3MI LABS

COED Technologies

What they can and can't do

Cyprien de Saint Guilhem

2025-06-03

Introduction

The Purpose of Cryptography

Cryptographic technologies provide security properties:

- ◇ Confidentiality (**only** the intended recipient can read a message)
- ◇ Authentication (**only** the expected sender can write a message)
- ◇ Integrity (**no one** can change a message after it's been written)

The Purpose of Cryptography

Cryptographic technologies provide security properties:

- ◇ Confidentiality (**only** the intended recipient can read a message)
- ◇ Authentication (**only** the expected sender can write a message)
- ◇ Integrity (**no one** can change a message after it's been written)

The cryptography used depends on *who* is the **sender** and the **receiver**.

- ◇ If **sender = receiver** ⇒ *data at rest* ⇒ symmetric cryptography
- ◇ If **sender ≠ receiver** ⇒ *data in transit* ⇒ public-key cryptography

(With key-exchange, we can bootstrap symmetric cryptography from public-key cryptography.)

What's Next?

After data **at rest** and **in transit**, what about **data in use**?

Can we hope to provide security guarantees for data *while it is being used*?

- ◇ Confidentiality?
- ◇ Authentication?
- ◇ ~~Integrity?~~ Verifiability?

Problem Statement

Big data contains value, but extracting it requires computing power.

On the other hand, companies have a duty to the data’s privacy.

- Small companies need to delegate the computation. They have to
  1. share their data in plaintext for the “cloud” to perform computation,
  2. trust that the “cloud” (a) will perform the computation honestly, and (b) won’t look at the data.

Problem Statement

Big data contains value, but extracting it requires computing power.

On the other hand, companies have a duty to the data's privacy.

- ◇ Small companies need to delegate the computation. They have to
  1. share their data in plaintext for the "cloud" to perform computation,
  2. trust that the "cloud" (a) will perform the computation honestly, and (b) won't look at the data.
- ◇ Big companies don't need to delegate,
  - ▶ but have to protect against breaches, and
  - ▶ could choose to offer "never decrypted" services.

Problem Statement

Big data contains value, but extracting it requires computing power.

On the other hand, companies have a duty to the data's privacy.

- ◇ Small companies need to delegate the computation. They have to
  1. share their data in plaintext for the "cloud" to perform computation,
  2. trust that the "cloud" (a) will perform the computation honestly, and (b) won't look at the data.
- ◇ Big companies don't need to delegate,
  - ▶ but have to protect against breaches, and
  - ▶ could choose to offer "never decrypted" services.
- ◇ Not all data can legally be aggregated to be "big enough" (e.g. hospital databases).

The technologies we'll talk about today

1. Fully homomorphic encryption (FHE)

   Computation on ciphertexts. Conceptually straightforward, but with an arithmetic model of computation. Computationally expensive.

2. Multi-party computation (MPC)

   Computation on *secret-shared data*. Also with arithmetic model of computation. Less expensive but participants have a symmetric role. It *requires* multiple computing parties.

3. Zero-knowledge proofs (ZKP)

   For *proving* and *verifying* computation (with or without secrets). Expensive to generate on top of the computation itself, but cheap to verify at the receiving end.

The Technologies

Fully Homomorphic Encryption

Works like a traditional public-key encryption scheme:

1. A public/private key pair is generated;
2. Plaintext data is encrypted into ciphertext data;
3. The holder of the private key is able to decrypt ciphertexts.

Fully Homomorphic Encryption

Works like a traditional public-key encryption scheme:

1. A public/private key pair is generated;
2. Plaintext data is encrypted into ciphertext data;
3. The holder of the private key is able to decrypt ciphertexts.

The big difference: **anyone** can perform **operations** on ciphertexts.

Fully Homomorphic Encryption

Works like a traditional public-key encryption scheme:

1. A public/private key pair is generated;
2. Plaintext data is encrypted into ciphertext data;
3. The holder of the private key is able to decrypt ciphertexts.

The big difference: **anyone** can perform **operations** on ciphertexts.

- ◇ ***Additively*** *homomorphic encryption*: as many **additions** as we want;
- ◇ ***Multiplicatively*** *homomorphic encryption*: as many **multiplications** as we want.

Fully Homomorphic Encryption

Works like a traditional public-key encryption scheme:

1. A public/private key pair is generated;
2. Plaintext data is encrypted into ciphertext data;
3. The holder of the private key is able to decrypt ciphertexts.

The big difference: **anyone** can perform **operations** on ciphertexts.

◇ ***Additively*** *homomorphic encryption*: as many **additions** as we want;

◇ ***Multiplicatively*** *homomorphic encryption*: as many **multiplications** as we want.

Each of these is easy to achieve *on its own*. First scheme to do both appeared in ~2015. Efficiency has improved since but is plateauing. Next potential improvement is from hardware.

Fully Homomorphic Encryption: Challenges

Noise growth:

- ◇ Ciphertexts contain noise which grows with each operation
- ◇ Need to perform **bootstrapping** operations to reduce noise (encrypted decryption)

Fully Homomorphic Encryption: Challenges

Noise growth:

- Ciphertexts contain noise which grows with each operation
- Need to perform **bootstrapping** operations to reduce noise (encrypted decryption)
- Different schemes exist:
  - Some with fast bootstrapping, but quick noise growth (bootstrap often)
  - Some with slow bootstrapping, but slower noise growth

Fully Homomorphic Encryption: Challenges

Noise growth:

- ◇ Ciphertexts contain noise which grows with each operation
- ◇ Need to perform **bootstrapping** operations to reduce noise (encrypted decryption)
- ◇ Different schemes exist:
  - ▶ Some with fast bootstrapping, but quick noise growth (bootstrap often)
  - ▶ Some with slow bootstrapping, but slower noise growth

Expansion cost: ciphertexts are much larger than plaintext

- ◇ 10x increase for small parameters with low noise budget
- ◇ > 40x increase for large parameters that allow more computation

Fully Homomorphic Encryption: Challenges

- Programming is not easy:
  - Control flow needs to preserve privacy. (How to stop loops? How to hide branching choices?)
  - Floating point numbers are hard to represent with arithmetic data and operations.
  - (Libraries exist to help.)

Fully Homomorphic Encryption: Trust Setting

The private key holder *can decrypt everything*; the computation provider **must be a different entity**.

- The **creator** of the key has to be the **party receiving the output** (or give key-switching keys).
- The input provider(s) needs the public key to perform encryption (also for transciphering).
- The computation provider does not need to know any private key. (They might need to know an *encryption of the private key* to perform bootstrapping or key switching.)

Fully Homomorphic Encryption: Trust Setting

Importantly the output receiver cannot tell *how* the output ciphertext was computed.

It needs to **trust** the computation provider that the program was computed as requested.

In short, **FHE computation**, by default, is **not verifiable**; the computation party can:

- ◇ Use a different program than the given one
- ◇ Use different data than the given one

The **input party** can also given a *different input* than the promised one.

FHE alone is not enough for all trust settings.

Multi-Party Computation

MPC is also about **input privacy** during computation, but it can tolerate *active adversaries*.

Multi-Party Computation

MPC is also about **input privacy** during computation, but it can tolerate *active adversaries*.

In MPC:

- There **must be multiple computation parties** (at least 2).
- The **input parties** can be part of them or separate parties.
- They *secret-share* their data *between the computation parties*:
  - Each party holds a random-looking share of the data: no information is revealed.
  - Sharing pattern can be parametrised at will, with trade-offs between trust, security and speed.
  - **Output party** can be anyone, different data can be output to different parties.

Multi-Party Computation: Challenges

◇ Secret-shared data **requires communication** for **every** AND gate computation.

- Possible to optimise computation to some extent.
- On a good network, faster than FHE, even across continents. Large bandwidth requirements.
- Local computation is not very heavy.

Multi-Party Computation: Challenges

- Secret-shared data **requires communication** for **every** AND gate computation.
  - Possible to optimise computation to some extent.
  - On a good network, faster than FHE, even across continents. Large bandwidth requirements.
  - Local computation is not very heavy.
- Programming is not easy:
  - Control flow needs to preserve privacy. (How to stop loops? How to hide branching choices?)
  - Floating point numbers are hard to represent with arithmetic data and operations.
  - (Libraries exist to help.)

Multi-Party Computation: Trust Setting

Some argue this is the hardest part of deploying MPC:

Need **multiple distrusting** parties with **a common goal.**

Multi-Party Computation: Trust Setting

Some argue this is the hardest part of deploying MPC:

Need **multiple distrusting** parties with **a common goal.**

- ◇ Often ill suited to business applications. Requires creative application, or a specific solution.

Multi-Party Computation: Trust Setting

Some argue this is the hardest part of deploying MPC:

Need **multiple distrusting** parties with **a common goal.**

- Often ill suited to business applications. Requires creative application, or a specific solution.
- Highly configurable:
  - Input parties can be separate from computation and/or output parties.
  - Computation parties keep check on each other: **verifiability by default** (from within).
  - Input parties need trust **only one** computation party (depending on protocol).
  - Secret-sharing can be configured according to existing trust links between parties.
  - Output parties can be multiple and receive different data (e.g. regulator vs. customer).

Cryptographic Proofs

Provide **unforgeable proof** of computation or condition satisfiability.

◇ **Soundness** guarantees the proving party cannot cheat.

Optionally:

◇ **Zero-knowledge** gives confidentiality to the prover’s data.
◇ **Succinctness** guarantees a small proof and fast verification.

Until recently, these systems served specific purposes:

◇ digital signatures (used in credit card PIN system, or in ECDSA signatures e.g. for TLS),
◇ specific applications in MPC protocols to guarantee honest behaviour (adversaries who can’t forge proofs have to behave honestly).

Cryptographic Proofs: Cutting-Edge Advances

Recently, the concept of **verifiable VM** has been made possible.

1. Given a *program* and some *inputs*, the VM is executed to generate an *execution trace*.
2. With this *trace*, a **proof of execution** is created.
3. A third party can **verify** the proof to check that the **output** of the program is correct.

This technology is **general purpose**: any program can be proven.

Cryptographic Proofs: Cutting-Edge Advances

Recently, the concept of **verifiable VM** has been made possible.

1. Given a *program* and some *inputs*, the VM is executed to generate an *execution trace*.
2. With this *trace*, a **proof of execution** is created.
3. A third party can **verify** the proof to check that the **output** of the program is correct.

This technology is **general purpose**: any program can be proven.

Zero-knowledge lets the prover hide some data, e.g. the inputs, or even the program.

Cryptographic Proofs: Cutting-Edge Advances

Recently, the concept of **verifiable VM** has been made possible.

1. Given a *program* and some *inputs*, the VM is executed to generate an *execution trace*.
2. With this *trace*, a **proof of execution** is created.
3. A third party can **verify** the proof to check that the **output** of the program is correct.

This technology is **general purpose**: any program can be proven.

Zero-knowledge lets the prover hide some data, e.g. the inputs, or even the program.

Succinctness means the system creates a proof that is *small* and *fast to verify*. It enables large system updates to be proven in a small amount of verification work and stored in data-efficient structures.

Cryptographic Proofs: Cutting-Edge Advances

Linear-time proof systems (not succinct) enable *real-time proofs of execution*.

This enables applications such as “proof of monitoring”. For example:

- ◇ sensors deployed in untrusted environments can prove that the data was collected and processed according to the hardware and software, and that it wasn’t tampered with.
- ◇ sensors that infringe on privacy can be equipped with *zero-knowledge* proofs of computation to prove correct data processing but without revealing the original data.
  - ▶ Traffic cameras could prove license-plate recognition without revealing the photograph.
  - ▶ Ankle bracelets could prove GPS localisation (to check the wearer is within an allowed area) without continuously reporting the exact location.

Verifiable FHE

A candidate for the "holy grail" of COED computation:

verifiable FHE proves computation performed on FHE-encrypted ciphertexts.

1. Data remains **confidential** *during computation*.
2. Receiver can **verify** the correctness of the computation *with less effort*.

Research in this area is ongoing, with constructions often combining FHE with succinct proof systems (such as SNARKs) to provide both privacy and verifiability.

However, these systems are still in their early stages and tend to be even more computation-ally expensive than FHE alone.

Differential Privacy

FHE and MPC provide ideal private computation:

no information about the input can be deduced from witnessing the computation take place.

However, there are **no guarantees** about the confidentiality of the input if the output is *also known*.

For this another technology is required: differential privacy.

◇ carefully calibrated *random noise* is added to the input or outputs of computation
◇ this hides individual data points while preserving output correctness.

An average computed with FHE or MPC still reveals information about the data points. Additional differential privacy can mitigate that.

Applications

Examples

- ◇ Peer-to-peer energy bill computation with FHE
- ◇ Private-key decentralisation with MPC
- ◇ Blockchain decentralised computer scaling with ZKPs

Peer-to-peer Energy Billing[1]

- ◇ Community-sized distribution network: households sell to their neighbours
- ◇ Distribution company is required to act as an intermediary.
- ◇ For accurate negotiation and billing, the following information needs to be processed:
  1. The generation capacity of each household *(sensitive)*
  2. The real-time consumption pattern of each household *(sensitive)*
  3. The price of energy *(public)*

[1]https://eprint.iacr.org/2024/1562

Peer-to-peer Energy Billing[1]

This work implements a billing algorithm using FHE:

1. the households encrypt their production/consumption predictions and usages
2. the network company computes an encrypted bill for each household
3. Each household can then decrypt their bill, knowing that their specific information remained confidential during the computation.

[1]https://eprint.iacr.org/2024/1562

Peer-to-peer Energy Billing[1]

However, the network company might not trust households to declare their bills honestly: they might forge their bill to show a smaller cost than the real one.

This is where **zero-knowledge proofs** can *replace the trust* requirement.

If the company requests a proof that the bill declared by the household is the one decrypted from the ciphertext that it received, forging the bill becomes as hard as forging the cryptographic proof.

[1]https://eprint.iacr.org/2024/1562

Private-Key Decentralisation

The security of central private keys for banks or other big financial institutions is critical. Keeping them stored in a single location creates a costly single point of failure.

Private-Key Decentralisation

The security of central private keys for banks or other big financial institutions is critical. Keeping them stored in a single location creates a costly single point of failure.

Unbound (since acquired by Coinbase) commercialised a solution based on MPC:

◇ the private signing key was secret-shared between different servers

◇ and an MPC protocol was executed each time a signature needed to be generated.

Private-Key Decentralisation

The security of central private keys for banks or other big financial institutions is critical. Keeping them stored in a single location creates a costly single point of failure.

Unbound (since acquired by Coinbase) commercialised a solution based on MPC:

- ◇ the private signing key was secret-shared between different servers
- ◇ and an MPC protocol was executed each time a signature needed to be generated.

Using several shares decentralises the risk of secret-key compromise for the bank =since an attacker would need to compromise several systems simultaneously.

Deploying this system within a single institution lowers the requirements on the security of the MPC protocol. A passively secure is sufficient since the company trusts its own systems.

Nokia's Blind Audio Mixing: End-to-End Encrypted Audio Calls [RWC 2025]

Peer-to-peer audio signal communication allows for end-to-end encryptions: only need p2p channels; but network complexity is $O(n^2)$.

Nokia's Blind Audio Mixing: End-to-End Encrypted Audio Calls [RWC 2025]

Peer-to-peer audio signal communication allows for end-to-end encryptions: only need p2p channels; but network complexity is $O(n^2)$.

To have $O(n)$ communication, central server can mix the audio signal before delivering it, but end-to-end encryption is no longer possible:

- ◇ the server has to decrypt the audio streams from all the participants.

Nokia's Blind Audio Mixing: End-to-End Encrypted Audio Calls [RWC 2025]

Nokia took another look at existing solutions based on homomorphic encryption: the idea is to perform **signal addition** on encrypted data.

Nokia's Blind Audio Mixing: End-to-End Encrypted Audio Calls [RWC 2025]

Nokia took another look at existing solutions based on homomorphic encryption: the idea is to perform **signal addition** on encrypted data.

1. Encrypting the plaintext signal directly is too costly because of the ciphertext expansion.

Nokia's Blind Audio Mixing: End-to-End Encrypted Audio Calls [RWC 2025]

Nokia took another look at existing solutions based on homomorphic encryption: the idea is to perform **signal addition** on encrypted data.

1. Encrypting the plaintext signal directly is too costly because of the ciphertext expansion.
2. Compressing before encryption lowers bandwidth, but signal addition results in garbage.

Nokia's Blind Audio Mixing: End-to-End Encrypted Audio Calls [RWC 2025]

Nokia took another look at existing solutions based on homomorphic encryption: the idea is to perform **signal addition** on encrypted data.

1. Encrypting the plaintext signal directly is too costly because of the ciphertext expansion.
2. Compressing before encryption lowers bandwidth, but signal addition results in garbage.

   (adding compressed files together doesn't add the uncompressed signals).
3. Evaluating the decompression circuit in FHE is too slow for the requirements of real-time audio.

Nokia’s Blind Audio Mixing: End-to-End Encrypted Audio Calls [RWC 2025]

Nokia took another look at existing solutions based on homomorphic encryption: the idea is to perform **signal addition** on encrypted data.

1. Encrypting the plaintext signal directly is too costly because of the ciphertext expansion.
2. Compressing before encryption lowers bandwidth, but signal addition results in garbage.

   (adding compressed files together doesn’t add the uncompressed signals).
3. Evaluating the decompression circuit in FHE is too slow for the requirements of real-time audio.
4. *Neural audio compression* creates embeddings with additive homomorphism.

   These embeddings are also customisable in real time to adjust to bandwidth requirements.

Apple Deploys Homomorphic Encryption at Scale [RWC 2025]

On-device features can be enriched with data from servers.

Need to consider which data need to leave the device, and how can its privacy be enhanced. Client queries have to remain private even from Apple.

1. **Real-time private caller ID**

   Identifies business or spam phone numbers from a register on the server without the device sharing the number.

Apple Deploys Homomorphic Encryption at Scale [RWC 2025]

2. **Enhancing visual search by tagging pictures on device**

◇ On-device model detects region of interest on the photo with a potential landmark
◇ A second model converts the region of interest into an embedding which retains semantic meaning about the input.

This means that the distance between embeddings is proportional to the similarity of the images.

Apple Deploys Homomorphic Encryption at Scale [RWC 2025]

Several options are then possible to compute the best tag based on the scores:

1. The database of embeddings can be downloaded to the device.
2. The device can send the embedding, but it can be reverse-engineered.
3. Use HE directly with on-device private key and compute everything in ciphertext space 17 MB per query -> too expensive.
4. Solution: use differential privacy to access a subset, helps with server computation. Also use efficient HE to reduce communication.
    1. the user reports the cluster that the embedding belongs to
    2. hide the client with a batch of queries.

Apple Deploys Homomorphic Encryption at Scale [RWC 2025]

Several options are then possible to compute the best tag based on the scores:

1. The database of embeddings can be downloaded to the device.
2. The device can send the embedding, but it can be reverse-engineered.
3. Use HE directly with on-device private key and compute everything in ciphertext space 17 MB per query -> too expensive.
4. Solution: use differential privacy to access a subset, helps with server computation. Also use efficient HE to reduce communication.
   1. the user reports the cluster that the embedding belongs to
   2. hide the client with a batch of queries.

3. **Categorisation of business email addresses**.

Google ID Passes and Zero-Knowledge Proofs

On Apr 29, 2025, Google announced ID Passes coming to Google Wallet in the U.K.

- The User can store an identity document in their Google Wallet
- The Wallet can provide certain information to Apps and Services
  - At launch, this will work for U.K. Railcards (age restricted).

However, sharing the credentials with a third-party is overreach, it reveals more information (the age) than required (that the User is over 18, or under 30), or allows the Service to link the age and the identity.

- Google has developed a ZKP solution to limit this.
- Only a **proof of age** is shared with the Service (if I understood correctly)

Enabling blockchain scalability

Blockchains enabled the development of cryptocurrencies in the late 2000s with the appearance of Bitcoin, a digital ledger aiming to replace banks as the records of ownership.

In the mid-2010s, a *decentralised computer* was created: **the Ethereum network**.

- ◇ Transactions can also call functions from deployed programs (smart contracts)
- ◇ They are executed in parallel by all the validators of the network.

The replicated nature of the system creates **trust**.

- ◇ A majority of validators will reach the same result and agree on the updated state of the computer.
- ◇ This creates high costs and is an obstacle to further scaling.

Enabling blockchain scalability

Many projects have turned to **succinct proofs** instead

- ◇ A parallel network is created, which also runs the same “operating system”
- ◇ Updates are proven by a single operator
- ◇ The “proof of state update” is then posted on the “central” computer that is the Ethereum network and verified by a smart contract.
- ◇ **Succinctness**: much less expensive work for the replicated computer to perform.
- ◇ **Soundness**: the operator’s actions are “mathematically” secure (rather than economic).