Fully Homomorphic Encryption and its Use Cases

In cooperation with Pascal Meyer and Alexander Widak (Atruvia AG)

Felix P. Paul

Johannes Gutenberg-Universität Mainz

March 26, 2024

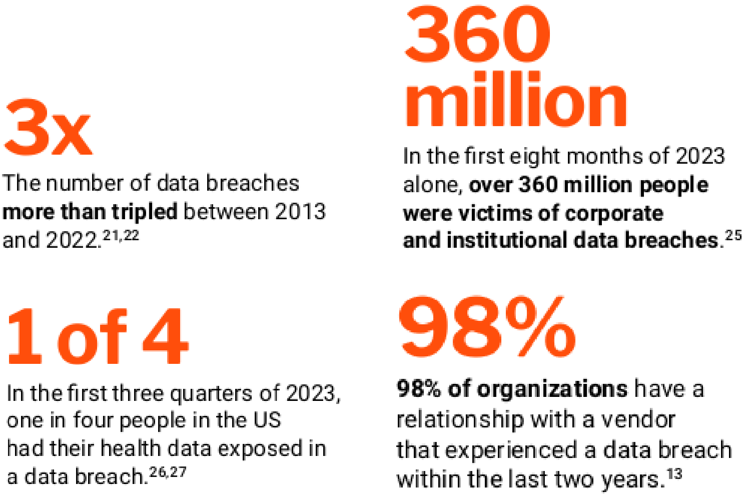

Figure 1: Rise of data breaches in the cloud [3]

FHE allows secure cloud computations

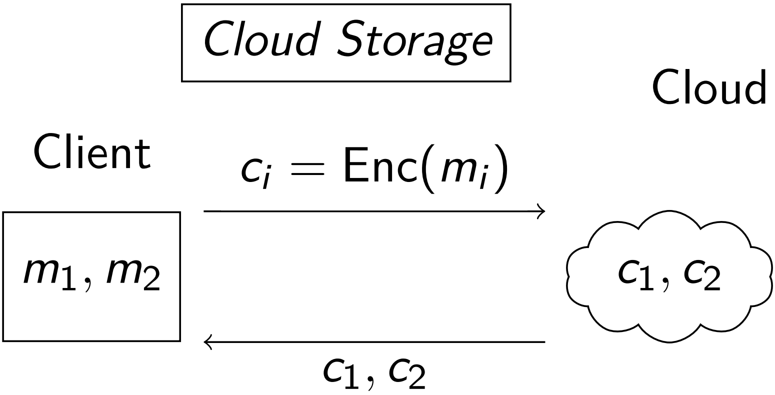

Figure 2: Usage of cloud storage - always encrypted

FHE allows secure cloud computations

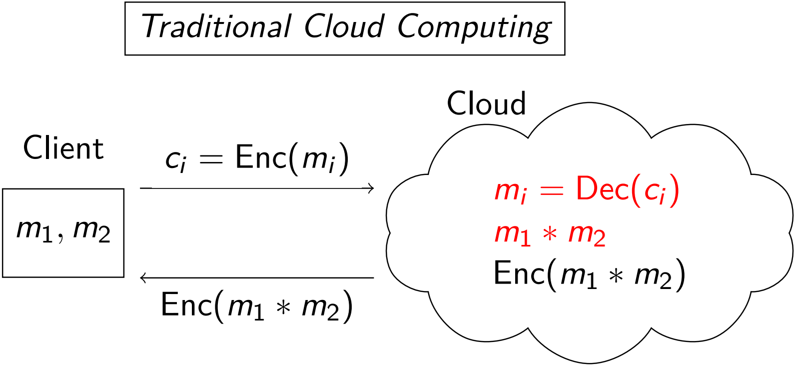

Figure 2: Usage of traditional cloud computing - unencrypted

FHE allows secure cloud computations

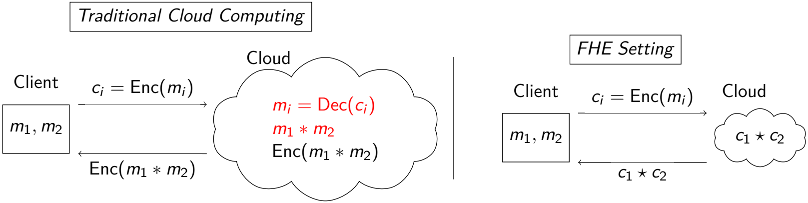

Figure 2: Usage of FHE in the public cloud - always encrypted

FHE allows secure cloud computations

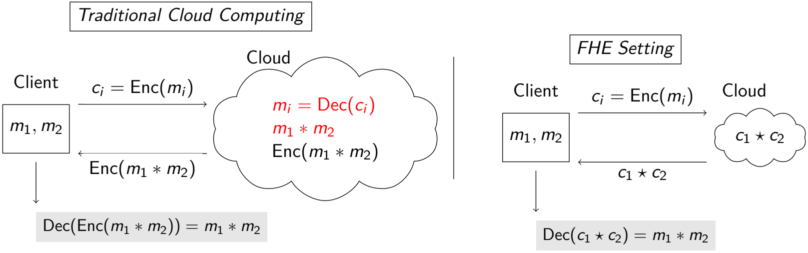

Figure 2: Usage of FHE in the public cloud - always encrypted

Functional completeness

Theorem (Functional Complete Set)

The ability to evaluate any function homomorphically is achievable if addition and multiplication can be performed homomorphically and can be iterated, since they constitute a functionally complete set over finite rings.

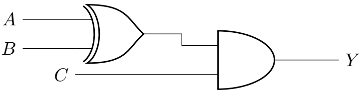

Figure 3: Example Circuit with XOR and AND

Procedures in (correct) HE schemes

Table 1: Algorithms and keys of HE vs. classic encryption

		classic encryption	homomorphic encryption
keys	SK	•	•
	PK	•	•
	EK	◦	•
procedure	KeyGen	•	•
	Enc	•	•
	Dec	•	•
	Eval	◦	•
	Refresh	◦	•

Definition ((correct) Eval)

$\mathrm{Eval}(\mathrm{EK}, f, c) \to c'$

Procedures in (correct) HE schemes

Table 1: Algorithms and keys of HE vs. classic encryption

		classic encryption	homomorphic encryption
keys	SK	•	•
	PK	•	•
	EK	◦	•
procedure	KeyGen	•	•
	Enc	•	•
	Dec	•	•
	Eval	◦	•
	Refresh	◦	•

Definition ((correct) Eval)

$\mathsf{Eval}(\mathsf{EK}, f, c) \to c'$:

$$\mathsf{Dec}(c') = \mathsf{Dec}[\mathsf{Eval}(\mathsf{EK}, f, c)] = f(m).$$

Correctness

We assume correctness here. Formally correct the Eval function just returns a ciphertext c'.

Procedures in (correct) HE schemes

Table 1: Algorithms and keys of HE vs. classic encryption

		classic encryption	homomorphic encryption
	SK	$\bullet$	$\bullet$
keys	PK	$\bullet$	$\bullet$
	EK	$\circ$	$\bullet$
	KeyGen	$\bullet$	$\bullet$
	Enc	$\bullet$	$\bullet$
procedure	Dec	$\bullet$	$\bullet$
	Eval	$\circ$	$\bullet$
	Refresh	$\circ$	$\bullet$

Definition ((correct) Eval)

$\mathrm{Eval}(\mathrm{EK}, f, c) \rightarrow c'$

$$\mathrm{Dec}(c') = \mathrm{Dec}[\mathrm{Eval}(\mathrm{EK}, f, c)] = f(m).$$

Definition (Refresh)

$\mathrm{Refresh}(\mathrm{EK}, c, \mathrm{flag}) \rightarrow c'$:

$$\mathrm{noise}(c') < \mathrm{noise}(c)$$

Correctness

We assume correctness here. Formally correct the Eval function just returns a ciphertext c'.

Properties of FHE

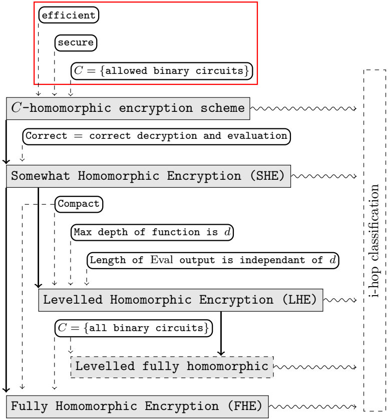

Figure 4: Classification of FHE

- efficient: run in polynomial time in relation to the security parameter λ
- secure: IND-CPA secure
- C: allowed binary circuits

Properties of FHE

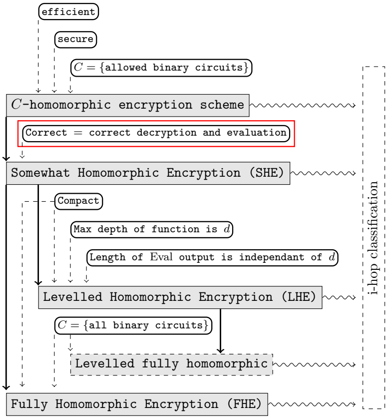

Figure 4: Classification of FHE

- correct:
  - decrypt the encryption of a message without any error
  - for all functions $f \in C$, it can correctly decrypt the results of the evaluation of f over fresh ciphertexts with overwhelming probability

Properties of FHE

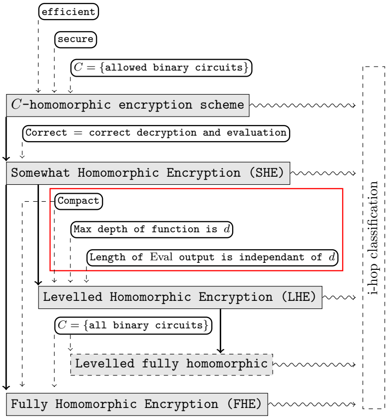

Figure 4: Classification of FHE

- compact: the output of the Eval function is not bigger than $p(\lambda)$ bits, independent of the complexity of the evaluated function f
- Max depth of function is d
- Length of Eval output is independant of d

Properties of FHE

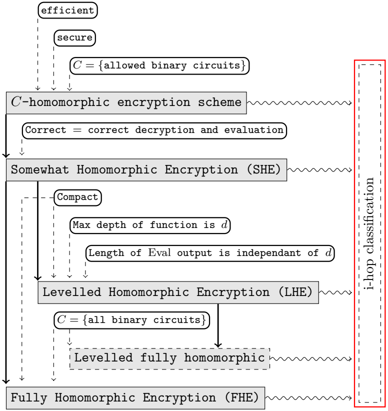

Figure 4: Classification of FHE

Remark (i-hop correctness)

Evaluating an arbitrary function is not equal to consecutively evaluating arbitrary many functions.

$$f(\dots(f(m))) := F_n(m) \rightarrow \mathrm{Eval}(\mathrm{EK}, F_n)\checkmark$$

$$\mathrm{Eval}\left(\mathrm{EK}, f\left(\dots\left(\mathrm{Eval}(\mathrm{EK}, f)\right)\right)\right) \rightarrow \lightning$$

Notes on classification

Definition (Circuit Privacy)

A C-homomorphic encryption scheme is (perfectly, statistically or computationally) *circuit private* if $D_1 = \mathsf{Eval}(\mathsf{EK}, f, c)$ and $D_2 = \mathsf{Enc}(\mathsf{PK}, f(m))$ are (perfectly, ...) indistinguishable.

Notes on classification

Definition (Circuit Privacy)

A C-homomorphic encryption scheme is (perfectly, statistically or computationally) *circuit private* if $D_1 = \mathsf{Eval}(\mathsf{EK}, f, c)$ and $D_2 = \mathsf{Enc}(\mathsf{PK}, f(m))$ are (perfectly, ...) indistinguishable.

Table 2: Circuit Privacy vs. Function Privacy

Privacy	Distributions of ... are the same	
Circuit	Eval output of f_1	fresh ciphertexts
Function	Eval output of f_1	Eval output of f_2

FHE does not hide the structure of ML models

FHE generations

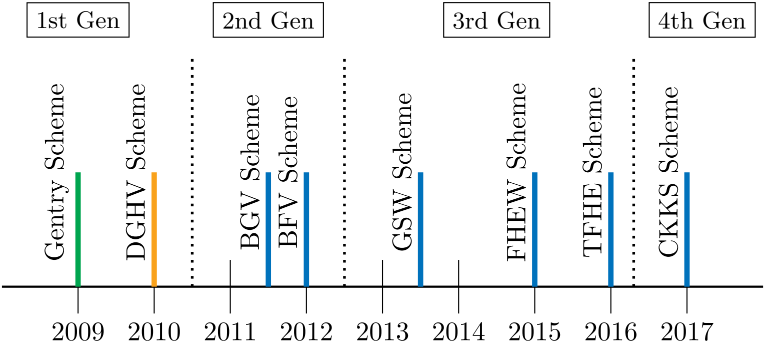

Figure 5: Timeline of the main FHE schemes.
■ Schemes based on ideal lattices, ■ Schemes based on AGCD,
■ Schemes based on LWE and RLWE [1]

FHE generations

Table 3: Comparison of FHE generations

SCHEMES		2nd Generation BGV BFV	3rd Generation TFHE	4th Generation CKKS
		Integer Arithmetic	Bitwise operations	Real Number Arithmetic
FAST OPERATIONS	scalar mult	●	●	●
	arithmetic	●	●	●
	non-arithmetic	○	●	○
PROPERTIES	fast bootstrapping	○	●	●[2]
	fast packing/ batching/ SIMD	●	○	●
	levelled design	●	●	●
PROS	fast	scalar multiplication linear functions	number comparison -	polynomial approx. multiplicative inverse
	efficient	-	boolean circuits	DFT, logistic regression
CONS		slow non-linear functions	-	slow non-linear functions
USAGE		large arrays of numbers	bit-wise operations	real numbers arithmetic

[2] CKKS has a fast amortized bootstrapping procedure.

From SHE to FHE

Noise reducing techniques

noise growth $\rightarrow$ Refresh procedure needed

- bootstrapping
- key-switching
  - re-linearization
  - modulus switching

From SHE to FHE

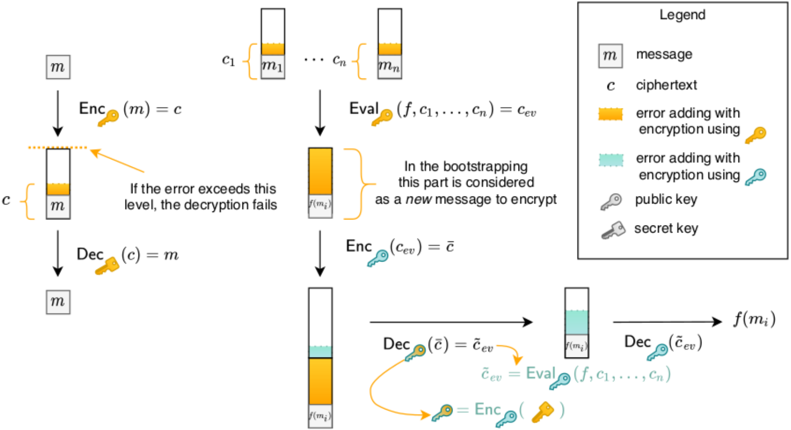

Figure 6: Illustration of the *bootstrapping* technique by Marcolla et al. [1]

Security

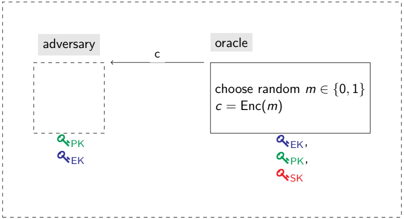

Figure 7: IND-CPA Security

Definition (IND-CPA Security)

The scheme is *IND-CPA* secure if for an efficient adversary $\mathcal{A}$, it holds that:

$$\Pr[\mathcal{A}(\mathrm{PK}, \mathrm{EK}, \mathrm{Enc}_{\mathrm{PK}}(0)) = 1] - \Pr[\mathcal{A}(\mathrm{PK}, \mathrm{EK}, \mathrm{Enc}_{\mathrm{PK}}(1)) = 1] = \mathrm{negl}(\lambda)$$

where $(\mathrm{SK}, \mathrm{PK}, \mathrm{EK}) \leftarrow \mathrm{KeyGen}(\lambda)$.

Security

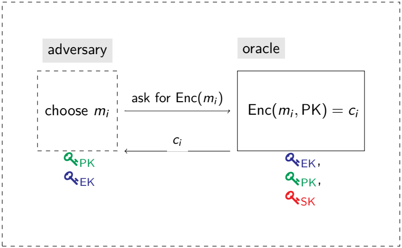

Figure 7: IND-CPA Security repeat $p(\lambda)$ times

Definition (IND-CPA Security)

The scheme is *IND-CPA* secure if for an efficient adversary $\mathcal{A}$, it holds that:

$$\Pr[\mathcal{A}(\text{PK}, \text{EK}, \text{Enc}_{\text{PK}}(0)) = 1] -$$
$$\Pr[\mathcal{A}(\text{PK}, \text{EK}, \text{Enc}_{\text{PK}}(1)) = 1] = \text{negl}(\lambda)$$

where $(\text{SK}, \text{PK}, \text{EK}) \leftarrow \text{KeyGen}(\lambda)$.

Security

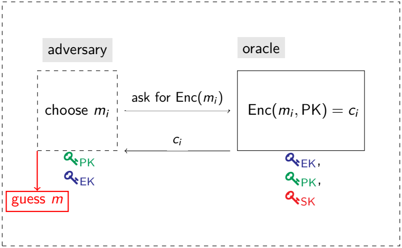

Figure 7: IND-CPA Security

Definition (IND-CPA Security)

The scheme is *IND-CPA* secure if for an efficient adversary $\mathcal{A}$, it holds that:

$$\Pr\left[\mathcal{A}\left(\mathrm{PK}, \mathrm{EK}, \mathrm{Enc}_{\mathrm{PK}}(0)\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathrm{PK}, \mathrm{EK}, \mathrm{Enc}_{\mathrm{PK}}(1)\right) = 1\right] = \mathrm{negl}(\lambda)$$

where $(\mathrm{SK}, \mathrm{PK}, \mathrm{EK}) \leftarrow \mathrm{KeyGen}(\lambda)$.

Security

Theorem

IND-CPA security is only achievable if the encryption scheme randomizes ciphertexts.

Proof.

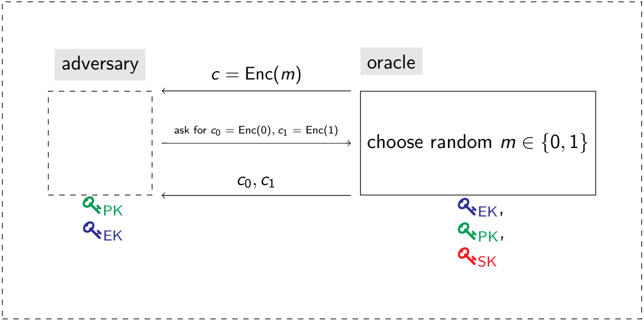

Figure 8: IND-CPA Security is only achievable with randomization

Theorem

IND-CPA security is only achievable if the encryption scheme randomizes ciphertexts.

Proof.

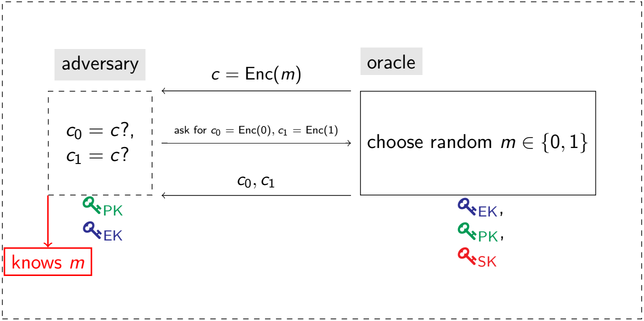

Figure 8: IND-CPA Security is only achievable with randomization

Theorem

By their design, HE schemes can not achieve indistinguishability under adaptive chosen ciphertext attack (IND-CCA2) security.

Proof.

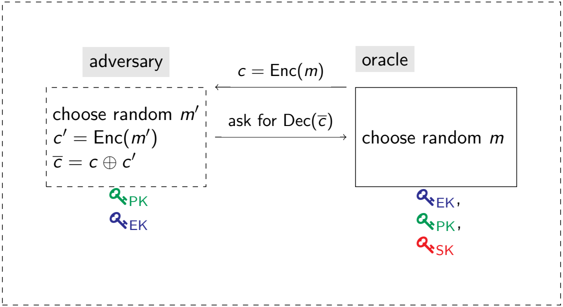

Figure 9: IND-CCA2 Security is not achievable

Security

Theorem

By their design, HE schemes can not achieve indistinguishability under adaptive chosen ciphertext attack (IND-CCA2) security.

Proof.

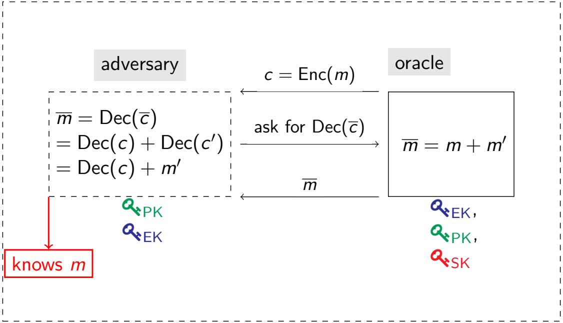

Figure 9: IND-CCA2 Security is not achievable

Security: malicious adversary

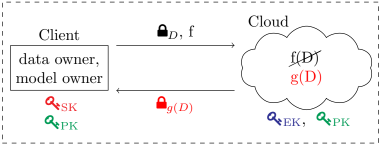

Figure 10: Malicious adversaries are a problem

Possible solutions

- known evaluation results
- statistics
- Trusted Execution Environments
- homomorphic hashes

Additional Notes on Security

The security of FHE

- is based on LWE/ RLWE,
- is considered quantum safe,
- can be implemented leakage resilient,
- can be circuit/ function private,
- allows key evolution,
- and no decryption is needed for outsourcing computations.

Additional Notes on Security

The security of FHE

- is based on LWE/ RLWE,
- is considered quantum safe,
- can be implemented leakage resilient,
- can be circuit/ function private,
- allows key evolution,
- and no decryption is needed for outsourcing computations.

Table 4: Circular Security vs. KDM Security

circular security	KDM
$\mathsf{Enc}(\mathsf{PK}, \mathsf{SK})$	$\mathsf{Enc}(\mathsf{PK}_2, \mathsf{SK}_1)$

Limitations

Table 5: Main limitations of FHE and their solution

Limitation	potential solution
computational overhead	Hardware acceleration and better packing techniques
lack of standardization	Homomorphic Encryption Standard and stable open source libraries
hard to use	High level compilers like HElayers

Table 6: Running times of multiplying 2 bits homomorphically [2]

Year	runtime	speedup	speedup per year
2009	30 min	-	-
2014	2000 ns	$9 \cdot 10^8$	$18 \cdot 10^7$
2020	100 ns	20	3.33
	. . . Hardware Acceleration . . .		
2024	*0.1 ns*	*1000*	*250*

Limitations

Table 5: Main limitations of FHE and their solution

Limitation	potential solution
computational overhead	Hardware acceleration and better packing techniques
lack of standardization	Homomorphic Encryption Standard and stable open source libraries
hard to use	High level compilers like HElayers

Industry:

1. Microsoft
2. Samsung SDS
3. Intel
4. Duality Technologies
5. IBM
6. Google
7. SAP
8. . . .

Government:

1. NIST
2. SLAC National Accelerator Lab
3. United Nations / ITU

Limitations

Table 5: Main limitations of FHE and their solution

Limitation	potential solution
computational overhead	Hardware acceleration and better packing techniques
lack of standardization	Homomorphic Encryption Standard and stable open source libraries
hard to use	High level compilers like HElayers

Compilers adress engineering challenges

- parameter selection
- plaintext encoding
- data-independent execution
- ciphertext maintenance

Beyond Homomorphic Encryption

	FHE	MPC	TEE
no communication	●	○	●
no computational overhead	○	●	●
no known attacks	●	●	○
security based on	LWE, RLWE	protocols	hardware

Figure 11: Simplified comparison of FHE, MPC and TEE. MPC has a large communication overhead, FHE is computational expensive and TEEs are often proven to be vulnerable against side-channel attacks.

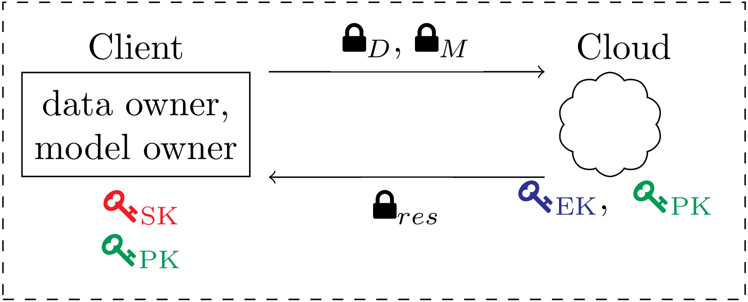

Figure 12: FHE basic use case

More information on use case

Used Techniques

- model: XGBoost
- scheme: CKKS
- library: to be chosen
- framework: HElayers (IBM)
- dataset: Bank Marketing
- benchmarking modes:
  - all-in-one
  - batch

Evaluation metrics

- latency
- throughput
- accuracy
- libraries
- parameters
- (dataset)
- (compressed model)

Summary

1. Fully Homomorphic Encryption
    - Properties
    - Classification - historical and formal
    - Security
    - Beyond
    - (Implementations)
2. Use Cases
    - (General)
    - Specific use case

Future Developments

Implement and analyze the use case with HeLayers

Summary

1. Fully Homomorphic Encryption
    - Properties
    - Classification - historical and formal
    - Security
    - Beyond
    - (Implementations)
2. Use Cases
    - (General)
    - Specific use case

Future Developments

Implement and analyze the use case with HeLayers

Thank you for your attention - Any questions?

Summary

Contribution:

- adding efficiency, security to properties
- distinguish between plain- and ciphertext operations
- increased understanding of i-hop correctness
- security described with practical implications
- KDM vs. circular security
- incorrect evaluation solutions
- limitations of FHE and positioning in cryptography
- overview of most common use cases

Future Developments

Implement and analyze the use case with HeLayers

Thank you for your attention - Any questions?

Link to the slides

Figure 13: Link to the presentation slides

References

See References in the paper of the master seminar and

[1] Frederik Armknecht et al. "A guide to fully homomorphic encryption". In: *Cryptology ePrint Archive* (2015).

[2] Duality. *The HomomorphicEncryption.org Community and the Applied Fully Homomorphic Encryption Standardization Efforts*. `https://csrc.nist.gov/csrc/media/Presentations/2023/stppa6-fhe/images-media/20230725-stppa6-he-fhe--kurt-rohloff.pdf`. Accessed: 2024-01-29. July 2023.

[3] Ph.D. Madnick Stuart E. *The Continued Threat to Personal Data: Key Factors Behind the 2023 Increase*. Tech. rep. Accessed: 18.02.2024. Apple, Dec. 2023.

Encryption during Processing

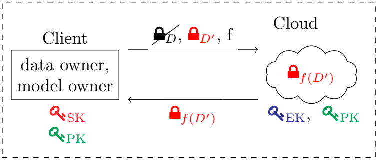

Figure 14: Problem: Malleability during processing

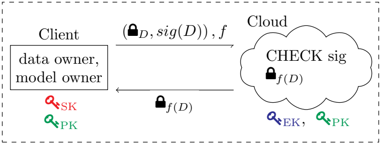

Figure 15: Solution: signature $sig(D) = \mathsf{Enc}_{\mathsf{normal}}\left(h(D), \mathsf{k}_{\mathsf{priv}}\right)$

Remark (Other solution)

Use traditional encrypted transport protocols additionally to FHE encryption
$\rightarrow$ small overhead, but implemented and known

Beyond Homomorphic Encryption

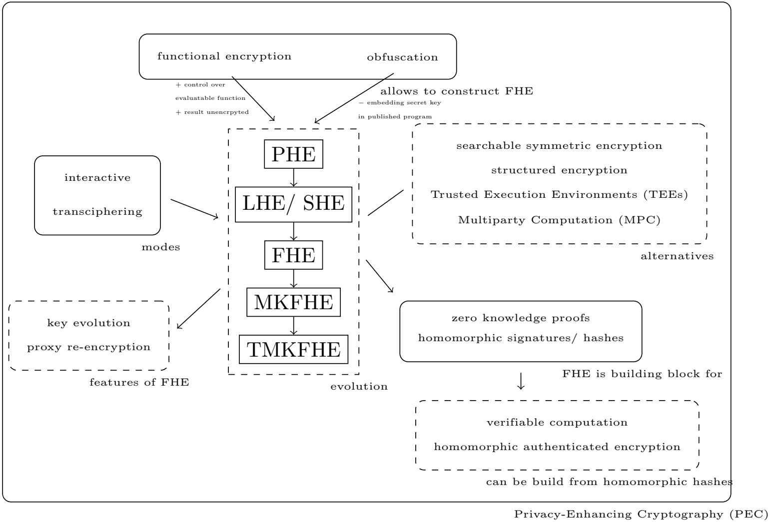

Figure 16: Beyond FHE

More Use Cases

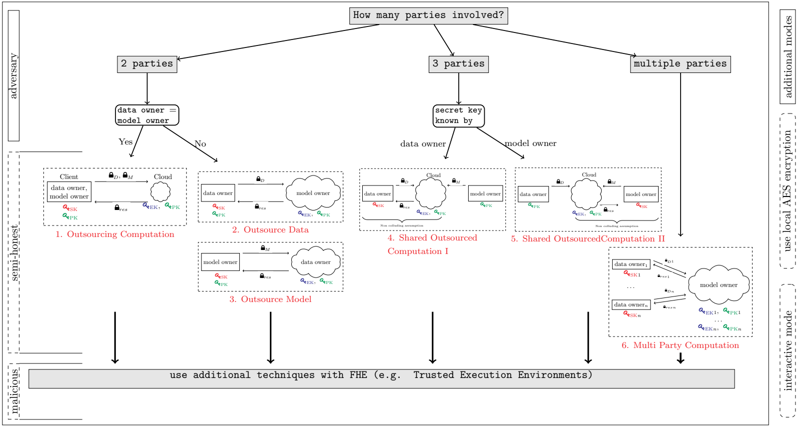

Figure 17: FHE use cases

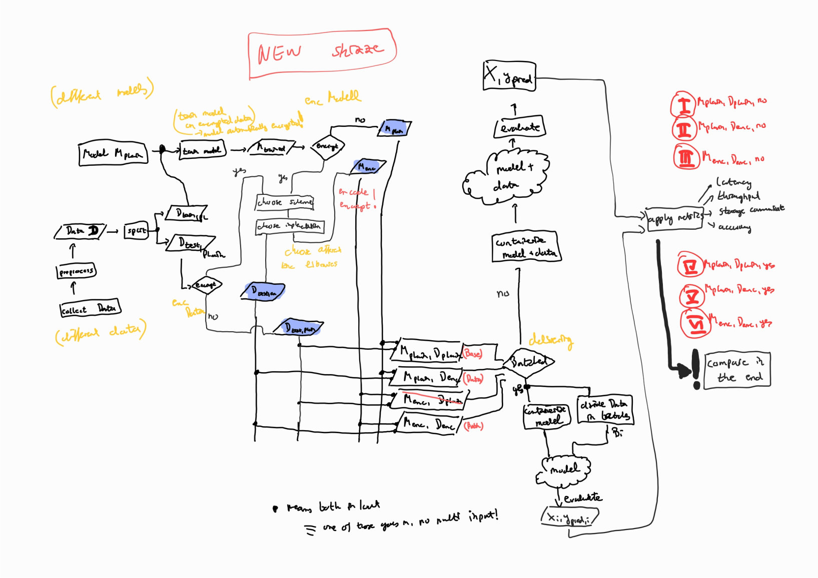

Figure 18: ML pipeline with FHE

Overview Schemes

Operation	BFV	BGV	CKKS	FHEW	TFHE
Native Add/Sub	$\bullet$	$\bullet$	$\bullet$	$\circ$	$\circ$
Native Mult	$\bullet$	$\bullet$	$\bullet$	$\circ$	$\circ$
SIMD	$\bullet$	$\bullet$	$\bullet$	$(\bullet)$	$(\bullet)$
Boolean Logic	$\circ$	$\bullet$	$\circ$	$\bullet$	$\bullet$
$< 1s$ Bootstrapping	$\circ$	$\circ$	$\circ$	$\bullet$	$\bullet$

Figure 19: Schemes

Overview Libraries

	Library	Language	BGV	BFV	FHEW	TFHE	CKKS
			Schemes				
in HeLayers	HEAAN	C++	○	○	○	○	●
in HeLayers	HElib	C++	●	○	○	○	●
in HeLayers	PALISADE	C++	●	●	●	●	●
in HeLayers	OpenFHE	C++	●	●	●	●	●
in HeLayers	Lattigo	Go	●	●	○	○	●
in HeLayers	SEAL	C++/ C#	●	●	○	○	●
	FHEW	C++	○	○	●	○	○
	TFHE	C++/ C	○	○	○	●	○
	concrete	Rust	○	○	○	●	○
	RNS-HEAAN	C++	○	○	○	○	●
	FV-NFLlib	C++	○	●	○	○	○
	CuFHE	Cuda/C++	○	○	○	●	○
	NuFHE	Python	○	○	○	●	○

Figure 20: Libraries

Overview Frameworks

Compiler	Language	Library					
		HElib	SEAL	PALISADE	FHEW	TFHE	HEAAN
ALCHEMY	Haskell	○	○	○	○	○	○
Cingulata	C++	○	○	○	○	●	○
E^3	C++	●	●	●	●	●	○
SHEEP	C++	●	●	●	○	●	○
EVA	C++	○	●	○	○	○	○
Marble	C++	●	●	○	○	○	○
RAMPARTS	Julia	○	○	●	○	○	○
Transpiler	C++	○	○	●	○	●	○
CHET	C++	○	●	○	○	○	●
nGraph-HE	C++	○	●	○	○	○	○
SEALion	C++	○	●	○	○	○	○
HElayers	C++, python API	●	●	●	○	○	●

Figure 21: Compilers/ Frameworks