8192.exe

```
Microsoft SEAL version: 3.5.3

+--------------------------------------------------------+
|         Demo 1: Noise growth in BFV encryption         |
+--------------------------------------------------------+
/
| Encryption parameters :
|   scheme: BFV
|   poly_modulus_degree: 8192
|   coeff_modulus size: 218 (43 + 43 + 44 + 44 + 44) bits
|   plain_modulus: 2048
\
Parameter validation (success): valid

~~~~~~ Calculate (x^4+1) for x = 6 ~~~~~~
STEP 1 -->
Express x = 6 as a hexadecimal plaintext polynomial 0x6.
STEP 2 -->
Encrypt the plaintext.
    + number of polynomials (size) of freshly encrypted x: 2
    + noise budget in freshly encrypted x: 155 bits
STEP 3 -->
First, compute x_square (x^2).
    + number of polynomials (size) of x_square: 3
    + noise budget in x_square: 132 bits
STEP 4 -->
Then, compute x_fourth (x^4).
    + number of polynomials (size) of x_fourth: 5
    + noise budget in x_fourth: 101 bits
    + multiplication roughly doubles the size of the ciphertext
STEP 5 -->
Next, add 1 to the result (x^4 + 1).
    + number of polynomials (size) of x_fourth_plus_one: 5
    + noise budget in x_fourth_plus_one: 101 bits
    + addition is a negligible operation and does not have a large effect on the noise
STEP 6 -->
    + decryption of encrypted_result: 0x511 ...... expected was 0x511

~~~~~~~~~~~~ End of Program ~~~~~~~~~~~~
```

demo1: Summary of Steps

1. Encrypt plaintext x to get Enc(x)
2. $Enc(x^2) = Enc(x) * Enc(x)$
3. $Enc(x^4) = Enc(x^2) * Enc(x^2)$
4. $Enc(x^4 + 1) = Enc(x^4) + Enc(1)$
5. Get a ciphertext representing $Enc(x^4 + 1)$
6. Decrypt the ciphertext to get $x^4 + 1$

demo1: Summary of Steps

1. Encrypt plaintext x to get Enc(x)
2. $Enc(x^2) = Enc(x) * Enc(x)$
3. $Enc(x^4) = Enc(x^2) * Enc(x^2)$
4. $Enc(x^4 + 1) = Enc(x^4) + Enc(1)$
5. Get a ciphertext representing $Enc(x^4 + 1)$
6. Decrypt the ciphertext to get $x^4 + 1$

Homomorphic operations

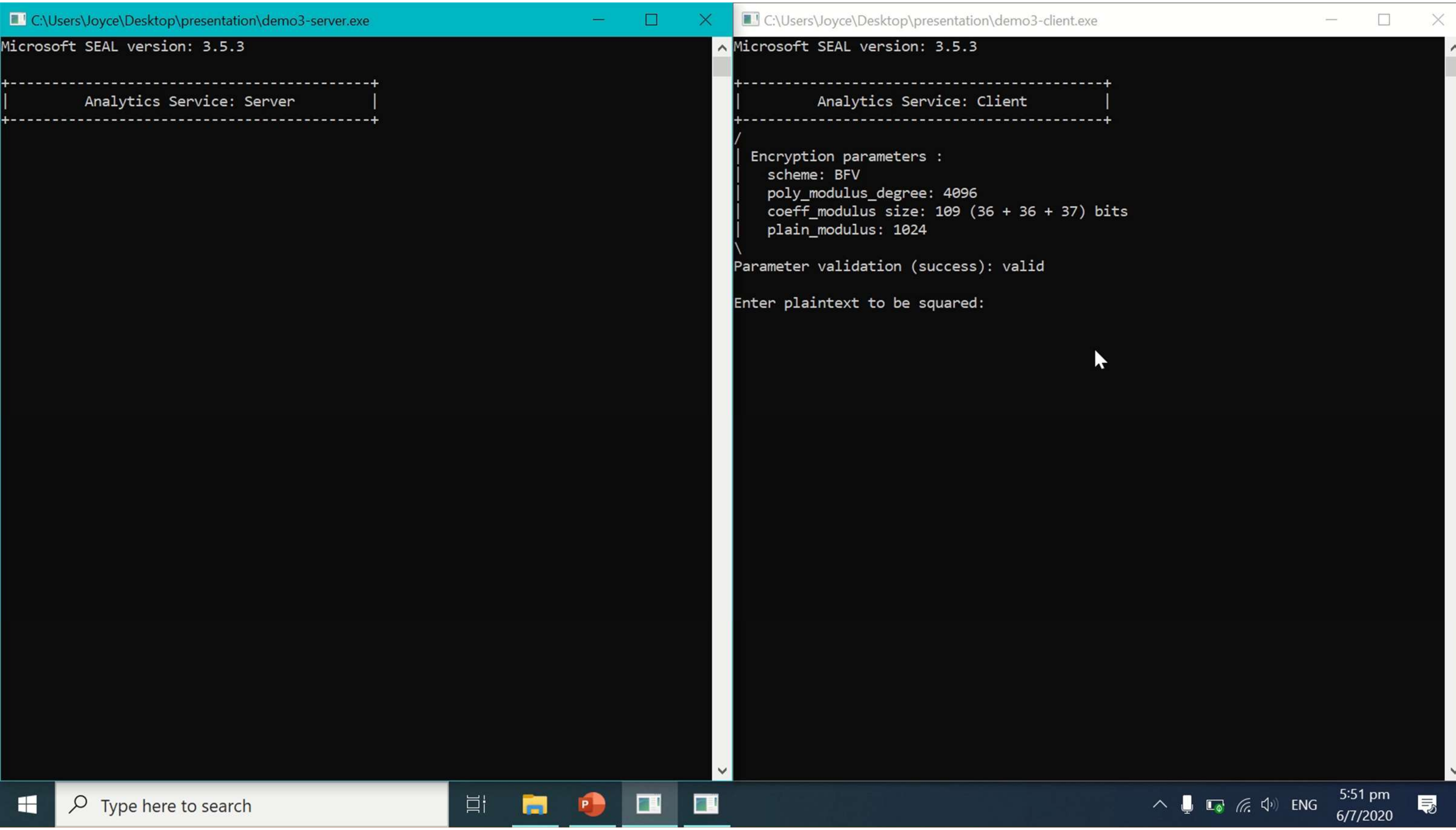
demo1: Size and noise budget changes

Ciphertext	Size (number of polynomials)	Noise budget (Remaining noise space)
x	2	54
x^2	3	31
x^4	5	2
$x^4 + 1$	5	2

demo: client-server model

Client encrypts raw data, and holds encryption key

Server does not have access to raw data and encryption keys


C:\Users\Joyce\Desktop\presentation\demo3-server.exe
Microsoft SEAL version: 3.5.3
Analytics Service: Server
C:\Users\Joyce\Desktop\presentation\demo3-client.exe
Microsoft SEAL version: 3.5.3
Analytics Service: Client
Encryption parameters :
scheme: BFV
poly_modulus_degree: 4096
coeff_modulus size: 109 (36 + 36 + 37) bits
plain_modulus: 1024
Parameter validation (success): valid
Enter plaintext to be squared:
Type here to search
ENG
5:51 pm
6/7/2020

C:\Users\Joyce\Desktop\presentation\demo3-server.exe

```
Microsoft SEAL version: 3.5.3

+---------------------------------------------+
|         Analytics Service: Server           |
+---------------------------------------------+
Size of ciphertext received: 91303 bytes
Parameter validation (success): valid

Compute x_square
Sending results back...
Size of ciphertext sent: 136594 bytes
Connection closing...
~~~~~~~~~~~~ End of Program ~~~~~~~~~~~~
```

C:\Users\Joyce\Desktop\presentation\demo3-client.exe

```
Microsoft SEAL version: 3.5.3

+---------------------------------------------+
|         Analytics Service: Client           |
+---------------------------------------------+
/
| Encryption parameters :
|   scheme: BFV
|   poly_modulus_degree: 4096
|   coeff_modulus size: 109 (36 + 36 + 37) bits
|   plain_modulus: 1024
\
Parameter validation (success): valid

Enter plaintext to be squared:
4
Size of ciphertext sent: 91303 bytes
Size of ciphertext received: 136594 bytes
Connection closed
Expected result (decimal): 16
Received result (hexadecimal): 0x10
~~~~~~~~~~~~ End of Program ~~~~~~~~~~~~
```

demo: Space consumption of data

- Large increase in space consumption

- Unencrypted integers = 4 bytes
- After encryption, 100 000 bytes
- Not very scalable for large data sets

Problems with SEAL::BFV

Limited practical use

- **Lack of functionality - Only primitive operations are available** (addition, multiplication)
- Encryption and decryption takes up data owner’s resources

Trade-off between spending resources on encryption vs spending resources on computation and analytics

Problems with SEAL::BFV

A rather **low-level library working with polynomial operations**

- Developers need to have an understanding of how to represent the data in polynomials
- To select the BFV parameters, there is a need to estimate the computations results beforehand
- Difficult to adapt existing machine learning models to HE libraries

Moving Forward with HE

Still a research-level technique.

Have to solve fundamental issues like **making basic operations faster**.

Ultimately, the goal is to create **data-agnostic software** with generic analytics capabilities.

More about the Development Environment

Done using native C++ on Windows

SEAL library with BFV encryption

- Homomorphic for addition and multiplication

Server is on only localhost, and communicates with Client via WinSock

For Developers

Libraries:

- PALISADE (a collaboration by various universities)
- **SEAL** (Microsoft)
- HELib (IBM)
- Private Join and Compute (Google)

Thank You!