State of the art of Lattice-Based Threshold Cryptography

Gabrielle Beck, 05/12/2025

Postdoctorant au LIRMM avec Fabien et Katharina

What is fully homomorphic encryption (FHE)?

Say you want to compute a function *f* on some inputs but…

- f is a very large function (in depth or width)
- you don't have much (relative) computation power
- your function input is sensitive (or you need access to someone else's private data to do the calculation)

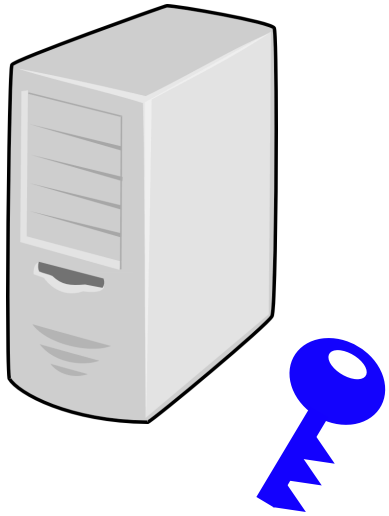
What is fully homomorphic encryption (FHE)?

What types of functions can this capture?

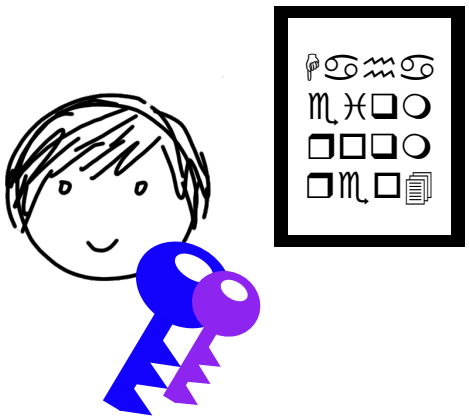
- Intensive statistical calculations on databases w. PII
- Computations on medical data, genomic sequencing
- Other cryptographic primitives (e.g. private information retrieval)

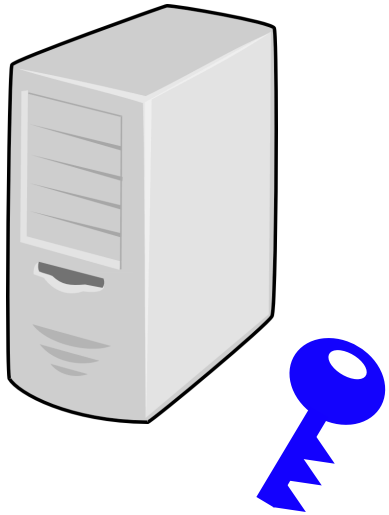
What is FHE?

A public/private key pair

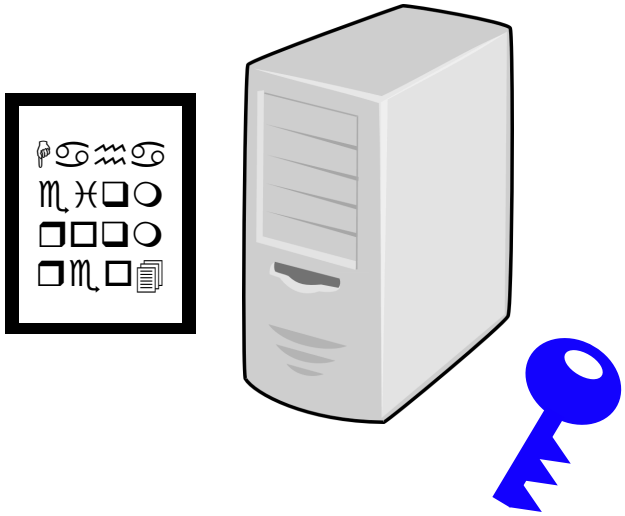


What is FHE?

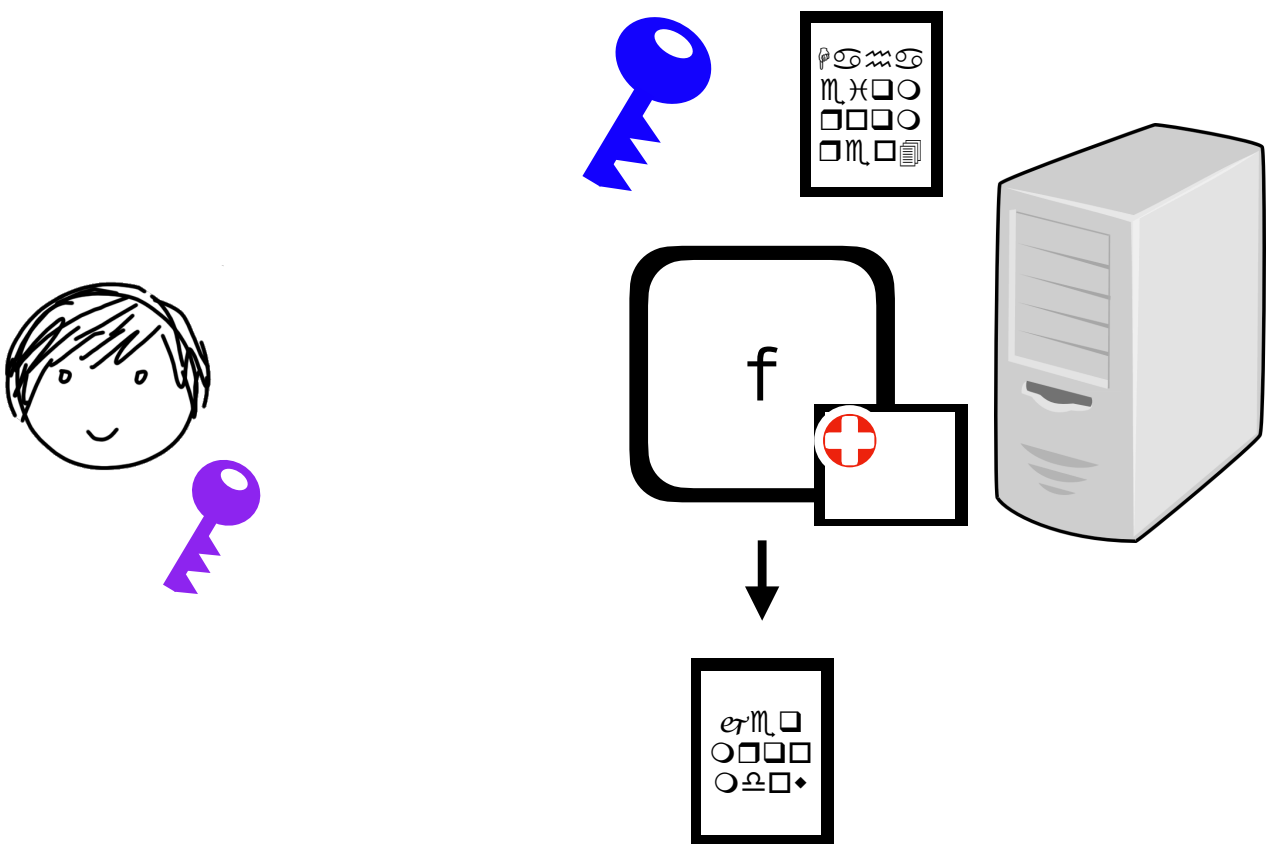




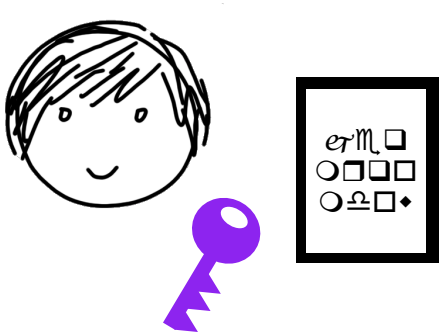
What is FHE?



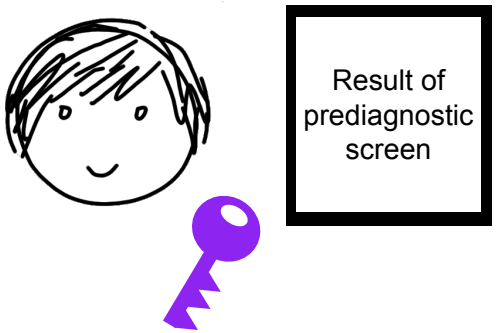
What is FHE?



What is FHE?

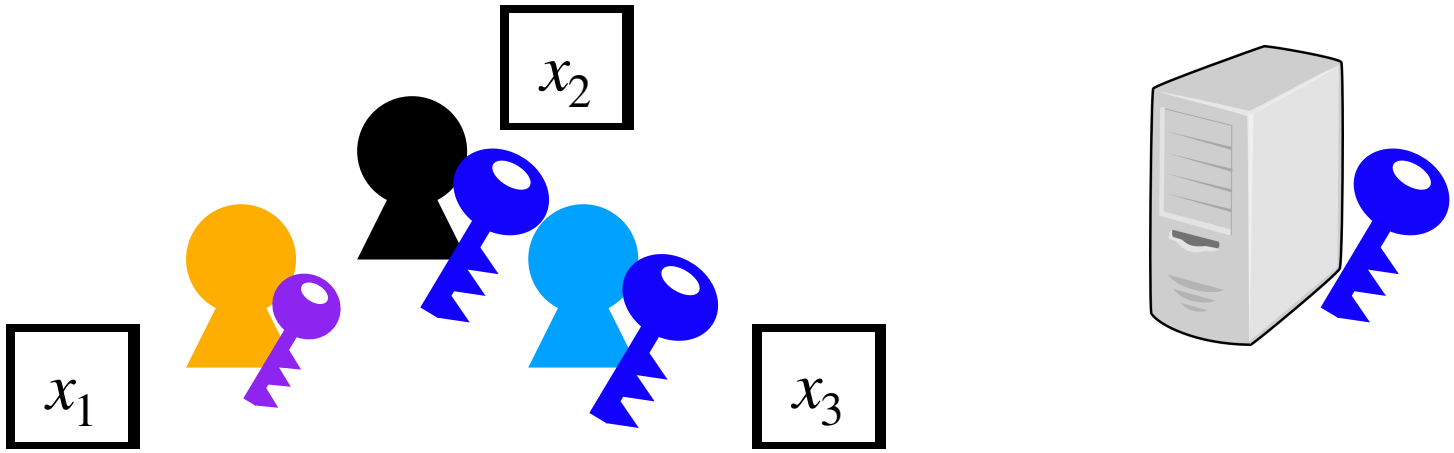


What is FHE?



What if we want different trust assumptions?

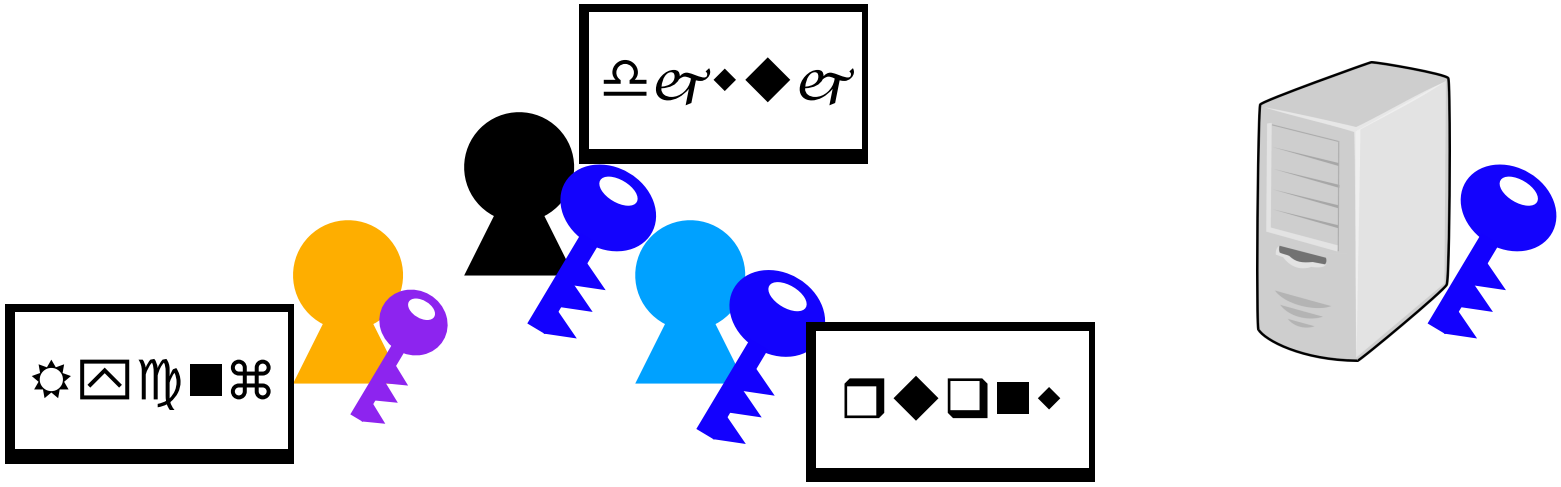
Suppose we have more than one client that wants to participate in the computation… can we do the same thing as before?



Goal: Compute $f(x_1, x_2, x_3)$

What if we want different trust assumptions?

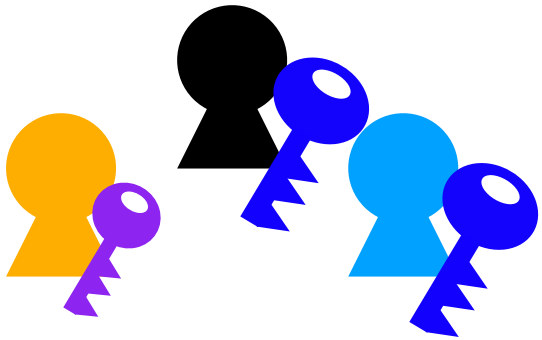
Suppose we have more than one client that wants to participate in the computation… can we do the same thing as before?

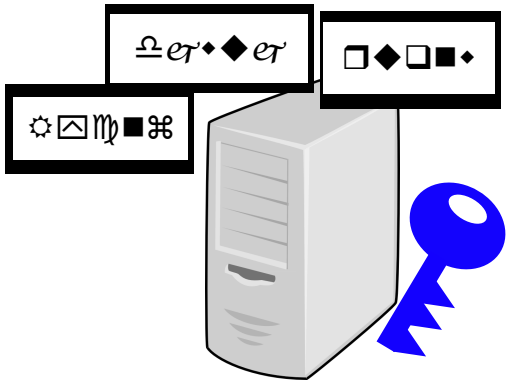


Goal: Compute $f(x_1, x_2, x_3)$

What if we want different trust assumptions?

Suppose we have more than one client that wants to participate in the computation… can we do the same thing as before?

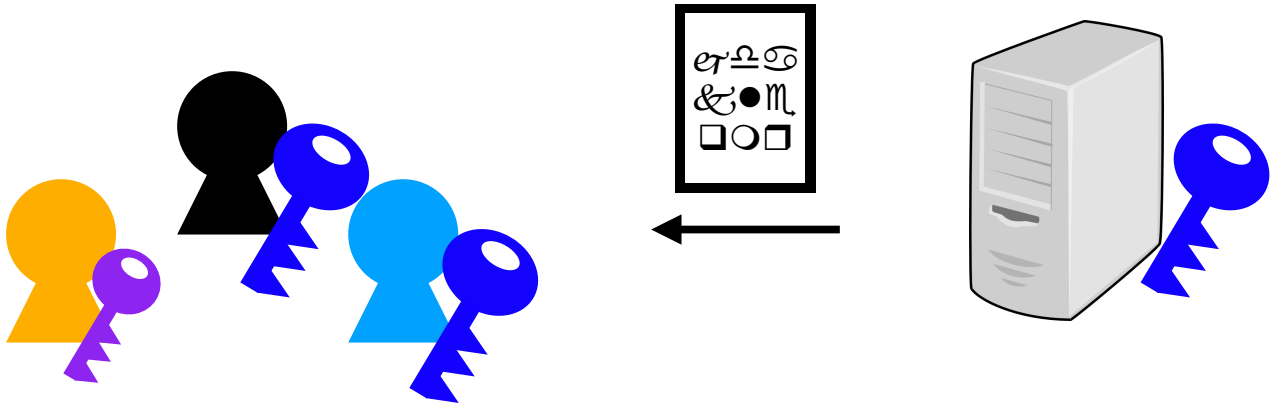




Goal: Compute $f(x_1, x_2, x_3)$

What if we want different trust assumptions?

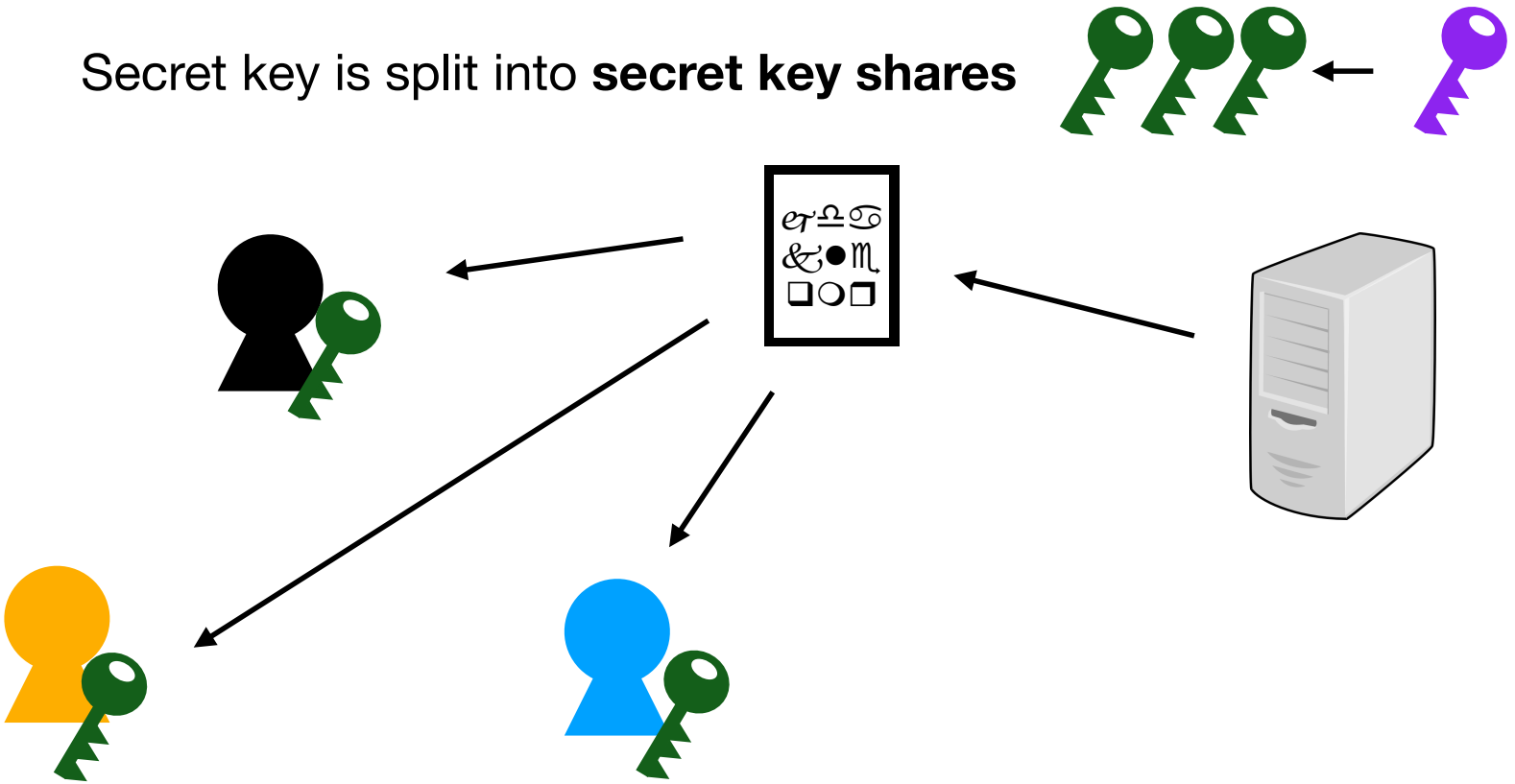
Suppose we have more than one client that wants to participate in the computation… can we do the same thing as before?



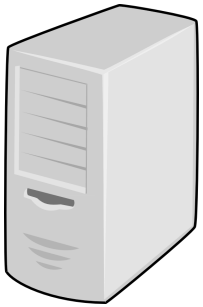
Problem: We don’t want to trust just one client!

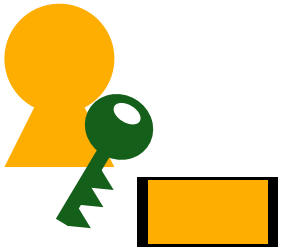
Break up the secret key! (Use Threshold Cryptography)

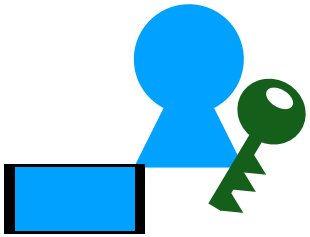
Secret key is split into **secret key shares**



Break up the secret key! (Use Threshold Cryptography)

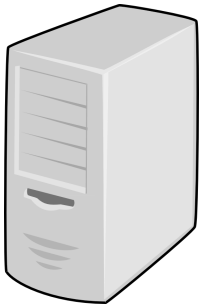






Break up the secret key! (Use Threshold Cryptography)

$f(x_1, \ldots, x_n)$



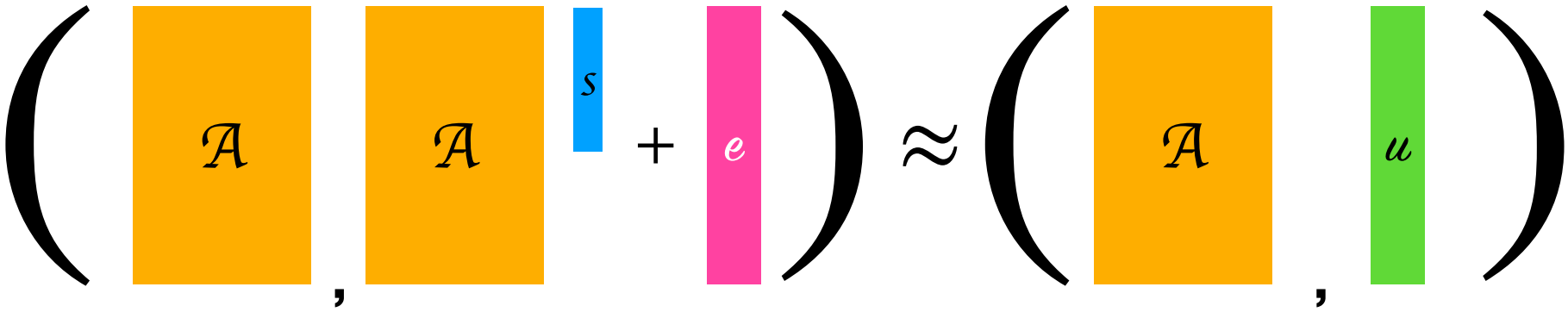
All parties combine decryption shares to get a result!

Lattice-based cryptography

- Threshold FHE schemes in cryptography are built off of lattice-based assumptions
  - Most constructions from unstructured lattices in some shape or form rely on the learning with errors problem (LWE)

Learning with Errors (high level)

- Distinguish “noisy samples” from a matrix multiplication with uniformly random values



Structure of decryption for LWE-based constructions

- Decryption for these schemes is usually simple
  - Ciphertext is a vector and an element $(\vec{c}_1, c_2) \in \mathbb{Z}_q^m \times \mathbb{Z}_q$
  - We decrypt using a secret key $\vec{s} \in \mathbb{Z}_q^m$
    - Compute $c_2 - \langle \vec{s}, \vec{c}_1 \rangle \approx m + e$
    - Round off the error e

Structure of decryption for LWE-based constructions

- Decryption for these schemes is usually simple
  - Ciphertext is a vector and an element $(\vec{c}_1, c_2) \in \mathbb{Z}_q^m \times \mathbb{Z}_q$
  - We decrypt using a secret key $\vec{s} \in \mathbb{Z}_q^m$
    - Compute $c_2 - \langle \vec{s}, \vec{c}_1 \rangle \approx m + e$
    - Round off the error e

Important Note! e depends on $\vec{s}$ (normally fine for single decryptor setting)

A template for Threshold FHE from lattices

- Linear secret sharing (LSS) can allow us to secret share a vector $\vec{s} \in \mathbb{Z}_q$ into multiple shares $\vec{s}_1, \ldots, \vec{s}_n$
  - Given a large enough set $T \subseteq [n]$, and shares $\vec{s}_1, \ldots, \vec{s}_T$, there exists coefficients $w_1, \ldots, w_T$ such that you can recover the secret vector $\vec{s} = \sum_{i=1}^{T} w_i \vec{s}_i$

A template for Threshold FHE from lattices

- Split the secret key for the encryption scheme into shares
- During decryption do the first part of “normal decryption” on the ciphertext
- Works because reconstruction is linear

$$\{\overrightarrow{sk}_i\}_{i\in[n]} \leftarrow \mathsf{Share}(\overrightarrow{sk}, n, t)$$

Partial decryption on $(\vec{c}_1, c_2)$ can be computed as: $c'_i = \langle \vec{c}_1, \overrightarrow{sk}_i \rangle + e_i$

Shares can be combined to get output as $res = c_2 - \sum_i w_i c'_i$

A template for Threshold FHE from lattices

$$
\begin{aligned}
res &= c_2 - \sum_i w_i c_i' = c_2 - \sum_i w_i \langle \vec{c}_1, \overrightarrow{sk}_i \rangle + e_i \\
&= c_2 - \langle \vec{c}_1, \sum_i w_i \overrightarrow{sk}_i \rangle + \sum_i w_i e_i \\
&= c_2 - \langle \vec{c}_1, \overrightarrow{sk} \rangle + \sum_i w_i e_i \\
&= m + e + \sum_i w_i e_i
\end{aligned}
$$

Instantiating the template

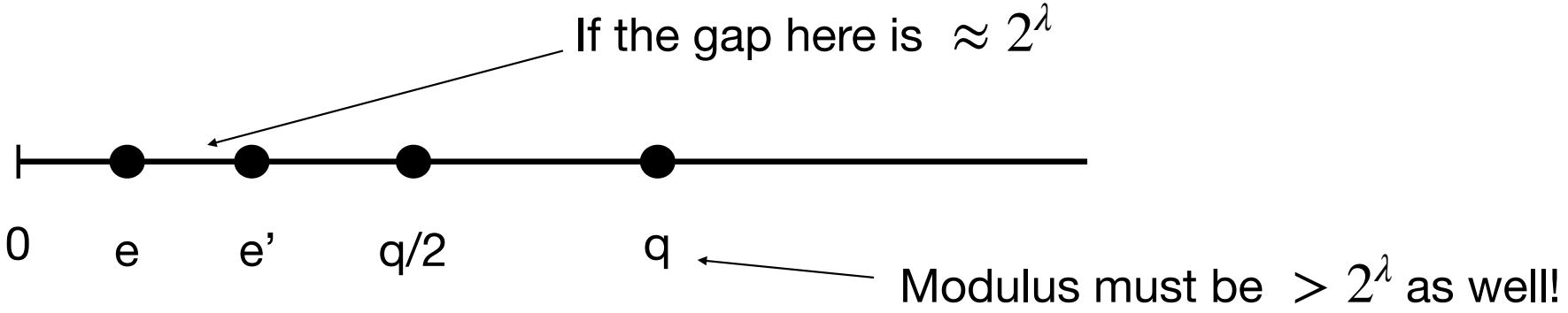
For security, e_i values must completely hide the distribution of secret key dependent error

For correctness, $\sum_i w_i e_i$ should be sufficiently small**

**this depends on $|w_i|$

Instantiating the template

- What distribution do we use for e_i?
  - First proposed technique was statistical noise flooding (think: so much extra noise you drown out signal of the other value entirely)
  - But this causes problems!



Instantiating the template

- What do we use for the LSS?
  - use Shamir secret sharing, but share size is $O(N\log(N))$ per party,
    - Ciphertext and decryption shares grow with N
  - use “folklore” construction with $w_{i,j} \in \{0,1\}$ but key size per party is $O(N^4)$
    - So-called {0,1} LSS

Can we do better?

- Use a different LSS
  - There are better {0,1} LSS schemes which imply smaller key sizes/more efficient constructions [ANP23]
- Use smaller noise flooding** (must still be large enough/shaped correctly to guarantee security!)
  - Can do this with **constant** noise flooding for Thresh. PKE w. some restrictions
  - Can also do **polynomially-sized** noise flooding using Reyni divergence or new Threshold-LWE assumption
    - Limitation: only secure assuming an a-priori bounded number of decryption queries

Compare and Contrast ThPKE/ThFHE (what is known)

Schemes	Noise Flooding	Thresholds Supported	Functionality	Other notes
Universal Thresholdizer [BGG+17]	Statistical	All	FHE	
BS23	Polynomial	All	ThPKE	Reyni, poly-bounded number of queries, weaker security notion
MS23	Constant	N-out-of-n	ThPKE	Known norm assumption, no known randomness
Pilvi (CLW25)	Polynomial	All	ThPKE	Poly-bounded number of queries

Compare and Contrast ThPKE/ThFHE (what is known)

Schemes	Noise Flooding	Thresholds Supported	Functionality	Other notes
Universal Thresholdizer [BGG+17]	Statistical	All	FHE	
BS23	Polynomial	All	ThPKE	Reyni, poly-bounded number of queries, weaker security notion
MS23	Constant	N-out-of-n	ThPKE	Known norm assumption, no known randomness
Pilvi (CLW25)	Polynomial	All	ThPKE	Poly-bounded number of queries

References

- [PS25] - Low Communication Threshold Fully Homomorphic Encryption. Passelègue and Stehlé. https://eprint.iacr.org/2024/1984
- [MS23] - Simulation-Secure Threshold PKE from LWE with Polynomial Modulus. Micciancio and Suhl. https://eprint.iacr.org/2023/1728
- [BS23] - Simple Threshold (Fully Homomorphic) Encryption From LWE With Polynomial Modulus. Boudgoust and Scholl. https://eprint.iacr.org/2023/016
- [CLW] Pilvi: Lattice Threshold PKE with Small Decryption Shares and Improved Security. Cini, Lai, and Woo. https://eprint.iacr.org/2025/1691
- [ANP23] - How to Recover a Secret with O(n) Additions. Applebaum, Nir Pinkas. https://eprint.iacr.org/2023/838
- [BGG+17] Threshold Cryptosystems FromThreshold Fully Homomorphic Encryption. Boneh et al. https://eprint.iacr.org/2017/956