

Fully Homomorphic Encryption and its Use Cases

Master Seminar in Scientific Computing

Felix P. Paul

Johannes Gutenberg-Universität Mainz

March 5, 2024

Data breaches in the cloud

3x

The number of data breaches **more than tripled** between 2013 and 2022.^{21,22}

360
million

In the first eight months of 2023 alone, **over 360 million people were victims of corporate and institutional data breaches.**²⁵

1 of 4

In the first three quarters of 2023, one in four people in the US had their health data exposed in a data breach.^{26,27}

98%

98% of organizations have a relationship with a vendor that experienced a data breach within the last two years.¹³

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

FHE allows secure cloud computations

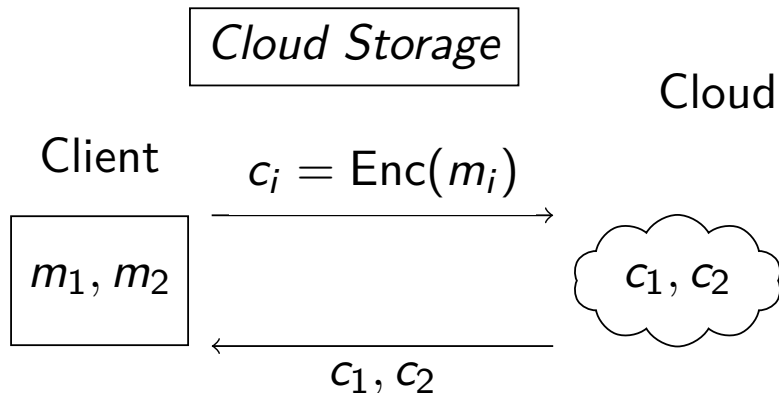


Figure 2: Usage of cloud storage - always encrypted

FHE allows secure cloud computations

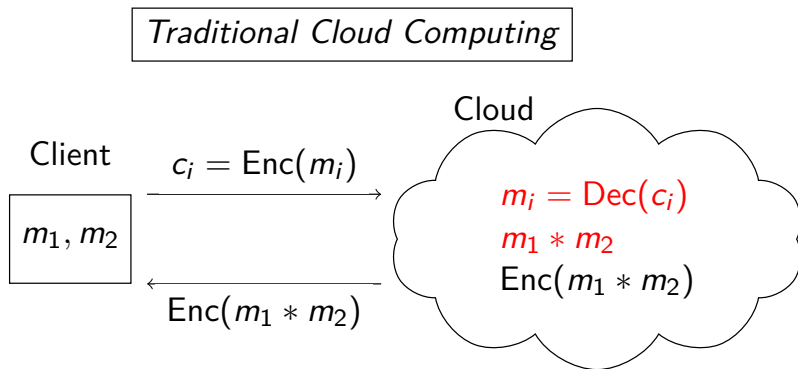


Figure 2: Usage of traditional cloud computing - unencrypted

FHE allows secure cloud computations

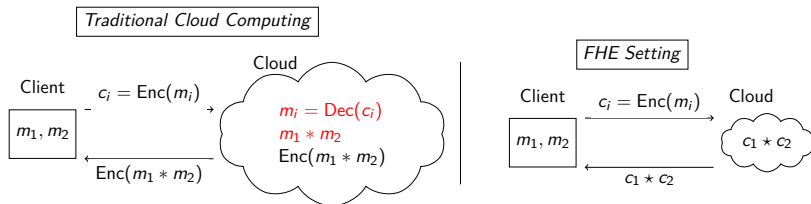


Figure 2: Usage of FHE in the public cloud - always encrypted

FHE allows secure cloud computations

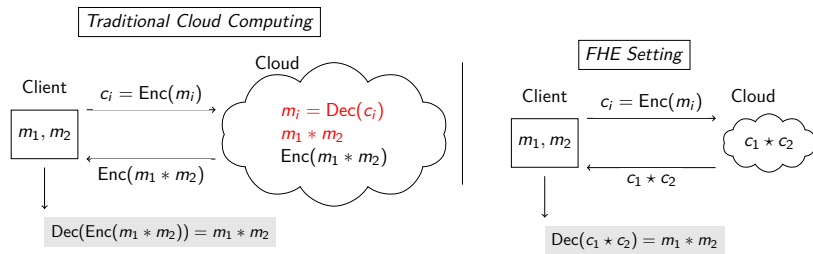


Figure 2: Usage of FHE in the public cloud - always encrypted

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.

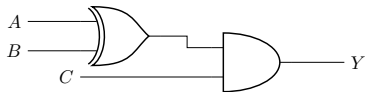


Figure 3: Example Circuit with XOR and AND

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)

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

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)

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

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

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
keys	SK	•	•
	PK	•	•
	EK	○	•
procedure	KeyGen	•	•
	Enc	•	•
	Dec	•	•
	Eval	○	•
	Refresh	○	•

Definition ((correct) Eval)

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

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

Definition (Refresh)

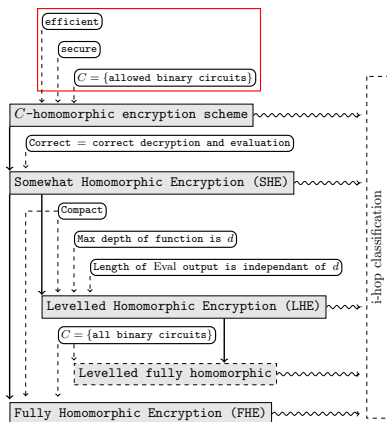
$$\text{Refresh}(\text{EK}, c, \text{flag}) \rightarrow c':$$

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

Correctness

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

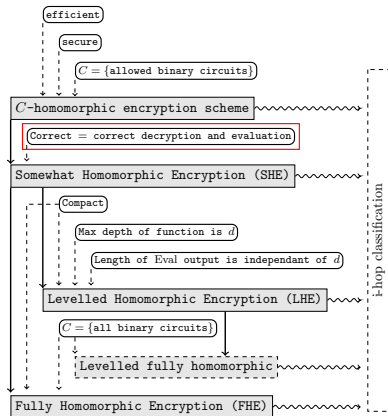
Properties of FHE



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

Figure 4: Classification of FHE

Properties of FHE

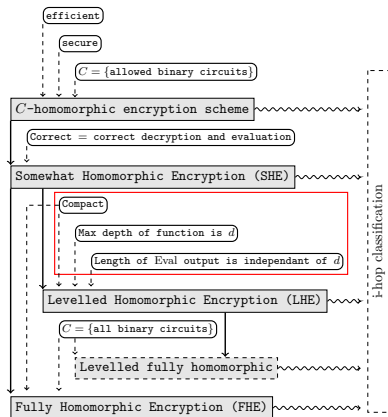


► correct:

- 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

Figure 4: Classification of FHE

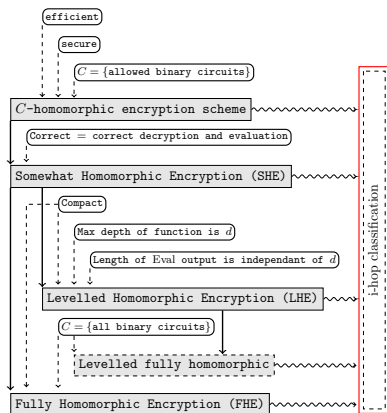
Properties of FHE



- 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

Figure 4: Classification of FHE

Properties of FHE



Remark (i-hop correctness)

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

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

$$\text{Eval}(\text{EK}, f(\dots(\text{Eval}(\text{EK}, f)))) \rightarrow \text{!}$$

Figure 4: Classification of FHE

Definition (Circuit Privacy)

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

Notes on classification

Definition (Circuit Privacy)

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

Table 2: Circuit Privacy vs. Function Privacy

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

FHE does not hide the structure of ML models

FHE generations

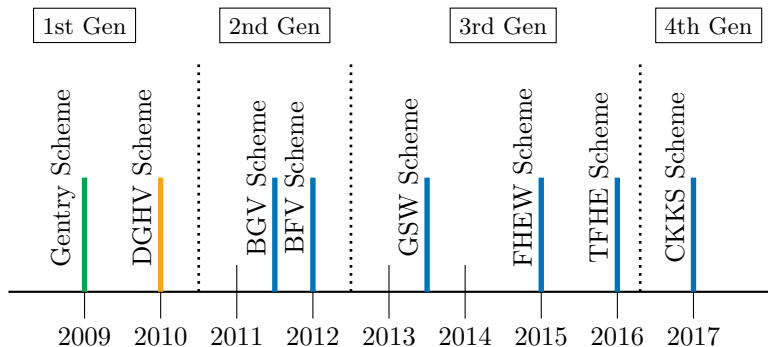


Figure 5: Timeline of the main FHE schemes.

- 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.
	efficient	-	-	boolean circuits	multiplicative inverse DFT, logistic regression
CONS		slow non-linear functions		-	slow non-linear functions
USAGE		large arrays of numbers		bit-wise operations	real numbers arithmetic

²CKKS 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

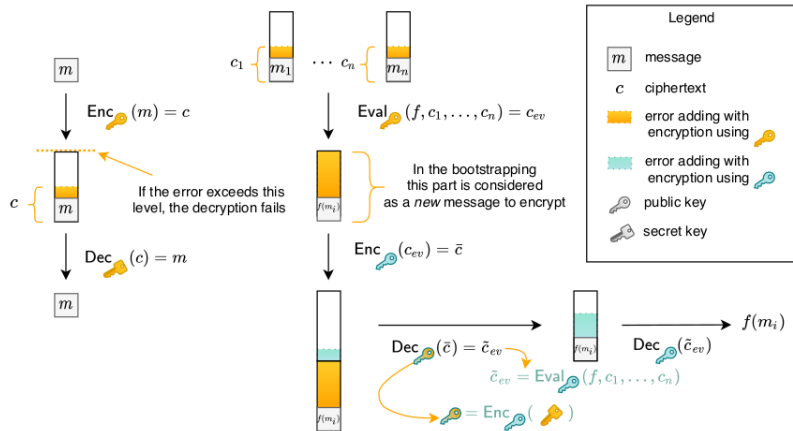


Figure 6: Illustration of the *bootstrapping* technique by Marcolla et al. [1]

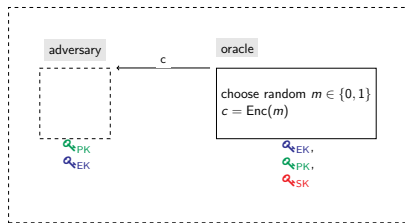


Figure 7: IND-CPA Security

Definition (IND-CPA Security)

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

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

where

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

Security

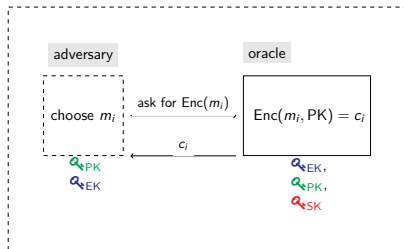


Figure 7: IND-CPA Security
repeat $p(\lambda)$ times

Definition (IND-CPA Security)

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

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

where

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

Security

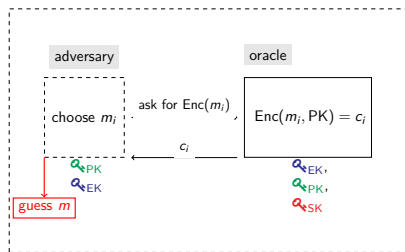


Figure 7: IND-CPA Security

Definition (IND-CPA Security)

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

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

where

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

Security

Theorem

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

Proof.

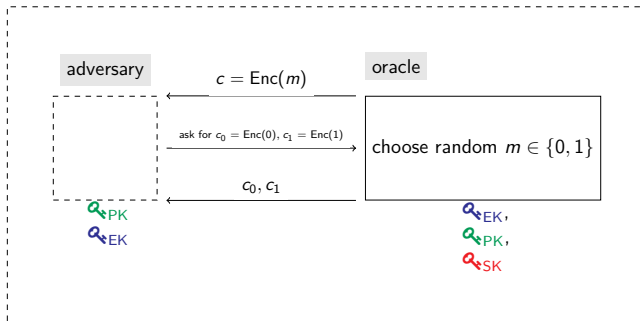


Figure 8: IND-CPA Security is only achievable with randomization

Security

Theorem

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

Proof.

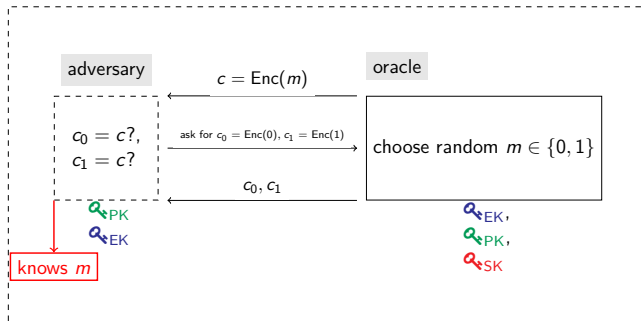


Figure 8: IND-CPA Security is only achievable with randomization

Security

Theorem

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

Proof.

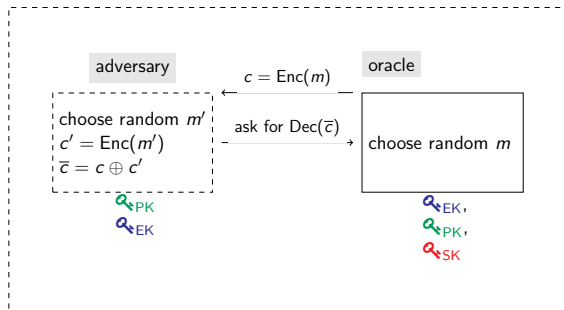


Figure 9: IND-CCA2 Security is not achievable

Security

Theorem

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

Proof.

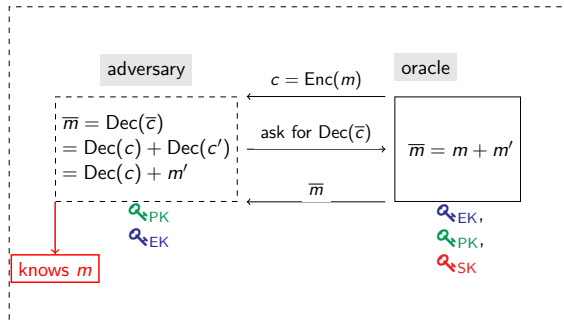


Figure 9: IND-CCA2 Security is not achievable

Security: malicious adversary

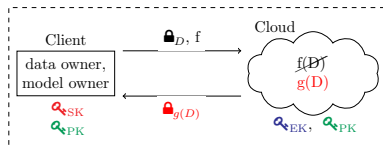


Figure 10: Malicious adversaries are a problem

Possible solutions

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

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.

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
$\text{Enc}(\text{PK}, \text{SK})$	$\text{Enc}(\text{PK}_2, \text{SK}_1)$

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.

Use case in master thesis

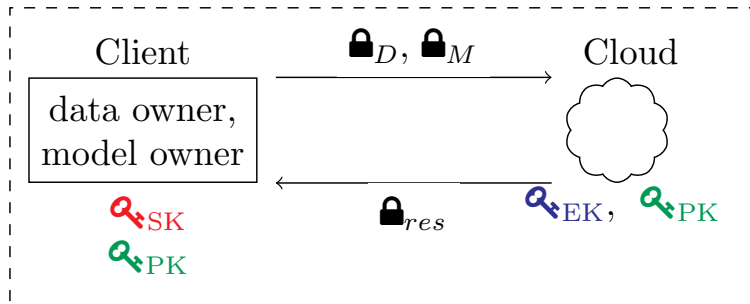


Figure 12: FHE basic use case

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?

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

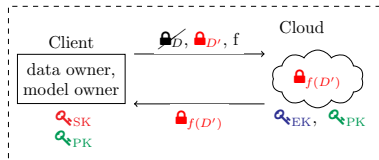


Figure 13: Problem: Malleability during processing

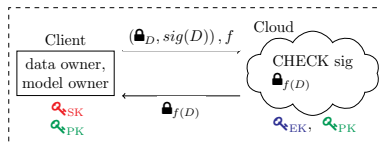


Figure 14: Solution: signature
 $sig(D) = \text{Enc}_{\text{normal}}(h(D), k_{\text{priv}})$

Remark (Other solution)

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

Beyond Homomorphic Encryption

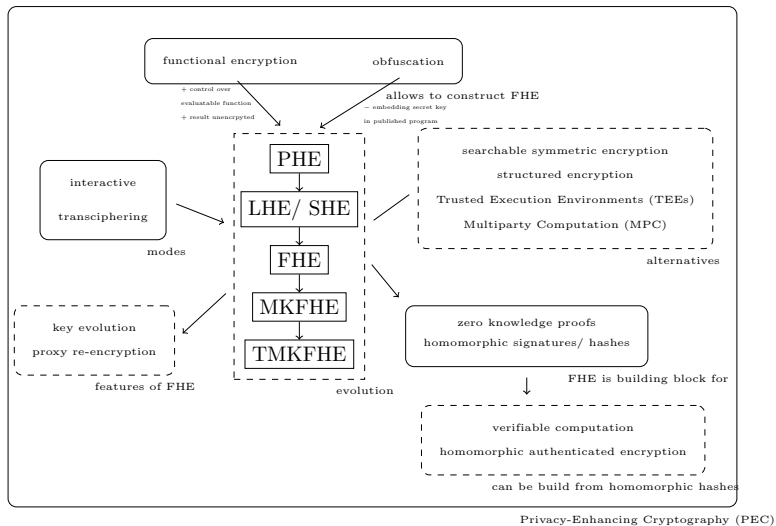


Figure 15: Beyond FHE

More Use Cases

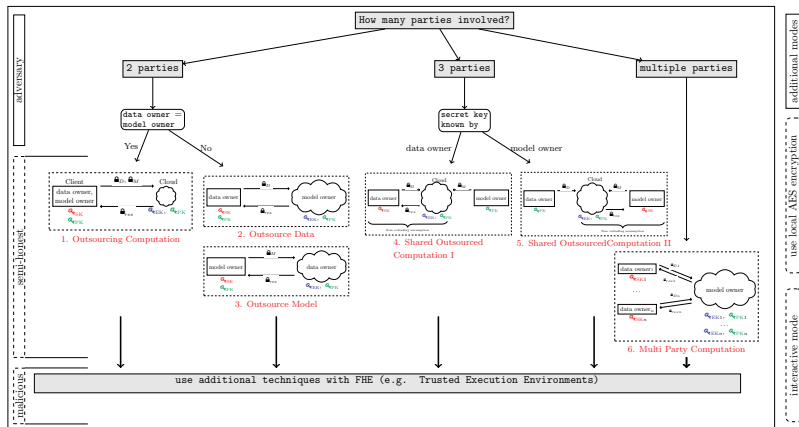


Figure 16: FHE use cases

Use Case Implementation

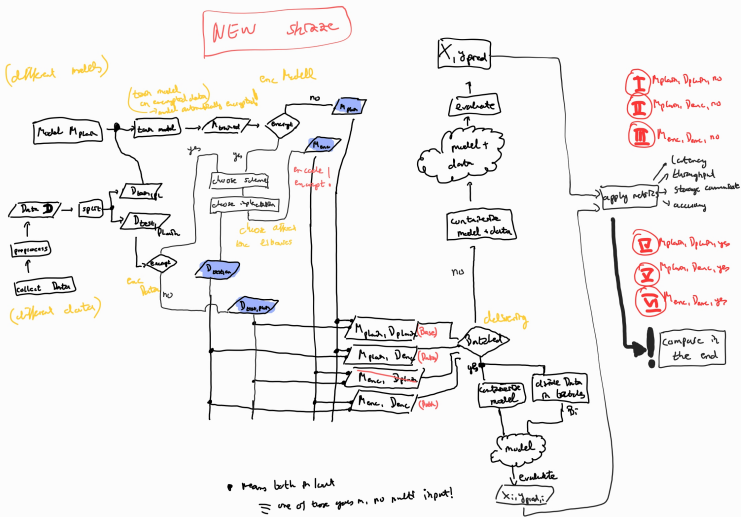


Figure 17: ML pipeline with FHE

Overview Schemes

Operation	BFV	BGV	CKKS	FHEW	TFHE
Native Add/Sub	●	●	●	○	○
Native Mult	●	●	●	○	○
SIMD	●	●	●	(●)	(●)
Boolean Logic	○	●	○	●	●
< 1s Bootstrapping	○	○	○	●	●

Figure 18: Schemes

Overview Libraries

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

Figure 19: 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 20: Compilers/ Frameworks