Fully Homomorphic Encryption

Mohsen Toorani

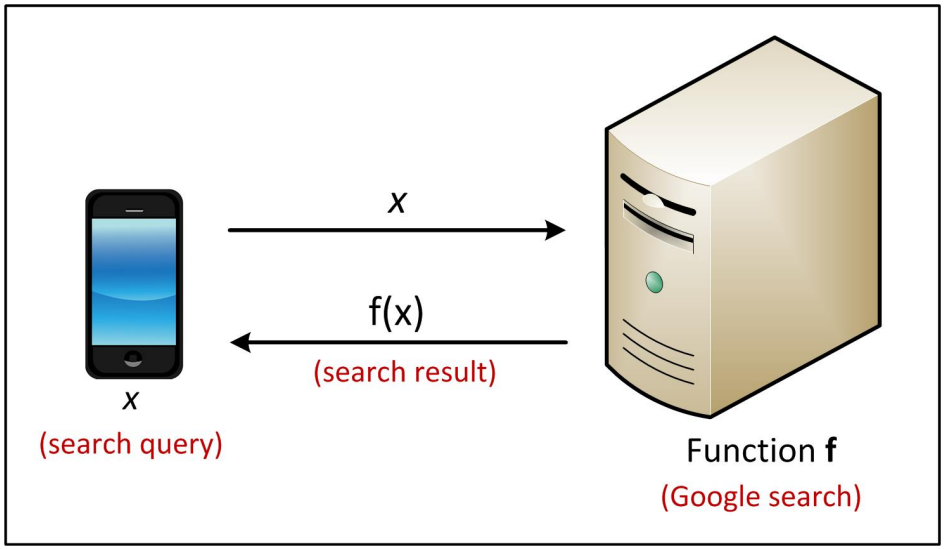
Department of Informatics
University of Bergen

Norwegian-Slovakian Workshop in Crypto
Bergen, Norway
February 10, 2016

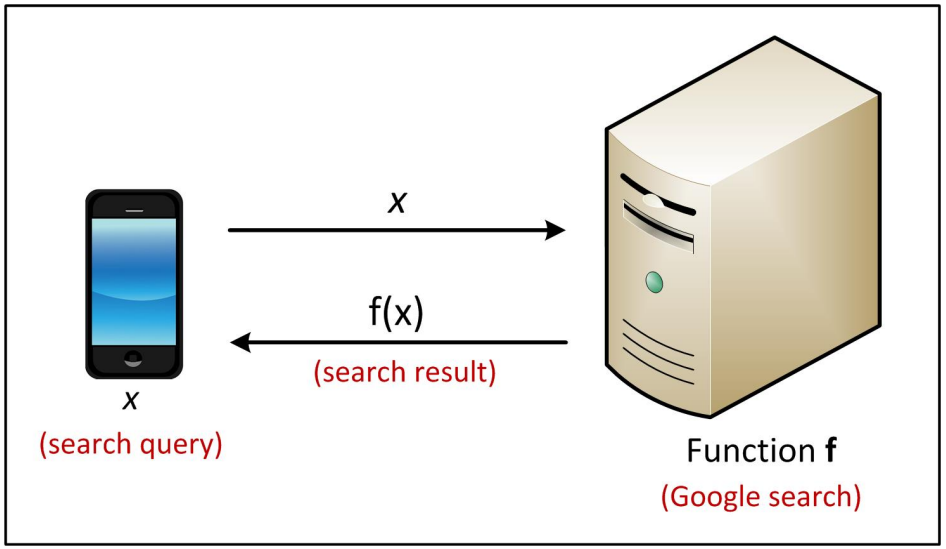
Homomorphic ...

- Homomorphic Encryption
- Homomorphic Signature
- Homomorphic MAC
- Homomorphic Hash
- ...

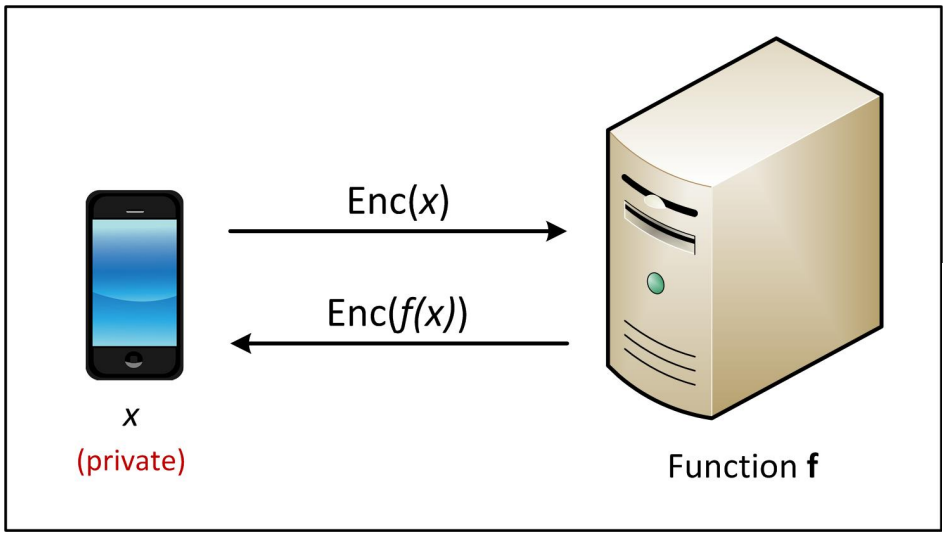
Computing on encrypted data

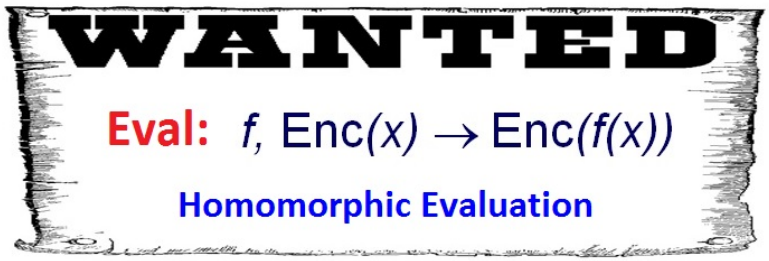


Computing on encrypted data



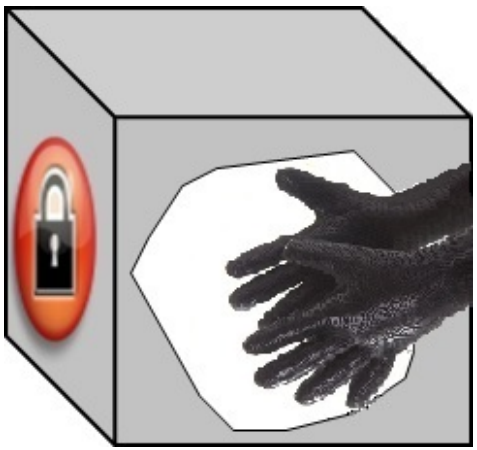
Privacy?





Homomorphic Encryption

- A way to delegate processing of data without giving access to it
- Encryption schemes that allow computations on the ciphertexts $E_k[m_1] \bullet E_k[m_2] = E_k[m_1 \circ m_2]$
- Applications:
  - E-voting: Votes are encrypted as 1 or 0 and ciphertexts are aggregated before decryption. No individual vote is revealed.
    Requires additive homomorphic encryption: $\circ$ is $+$
  - Secure cloud computing: Requires fully homomorphic encryption (homomorphic properties for both $+$ and $\times$)



Homomorphic Encryption

Multiplicative homomorphic encryption

- Unpadded RSA: $m_1^e \times m_2^e = (m_1 \times m_2)^e$
- ElGamal: Given public key $(g, h = g^a)$, ciphertexts $(g^{r_1}, h^{r_1} m_1)$ and $(g^{r_2}, h^{r_2} m_2)$, multiple both components $(g^{r_1+r_2}, h^{r_1+r_2} m_1 m_2)$

Additive homomorphic encryption

Paillier cryptosystem [Eurocrypt'99]: Additive on $\mathbb{Z}_n$

- Public key: (n, g) where p and q: two large prime, $n = pq$, $g \in_R \mathbb{Z}^*_{n^2}$
- Private key: (λ, μ) where $\lambda = lcm(p-1, q-1)$, and $\mu = (\frac{g^\lambda \bmod n^2 - 1}{n})^{-1} \bmod n$
- For encrypting $m \in \mathbb{Z}_n$: Select random $r \in_R \mathbb{Z}^*_n$
  Compute $c = g^m r^n \mod n^2$
- For decryption: compute $m = \mu \frac{c^\lambda \bmod n^2 - 1}{n} \mod n$

Homomorphic Encryption

Continued

Examples of schemes with limited functionality

- RSA works for MULT (mod N)
- Paillier works for ADD (XOR)
- BGN05 works for quadratic formulas
- MGH08 works for low-degree polynomials
  size of $c \leftarrow Eval(pk, f, c_1, ..., c_t)$ grows exponentially with degree of polynomial f

Somewhat Homomorphic Encryption (SHE)

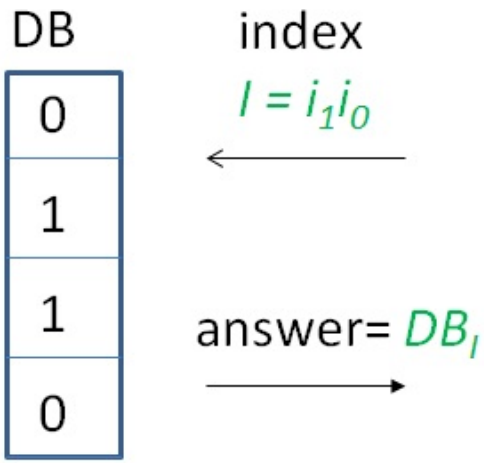
- Eval only works for some functions f

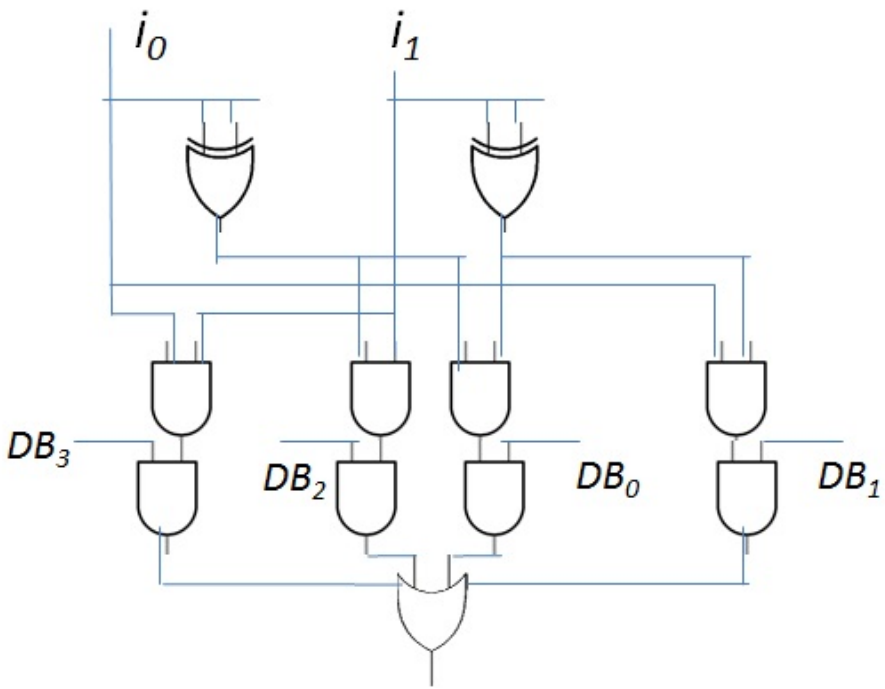
Fully Homomorphic Encryption (FHE)

- Fully means that it works for any arbitrary function f
- Supports both addition and multiplication
- Before Gentry's work (2009), no FHE scheme

Why both addition and multiplication?

- Because {XOR, AND} is Turing-complete: any function can be written as a combination of XOR and AND gates.
- If you can compute XOR and AND on encrypted bits, you can compute ANY function on encrypted inputs.





Example: Searching a database

Homomorphic Public-key Encryption

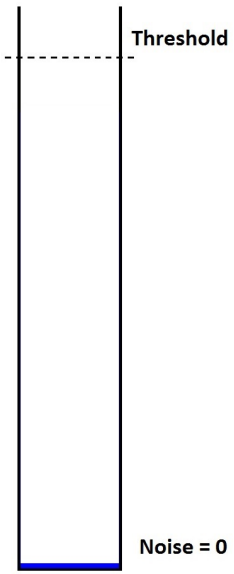
Properties

- Procedures: (KeyGen, Enc, Dec, Eval)
  $(sk, pk) \leftarrow KeyGen(\lambda)$
- Correctness: For any function f in supported family F,
  $c_1 \leftarrow Enc_{pk}(m_1)$, ... , $c_t \leftarrow Enc_{pk}(m_t)$
  $c^* \leftarrow Eval_{pk}(f, c_1, ..., c_t)$
  $Dec_{sk}(c^*) = f(m_1, ..., m_t)$
- No information about m_1, ..., m_t, and $f(m_1, ..., m_t)$ is leaked.
- Compactness: complexity of decrypting c^* does not depend on complexity of f.

SHE + Bootstrappability $\rightarrow$ FHE

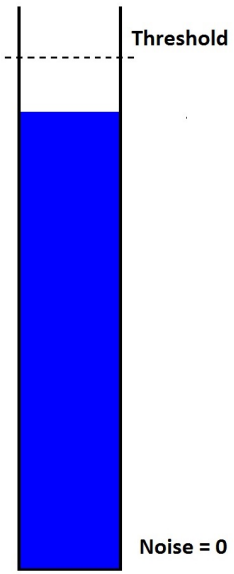
1. Construct a useful "Somewhat Homomorphic Encryption" scheme
2. Modify your SHE scheme and make it bootstrappable if it is not
3. Bootstrappable SHE $\xrightarrow[\text{Recryption}]{}$ FHE

Bootstrapping



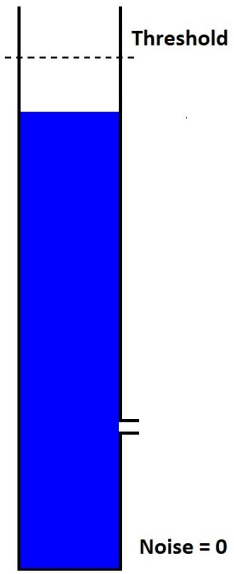
- Problem: The ciphertexts contain a random 'noise' component that grows in size as the ciphertext is processed to homomorphically evaluate a function f on its plaintext.
- Once the noise exceeds a certain level, the ciphertext can no longer be decrypted.
- Number of homomorphic operations that can be performed is limited.
- We need a noise-reduction
- The best noise-reduction procedure: Something that kills all noise and recovers the message: Decryption!
- But, the decryption should be done without releasing the secret key $\rightarrow$ We can release $Enc(sk)$: Circular Encryption
- Whenever noise level increases beyond a limit, use bootstrapping to reset it to a fixed level. Bootstrapping = "Valve" at a fixed height
- Key observation: Regardless of the noise in the input to the decryption, the noise level at the output is FIXED.
- Bootstrapping requires homomorphically evaluating the decryption circuit.
- Gentry's "bootstrapping" theorem: If an encryption scheme can evaluate its own decryption circuit, then it can evaluate everything [Gentry'09].

Bootstrapping



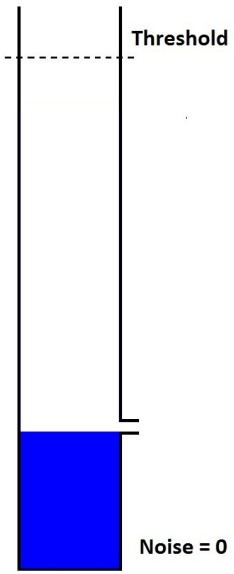
- Problem: The ciphertexts contain a random 'noise' component that grows in size as the ciphertext is processed to homomorphically evaluate a function f on its plaintext.
- Once the noise exceeds a certain level, the ciphertext can no longer be decrypted.
- Number of homomorphic operations that can be performed is limited.
- We need a noise-reduction
- The best noise-reduction procedure: Something that kills all noise and recovers the message: Decryption!
- But, the decryption should be done without releasing the secret key $\rightarrow$ We can release $Enc(sk)$: Circular Encryption
- Whenever noise level increases beyond a limit, use bootstrapping to reset it to a fixed level. Bootstrapping = "Valve" at a fixed height
- Key observation: Regardless of the noise in the input to the decryption, the noise level at the output is FIXED.
- Bootstrapping requires homomorphically evaluating the decryption circuit.
- Gentry's "bootstrapping" theorem: If an encryption scheme can evaluate its own decryption circuit, then it can evaluate everything [Gentry'09].

Bootstrapping



- Problem: The ciphertexts contain a random 'noise' component that grows in size as the ciphertext is processed to homomorphically evaluate a function f on its plaintext.
- Once the noise exceeds a certain level, the ciphertext can no longer be decrypted.
- Number of homomorphic operations that can be performed is limited.
- We need a noise-reduction
- The best noise-reduction procedure: Something that kills all noise and recovers the message: Decryption!
- But, the decryption should be done without releasing the secret key $\rightarrow$ We can release $Enc(sk)$: Circular Encryption
- Whenever noise level increases beyond a limit, use bootstrapping to reset it to a fixed level. Bootstrapping = "Valve" at a fixed height
- Key observation: Regardless of the noise in the input to the decryption, the noise level at the output is FIXED.
- Bootstrapping requires homomorphically evaluating the decryption circuit.
- Gentry's "bootstrapping" theorem: If an encryption scheme can evaluate its own decryption circuit, then it can evaluate everything [Gentry'09].

Bootstrapping



- Problem: The ciphertexts contain a random 'noise' component that grows in size as the ciphertext is processed to homomorphically evaluate a function f on its plaintext.
- Once the noise exceeds a certain level, the ciphertext can no longer be decrypted.
- Number of homomorphic operations that can be performed is limited.
- We need a noise-reduction
- The best noise-reduction procedure: Something that kills all noise and recovers the message: Decryption!
- But, the decryption should be done without releasing the secret key
  $\rightarrow$ We can release $Enc(sk)$: Circular Encryption
- Whenever noise level increases beyond a limit, use bootstrapping to reset it to a fixed level. Bootstrapping = "Valve" at a fixed height
- Key observation: Regardless of the noise in the input to the decryption, the noise level at the output is FIXED.
- Bootstrapping requires homomorphically evaluating the decryption circuit.
- Gentry's "bootstrapping" theorem: If an encryption scheme can evaluate its own decryption circuit, then it can evaluate everything [Gentry'09].

Recryption

- A central aspect in Gentry's FHE (and subsequent schemes).
- It allows to refresh a ciphertext: given a ciphertext C for some plaintext M, compute a new ciphertext C' for M (possibly for a different key) such that the size of the noise in C' is smaller than the size of the noise in C.
- By periodically refreshing the ciphertext (e.g., after computing each gate in f), one can evaluate arbitrarily large circuits f.
- The Recrypt operation is implemented by evaluating the decryption circuit of the encryption scheme homomorphically, given 'fresh' (low noise) ciphertexts for the bits of the ciphertext to be refreshed and the scheme's secret key.

Recryption

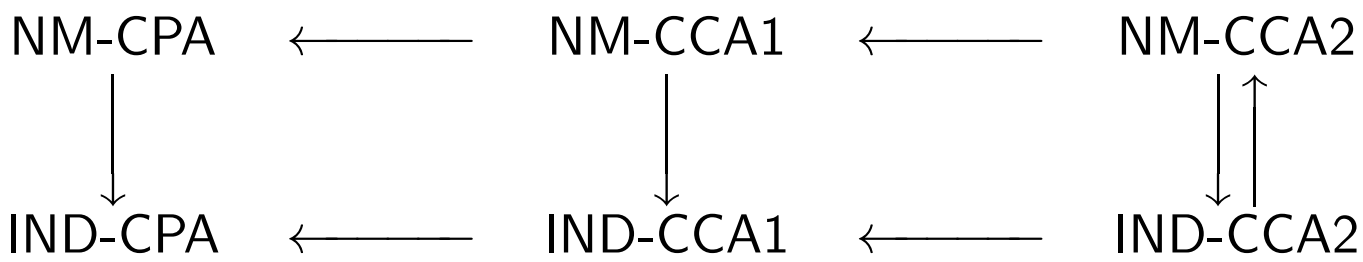
Continued

- $Recrypt_{\mathcal{E}}(pk_2, D_{\mathcal{E}}, \overline{sk_1}, c_1)$:
  - Generate $\overline{c_1}$ via $Enc_{\mathcal{E}}(pk_2, c_{1j})$ over the bits of c_1
  - Output $c \leftarrow Eval_{\mathcal{E}}(pk_2, D_E, \overline{sk_1}, \overline{c_1})$
- $Eval_{\mathcal{E}}$ takes in the bits of sk_1 and c_1, each encrypted under pk_2.
- $\mathcal{E}$ is used to evaluate the decryption circuit homomorphically.
- As long as $\mathcal{E}$ can handle $D_{\mathcal{E}}$, the output c is an encryption under pk_2 of $Dec_{\mathcal{E}}(sk_1, c_1) = m$.
- $Recrypt_{\mathcal{E}}$ therefore outputs a new encryption of m, but under pk_2.
- The $Eval_{\mathcal{E}}$ algorithm is used to remove the inner encryption: Box $\#i$ is unlocked while it is inside box $\#(i+1)$.
- By recursing this process, we obtain a fully homomorphic encryption scheme: The public key in $\mathcal{E}^{\dagger}$ consists of a sequence of public keys $(pk_1, ..., pk_{\ell+1})$ and a chain of encrypted secret keys $\{\overline{sk_1}, ..., \overline{sk_\ell}\}$, where sk_i is encrypted under pk_{i+1}.

Homomorphic Public-key Encryption

Semantic security

- Procedures: (KeyGen, Enc, Dec, Eval)
- Semantic security is defined like basic encryption.
- Notions of security in basic public-key encryption schemes:



- Malleability of ciphertexts $\rightarrow$ Homomorphic encryption cannot achieve IND-CCA2.
- FHE schemes that adopt Gentry's bootsrapping technique might not be CCA1-secure.

Hard Problems

For constructing homomorphic encryption schemes

- Shortest Vector Problem (SVP): shortest possible vector in the lattice
- Closest Vector Problem (CVP): closest vector to a point
- Learning With Errors (LWE): a generalization to "*parity with noise*" problem
  - Polynomial Learning With Errors (PLWE)
  - Ring Learning With Errors (RLWE)
- Sparse Subset Sum Problem (SSSP)
- Bounded Distance Decoding (BDD)
- Approximate Greatest Common Divisor (AGCD)
- Polynomial Coset Problem (PCP): related to Ideal Coset Problem

FHE and Functional Encryption (FE)

- FHE: compute $Enc(f(x))$ from $Enc(x)$ for any function f.
- FE: compute $f(x)$ from $Enc(x)$.
- For functions of the type Enc_f, where $Enc_f(x) = Enc(f(x))$ is a re-encryption of $f(x)$, FE would be very close to constructing an FHE scheme.
- Randomized FE can be used for constructing FHE [ABFGGTW'13].
- Randomized FE: An encryptor is able to hide an input within a ciphertext so that authorized decryptors can only recover the result of applying a randomized function to it.

FHE and Obfuscation

- Obfuscation:
  - The cloud is given an "encrypted" program $E(P)$.
  - For any input x, cloud can compute $E(P)(x) = P(x)$.
  - Cloud learns nothing about P, except $\{x_i, P(x_i)\}$.
- FHE does not provide obfuscation automatically.
- It is possible to use obfuscated circuits to obtain Randomized Functional Encryption schemes suitable for FHE constructions [ABFGGTW'13]:
  Obfuscated circuits $\rightarrow$ Randomized Functional Encryption $\rightarrow$ FHE

Multi-key FHE

- Different clients encrypt data under different FHE keys.
- The cloud combines data encrypted under different keys:
  $Enc_{pk_1,...,pk_t}(f(m_1,...,m_t)) \leftarrow Eval(pk_1,...,pk_t,f,c_1,...,c_t)$
- FHE does not provide it automatically.
- It is possible to construct FHE schemes with above property: [LATV12] "On-the-fly Multiparty Computation on the Cloud via Multi-key FHE."

A Construction of FHE [DGHV'10]

1. Construct a Symmetric Somewhat Homomorphic Encryption (under the approximate GCD assumption)
2. By a simple transformation, convert it to a Public-key Somewhat Homomorphic Encryption (under the approximate GCD assumption)
3. Use Gentry's techniques to have a public-key FHE (under approximate GCD + sparse subset sum)

Approximate GCD Problem

- Given many $x_i = s_i + q_i p$, output p
- Example parameters: $s_i \sim 2^{\lambda}$, $p \sim 2^{\lambda^2}$, $q_i \sim 2^{\lambda^5}$ (λ: security parameter)
- Best known attacks (lattice-based): $\sim 2^{\lambda}$ time

A Construction of FHE [DGHV'10]

Step 1: Constructing a symmetric homomorphic encryption scheme

Secret key

large odd number p

Encryption steps of a bit m

Choose at random large q and small r

$c = pq + 2r + m$

If $2r + m \ll p$ then ciphertext is close to a multiple of p

Parameters: $|r| = n$, $|p| = n^2$ and $|q| = n^5$

Decryption

$m \equiv (c \mod p) \mod 2$

Why is it homomorphic?

$c_1 = pq_1 + 2r_1 + m_1, \qquad c_2 = pq_2 + 2r_2 + m_2$

- $c_1 + c_2 = (q_1 + q_2)p + 2(r_1 + r_2) + (m_1 + m_2)$
  If $(r_1 + r_2) \ll \frac{p}{2} \Rightarrow (c_1 + c_2 \mod p) \mod 2 \equiv m_1 + m_2 (mod 2)$
  Noise $= 2 \times$ (Initial noise)
- $c_1 c_2 = (q_1 q_2 p + 2q_1 r_2 + q_1 m_2 + 2q_2 r_1 + q_2 m_1)p + 2(2r_1 r_2 + r_1 m_2 + m_1 r_2) + m_1 m_2$
  If $(2r_1 r_2 + r_1 m_2 + m_1 r_2) \ll \frac{p}{2} \Rightarrow (c_1 c_2 \mod p) \mod 2 \equiv m_1 m_2 (mod 2)$
  Noise $=$ (Initial noise)^2

Comparison of Fully Homomorphic Encryption Schemes

Scheme	Year	Underlying Problems	Asymptotic Runtime	Concrete Runtime
Gentry: A Fully Homomorphic Encryption Scheme	2009	BDDP & SSSP	$O(\lambda^{3.5})$ per gate for ciphertext refreshing	-
van Dijk, Gentry, Halevi, Vaikuntanathan: FHE over the Integers	2010	AGCD & SSSP	Public key size: $O(\lambda^{10})$, no gate cost given	-
Smart, Vercauteren: FHE with Relatively Small Key and Ciphertext Sizes	2010	PCP & SSSP	Key generation: $O(\log n.n^{2.5})$	Key generation: several hours even for small parameters, for larger parameters the keys could not be generated
Brakerski, Vaikuntanathan: Efficient FHE from (standard) LWE	2011	DLWE	Evaluation key size: $O(\lambda^{2C}\log(\lambda))$	-
Brakerski, Vaikuntanathan: FHE from Ring-LWE and Security for Key Dependent Messages	2011	PLWE	Very cheap key generation, unknown for bootstrapping	-
Brakerski, Gentry, Vaikuntanathan: FHE without Bootstrapping	2011	RLWE	Per-gate computation overhead $O(d^3\lambda\log\lambda)$ without bootstrapping, $O(\lambda^2\log\lambda)$ with bootstrapping	36 hours for an AES encryption on a supercomputer

d: Depth of the circuit, n: Dimension of the lattice, C: A very large parameter for ensuring bootstrappability

Comparison of Fully Homomorphic Encryption Schemes

Continued

Gentry, Halevi: Implementing Gentry's Fully-Homomorphic Encryption Scheme	2011	SVP & BDD	Key generation: $O(\log n.n^{1.5})$	Bootstrapping: From 30s (for small setting) to 30 min (for large setting)
Coron, Naccache, Tibouchi: Public Key Compression and Modulus Switching for FHE over the Integers	2012	DAGCD & SSSP	Public key size: $O(\lambda^5 \log(\lambda))$, no gate cost given	Recryption: 11 min
Rohloff, Cousins: A Scalable Implementation of Fully Homomorphic Encryption Built on NTRU	2014	SVP & RLWE	-	Recryption: 275s on 20 cores with 64-bit security
Halevi, Shoup: Bootstrapping for HElib	2015	RLWE	-	Vectors of 1024 elements from $GF(2^{16})$ was recrypted in 5.5 min at security level $\approx$ 76, single CPU core

Table From: Armknecht et al. [ABCGJRS'15]

Future works

Project

Cryptographic Tools for Cloud Security
Funded by the Norwegian Research Council

Partners

- NTNU (Department of Telematics + Department of Mathematics)
- Simula@UiB

Questions?

Thank you!