Overview of Fully Homomorphic Encryption:
functionality and security models

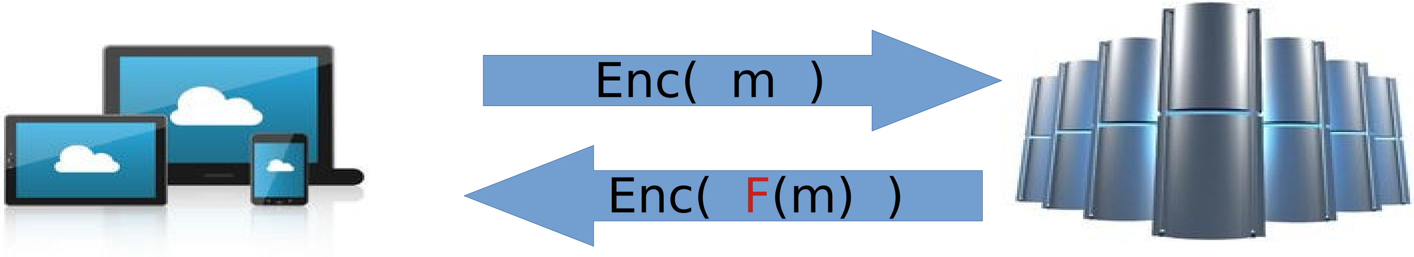
Daniele Micciancio
(UC San Diego)

Presented at NIST WPEC 2024 on September 25

Fully Homomorphic Encryption

- Encryption: used to protect data at rest or in transit

- Fully Homomorphic Encryption: supports arbitrary computations (F) on encrypted data



FHE Timeline

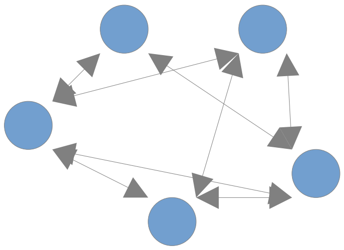
- 1978 – Rivest, Adleman, Dertouzos:
  - pose problem
- 2009 – Gentry:
  - **first** candidate solution
  - bootstrapping technique
- 2011 – Brakerski, Vaikuntanathan:
  - **first** solution based on standard lattice problems
- [BGV12,GHS12,GSW13,AP13/14,DM15,CGGI17,CKKS18,…, 2024]
  - new schemes, major efficiency improvements
  - Implementations: [SEAL, HElib, PALISADE, OpenFHE, HEAAN, Lol, FHEW, TFHE, LattiGo, …]
  - all based on lattices and use bootstrapping technique

This talk

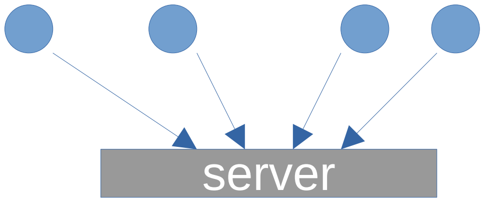
- Question: is FHE a good fit for a given application?
- Functionality
  - exact vs approximate computations
  - composability properties
- Security properties
  - passive vs active attacks
  - impact of decryption failures
- Advanced properties:
  - Verifiability, distributed decryption, etc.

FHE vs MPC

- Same problem: secure computation
- MPC (secure Multi Party Computation)
  - Data is “secret shared” among partecipants
  - Secure computation is done interactively



- FHE (Fully Homomorphic Encryption)
  - Data is protected using encryption scheme
  - Computation on encrypted data does not require interaction
  - Decryption key may be “secret shared”

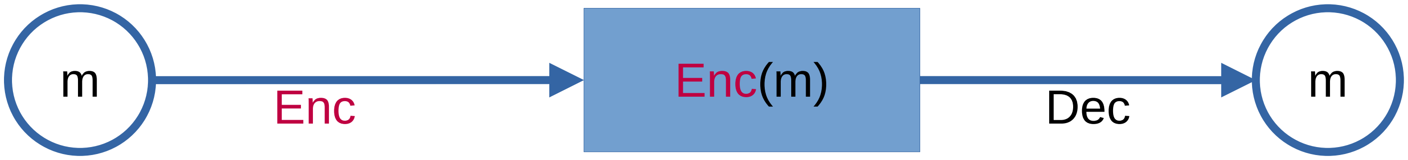


Use cases for FHE

- Public Key FHE scheme
- Workflow:
  - Multiple parties encrypt their data locally, under the same public key
  - Encrypted data is collected in encrypted form
  - Computation is performed on encrypted data
  - Final result is decrypted and shared with participants
- Examples:
  - Hospitals sharing patient data for join medical study
  - Similarly for financial, or other sensitive data

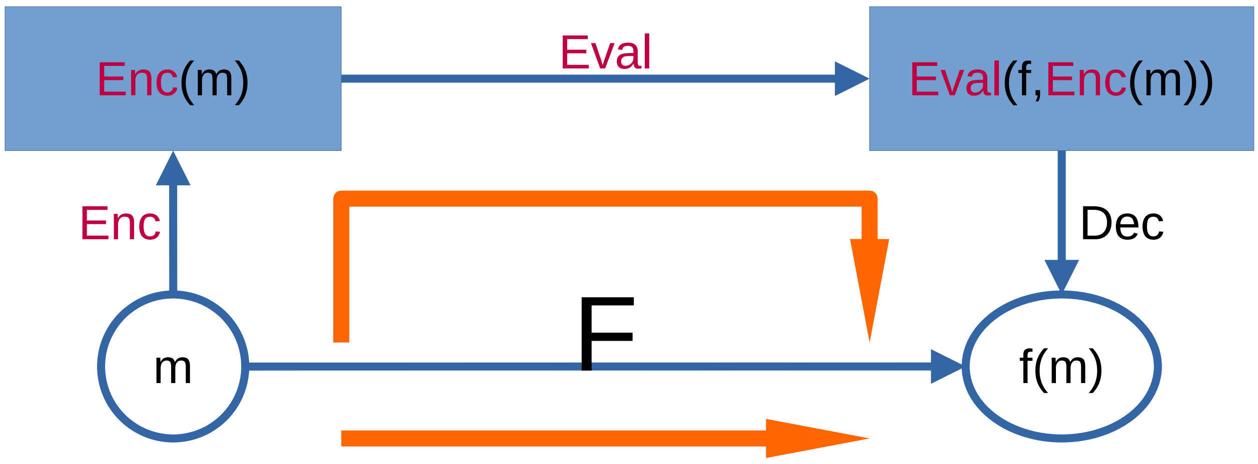
Encryption Scheme

- Syntax: (Gen,Enc,Dec)
- Correctness:
  - (pk,sk) $\leftarrow$ Gen
  - $Dec_{sk}(Enc_{pk}(m)) = m$



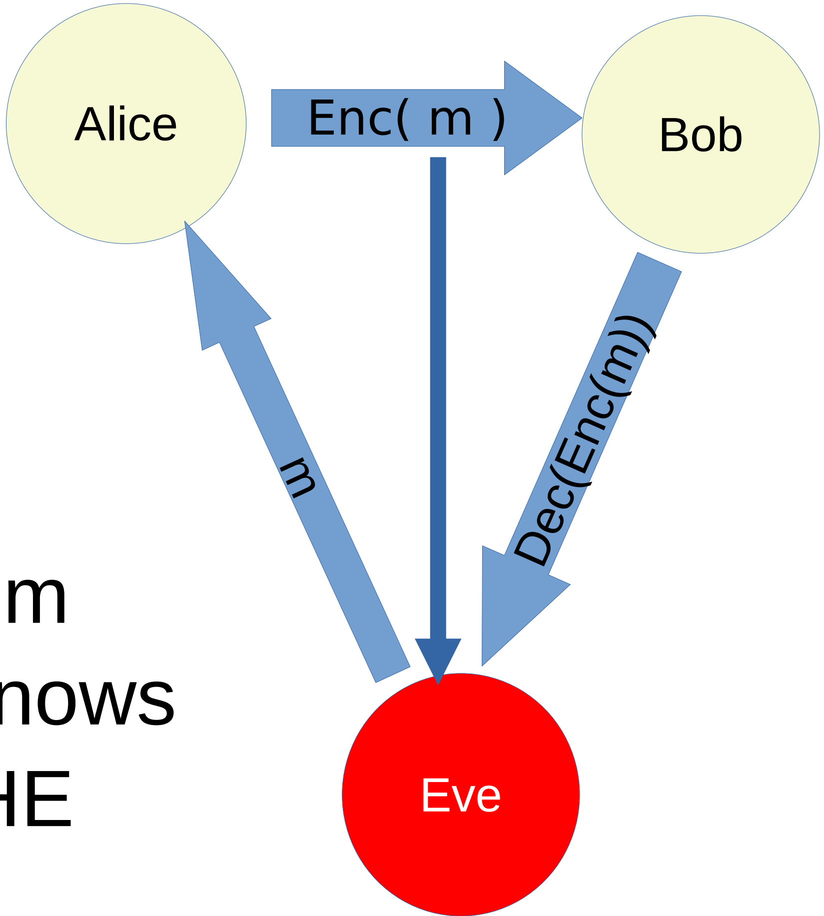
Fully Homomorphic Encryption

- FHE Scheme: (Gen,Enc,Dec,Eval)
  - (pk,sk) ← Gen
  - $Dec_{sk}(Eval_{pk}(F, Enc_{pk}(m)) = F(m)$



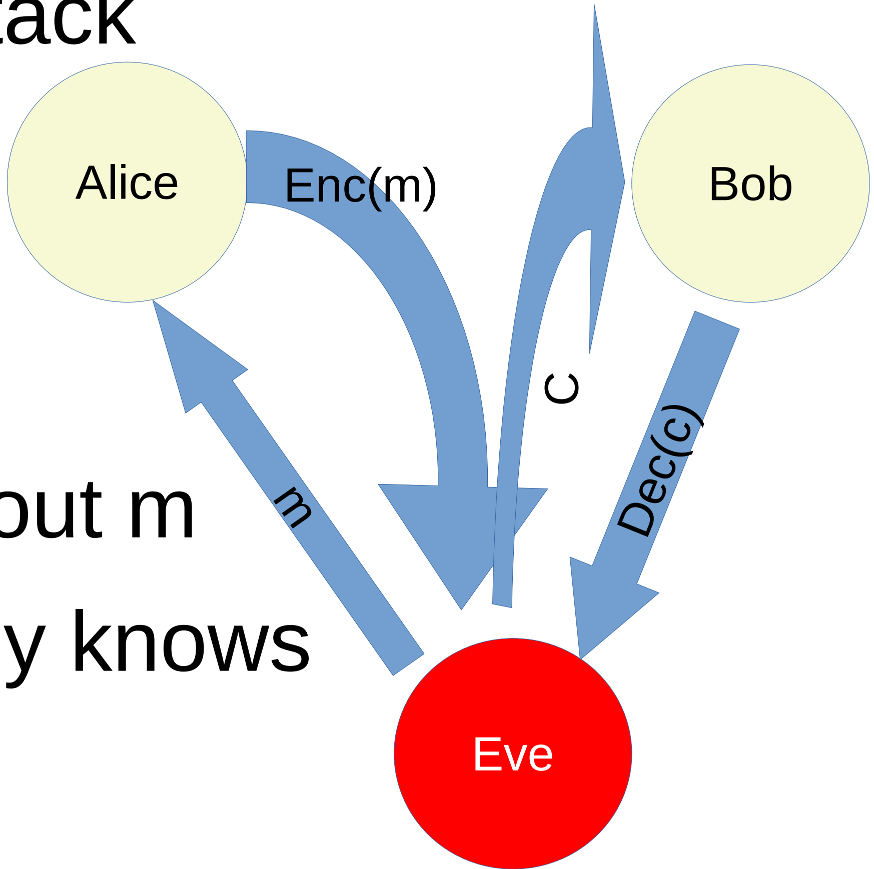
Passive attack model

- CPA: Chosen Plaintest Attack
- Adversary (eve) can:
  - Choose/influence message m
  - See the encryption Enc(m)
  - See result of decryption Dec(Enc(m))=m
- Still, cannot tell anything about m other than what she already knows
- Security definition applies to FHE



Active attack model

- CCA: Chosen Ciphertext Attack
- Adversary (eve) can:
  - See Enc(m) of any m
  - See Dec(c) of any c
- Still, cannot tell anything about m
  other than what she already knows



CPA/CCA security in Practice

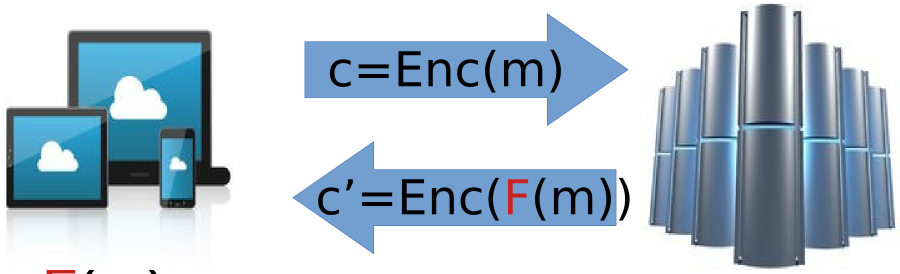
- Remarks
  - Most applications require Active security
  - Active security implies Passive security
  - Active security can be achieved at reasonable cost (e.g., Fujisaki-Okamoto transform)
  - Standards (NIST, etc.) require Active security
  - All this is for regular (non-homomorphic) encryption
- What about Homomorphic Encryption?

CCA security vs Non-Malleability

- CCA (active) security equivalent to non-malleability
  - Given c = Enc(m), adversary cannot compute encryption c’ of related message Dec(c’)=F(m)
  - Intuition: If adversary cannot change c into c’, then active attack reduces to passive attack
- But this is exactly the opposite of FHE:
  - ability to change Enc(m) → Enc(F(m)) is a useful feature!
  - FHE is perfectly malleable, and cannot be CCA secure

Concrete scenario

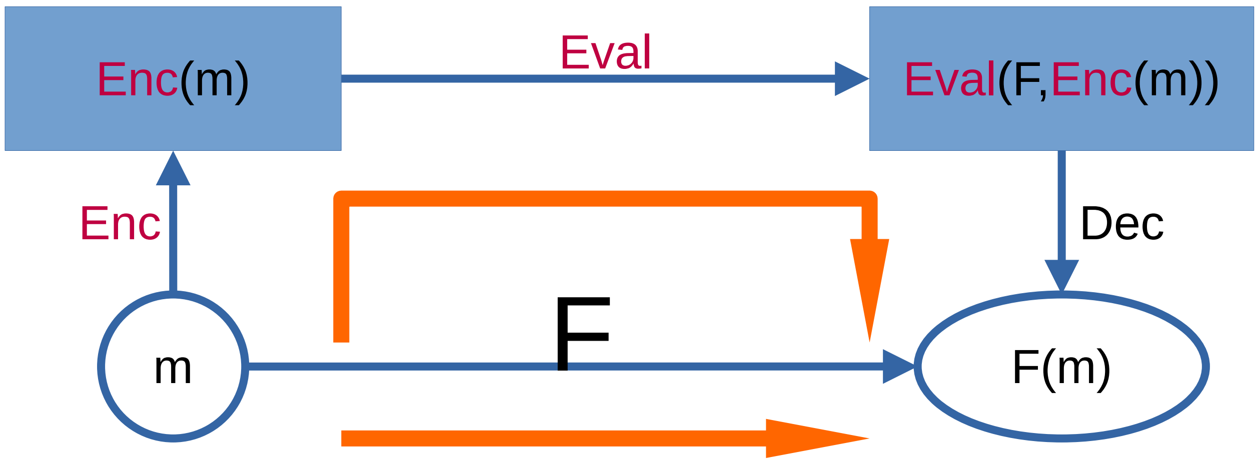
- Application:
  - Store c = Enc(m) on server
  - Server computes c’=Eval(F,c))
  - User decrypts final result Dec(c’) = F(m)



- Questions:
  - How do you know F was applied on correct c ?
  - How do you know the server evaluated the correct F?
- Problem: verifiable FHE
  - Can be addressed using zero-knowledge proofs, etc.
  - Active research area, but not as mature as basic FHE
- Rest of this talk: focus on passive security

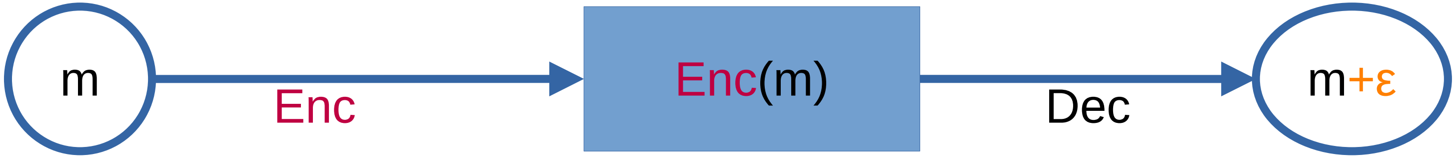
Fully Homomorphic Encryption

- FHE Scheme: (Gen,Enc,Dec,Eval)
  - (pk,sk) ← Gen
  - $Dec_{sk}(Eval_{pk}(F, Enc_{pk}(m)) = F(m)$

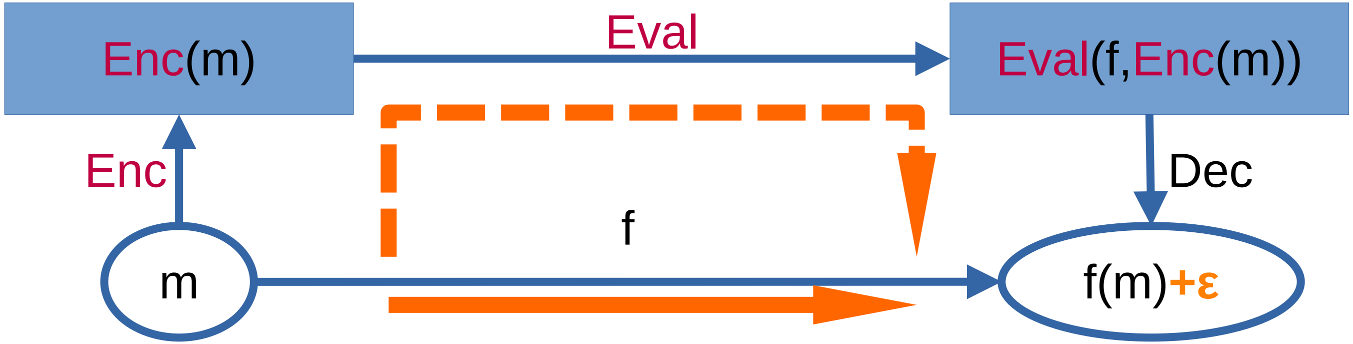


Approximate Encryption Scheme

- $Dec_{sk}(Enc_{pk}(m)) = m + \varepsilon$



- $Dec_{sk}(Eval_{pk}(f, Enc_{pk}(m))) = f(m) + \varepsilon$



Why approximate FHE?

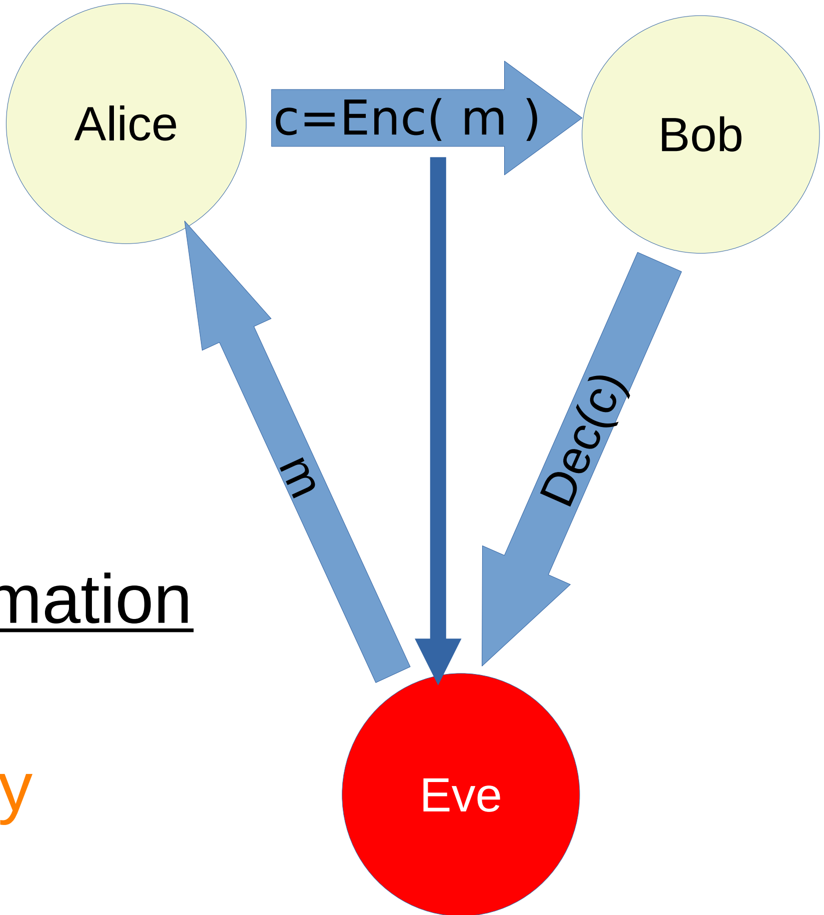
- Lattice cryptography (underlying FHE) is noisy:
  - In its most basic form Dec’(Enc’(m)) = m + ε
  - Solution: use an error correcting code
    - Enc(m) = Enc’(encode(m))
    - Dec(c) = decode(Dec’(c))
    - Dec(Enc(m)) = decode(encode(m)+ε) = m
- In FHE, homomorphic computations increase ε
  - Skipping encode/decode makes FHE much faster
  - In many applications, approximate results are acceptable (e.g., machine learning, statistics, etc.)

Approximate FHE

- [CKKS17]: Homomorphic Encryption for Arithmetics on Approximate Numbers
  - Much more efficient than exact FHE
  - Satisfies standard CPA security definition
- Widely implemented and applied to machine learning, genome analysis, etc.
- [LM21]: CKKS insecure under passive attacks!

Passive attack model

- CPA: Chosen Plaintest Attack
- Adversary (eve) can:
  - Choose/influence message m
  - See the encryption c = Enc(m)
  - See result of decryption Dec(c)
- For exact schemes
  - Dec(c) = m gives <u>no useful information</u>
- For approximate schemes
  - Dec(c) = m+ε may leak secret key



Securing Approximate FHE

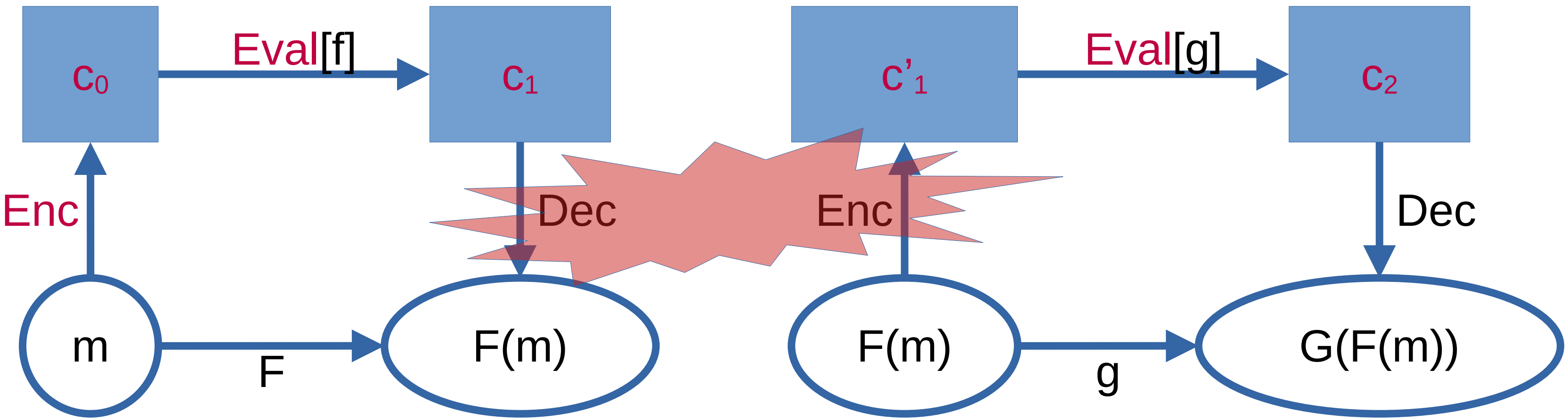
- [LM21]: new CPA-D security definition
  - Equivalent to CPA for exact schemes
  - Captures passive attacks when Dec(c) = m+ε
- [LMSS22]:
  - Add extra noise to decryption Dec(c) = Dec'(c)+ε'
  - Calibrate $\varepsilon' \gg \varepsilon$ to achieve CPA-D security
  - Reasonable cost, still more efficient than exact FHE

Composability

- c_0 = Enc(m)
- c_1 = Eval(F,c_0)
- c_2 = Eval(G,c_1)

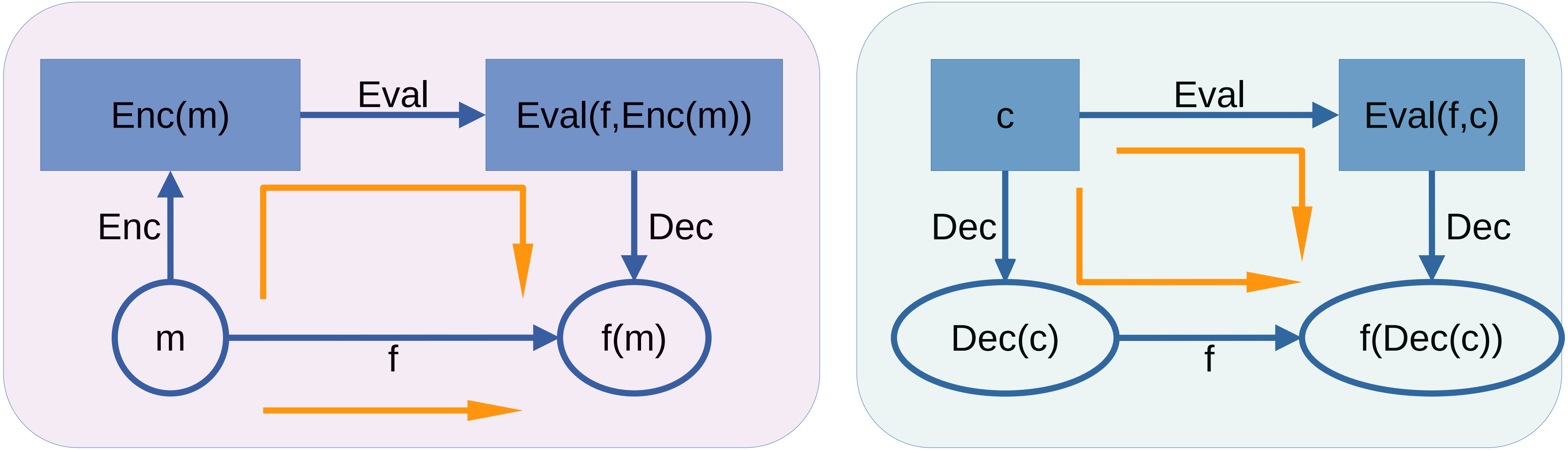
Question: Is Dec(c_2) = g(f(m)) ?

Answer: **not** necessarily



Standard vs Composable FHE

- Standard FHE: Dec(Eval(F,Enc(m))) = F(m)
- Composable FHE: Dec(Eval(F,c)) = F(Dec(c))



FHE Taxonomy

- Gentry: historical, but bootstrapping still relevant
- [BGV/BFV]:
  - Exact, operates on integer vectors (large message space)
  - Slow bootstrapping, but high bandwidth (SIMD)
- [DM/CGGI] (FHEW/TFHE):
  - Exact, fast, composable, single bit operations
  - Active research on SIMD extensions
- [CKKS]:
  - Approximate, operates on real vectors (large message space)
  - Faster than BGV/BFV at cost of approximate results
  - Requires LMSS noise padding to achieve security

Noise estimation / padding

- [LMSS]: securing CKKS requires adding noise
  - $\mathrm{Dec}(c) = \mathrm{Dec}'(c) + \varepsilon'$
- How much noise?
  - Larger ε' gives more security
  - Smaller ε' gives more accurate results
  - $\varepsilon' \gg \varepsilon$: should be larger than c's noise $\mathrm{Dec}'(c)=m+\varepsilon$
- Question: how big is ε?

Estimating ciphertext noise

- Dec(Enc(m)) = m+ε, for small ε chosen by Enc
- Eval(F,Enc(m)) = F(m) + ε, for larger ε, dependent on f
- In (lattice-based) FHE:
  - Parameters (encode/decode) should be set large enough to correct ciphertext ε noise
  - Large ε has negative effect on efficiency
  - Even more so for Approximate FHE, which requires adding extra ε’ ⪢ ε

Application-aware FHE [AAMP24]

- In many applications,
  - function F is fixed, and known in advance
  - E.g., common statistics: mean, average, standard deviation of encrypted data set
- Good trade-off between security and efficiency:
  - Use function F to estimate ciphertext noise ε
  - Generate FHE parameters specific to f,ε
- Warning: if c' = Eval(F',c) is called with different F':
  - Dec(c') may be incorrect
  - Dec(c') may leak information about secret key

Distributed FHE decryption

- FHE: c=Enc(m) → c'=Enc(F(m))
  - both input and output are encrypted
  - Good and bad at the same time
- Secret (decryption) key sk:
  - Needed to recover final result F(m) = Dec_{sk}(c')
  - It also allows to decrypt original input m=Dec_{sk}(c)
  - Single point of failure
- Solution: secret share sk, and decrypt using MPC

Threshold FHE

- FHE with specialized distributed Dec protocol
  - Lattice-based encryption is “key homomorphic”
  - $\mathrm{Dec}'(sk_1+\ldots+sk_n,c) = \mathrm{Dec}'(sk_1,c)+\ldots+\mathrm{Dec}'(sk_n,c)$
- How to share/use secret key sk:
  - Pick random $sk_1+\ldots+sk_n$ such that $sk_1+\ldots+sk_n=sk$
  - Each share holder computes $\mathbf{d_i}=\mathrm{Dec}'(sk_i,c)$
  - Results are combined into $\mathrm{decode}(\mathbf{d_1+\ldots+d_n}) = m$
- Problem: $\mathbf{d_i}$ are noisy and may <u>leak information</u> about sk_i
- Solution, similar to approx. FHE:
  - Add noise $\mathrm{Dec}(sk_i,c) = \mathrm{Dec}'(sk_i,c) + \varepsilon_i$

Concluding Remarks

- Current FHE implementations:
  - promising technology, potentially useful in many critical applications
  - major efficiency gains during the last 15 years
  - reasonably efficient to be used in practice
- FHE is a technical tool, to be used with care
  - Current schemes target passive security
  - Even passive security can already be quite tricky for approximate/threshold schemes
  - Current FHE research is about much more than just efficiency improvements

Some References

- [BGV] https://ia.cr/2011/277
- [DM] https://ia.cr/2014/816
- [CKKS] https://ia.cr/2016/421
- [CGGI] https://ia.cr/2018/421
- [LM] https://ia.cr/2020/1533
- [LMSS] https://ia.cr/2022/816
- [AAMP] https://ia.cr/2024/203