Homomorphic Encryption: Theory and Application

Cody Tinker

What is Homomorphic Encryption (HE)?

- Homomorphism: A structure-preserving map between two algebraic structures
  - $f : (A, \circ) \mapsto (B, *)$
  - $f(a \circ b) = f(a) * f(b)$
- An encryption scheme that allows for encrypted data to be evaluated with some operations in the scope of the encrypted group of which the result of the operation will decrypt to the result of a corresponding operation performed on the plaintext data in the scope of the plaintext group.

Motivation

- Privacy of volatile sensitive data
  - ex. Medical records in a database
- Privacy of outsourced processing
  - ex. Cloud computing

Types of Homomorphic Encryption

- Partially Homomorphic Encryption (PHE)
  - Allows one type of operation on encrypted data without limit.
  - Used in a few practical applications: e-voting, Private Information Retrieval (PIR)
  - Example Schemes: RSA, Goldwasser-Micali, El-Gamal, Benaloh, Paillier, ect.
- Somewhat Homomorphic Encryption (SWHE)
  - Allows a set of operations on encrypted data usually a limited number of times.
  - Could be used in applications where the depth of the evaluation operation is constant or known.
  - Example Schemes: Polly Cracker, Boneh-Goh-Nissim (BGN)
- Fully Homomorphic Encryption (FHE)
  - Allows an unlimited number of operations on encrypted data without limit.
  - This is the “holy grail”.
  - Theoretically, can be used for any application.

Gentry's Breakthrough

- SWHE schemes allow a limited number of operations on data before data is decrypted incorrectly.
- The ciphertext needs to be transformed to reduce the noise.
  - Ex. Decrypt then Encrypt (The data is compromised for a short period!)
- Gentry proposed two genius blueprint methods: Squashing and Bootstrapping
- Bootstrapping: a "recrypting" procedure to produce a "fresh" ciphertext
  - Two different key pairs: (pk1, sk1) and (pk2, sk2)
  - A noisy ciphertext c: c = $Enc_{pk1}(m)$
  - Encrypt sk1: $Enc_{pk1}(sk1)$
  - Decrypt the ciphertext homomorphically with $Enc_{pk1}(sk1)$
  - Encrypt the result with pk2: $Enc_{pk2}(Dec_{sk1}(c)) = Enc_{pk2}(m)$
- Squashing: a technique for reducing the complexity of the decryption "circuit"
  - Create a set of elements whose sum equals to the multiplicative inverse of the secret key. (SSSP)
  - Multiply and sum the ciphertext with each element of the set.

FHE Framework (Standardization)

- **ParamGen(λ, P, K, B)** → Params
  - Generate scheme parameters based on security parameter λ and other parameters.
- **PubKeygen(Params) → SK, PK, EK** *or* **SecKeygen → SK, EK**
  - Generate keys (secret, public, evaluation) to handle data.
- **PubEncrypt(PK, M) → C** *or* **SecEncrypt(SK, M) → C**
  - Encryption function.
- **Decrypt(SK, C) → M**
  - Decryption function.
- **EvalAdd(Params, EK, C1, C2) → C3** *and* **EvalMult(Params, EK, C1, C2) → C3**
  - Evaluation schemes to manipulate underlying plaintext data.
  - Should be able to use these to evaluate generate circuit *f()*.
- **Refresh(Params, flag, EK, C1) → C2**
  - Scheme for manipulating a “noisy” or “complex” ciphertext into a “fresh” or “simple” ciphertext.
  - Parameter *flag* is multivalued: “Relinearize”, “ModSwitch”, “Bootstrap”, etc.

Mathematical Construct: Rings

- A fundamental algebraic structure, much like groups and fields.
- Groups $\subseteq$ Rings $\subseteq$ Fields
- Consists of a set with two operations, often denoted by addition and multiplication.
- $(R, +, \cdot)$
- Polynomial Ring: K[x], in X over a field K
  - $a_0 + a_1X + a_2X^2 + a_3X^3 + \ldots$ such that $a_i \in K$
- Quotient Ring: $K[x]/(b_0 + b_1X + \ldots + b_2X^m)$
  - $a_0 + a_1X + a_2X^2 + \ldots \pmod{b_0 + b_1X + \ldots + b_2X^m}$
  - Mod term often $X^n + 1$; n is a power of 2

Must satisfy the following axioms with $a,b,c \in R$:

1. R is an abelian group under addition:
   - $(a + b) + c = a + (b + c)$
   - $a + b = b + a$
   - There exists an additive identity
   - There exists additive inverses
2. R is a monoid under multiplication:
   - $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
   - There usually exists a multiplicative identity (though not required)
3. Multiplication is distributive with respect to addition:
   - $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
   - $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$

Hardness Assumption: Learning With Errors (LWE)

"Given $q \geq 2$ and $n \in Z^+$, the Learning With Errors problem can be simply stated as recover $s \in Z_q^n$ from a sequence of "approximate" random linear equations."

Example:

$12s_1 + 3s_2 + 7s_3 + 3s_4 \approx 9 \bmod 15$
$3s_1 + 1s_2 + 3s_3 + 7s_4 \approx 3 \bmod 15$
$4s_1 + 8s_2 + 9s_3 + 3s_4 \approx 1 \bmod 15$
$14s_1 + 13s_2 + 2s_3 + 1s_4 \approx 12 \bmod 15$
$\vdots$
$11s_1 + 5s_2 + 10s_3 + 7s_4 \approx 3 \bmod 15$

Solve for s. (s is actually [2, 13, 5, 0])

Alternatively, the problem can be states as "given a set of samples, determine whether they originated from the $A_{s,X}$ oracle for some s or whether they follow the uniform distribution on $Z_q^n \times Z_q$."

In other words, it is hard to distinguish a set of samples generated from $(a, \langle as \rangle + e)$ where a is a sampled vector from Z_q^n uniformly at random and e is an element sampled from Z_q based on some distribution X from samples (a, u) where u is sampled from Z_q uniformly at random.

The problem is reduced to hard lattice problems.

Hardness Assumption: Ring-Learning With Errors (R-LWE)

Extension of LWE but with ring elements.

Example:

Let $R_q = Z_q[x]/\Phi(x)$

Let $a_i, b_i, s \in R_q$

$a_1 \cdot s \approx b_1 \bmod \Phi(x)$

$a_2 \cdot s \approx b_2 \bmod \Phi(x)$

$a_3 \cdot s \approx b_3 \bmod \Phi(x)$

$a_4 \cdot s \approx b_4 \bmod \Phi(x)$

$\vdots$

$a_m \cdot s \approx b_m \bmod \Phi(x)$

Solve for s.

Alternatively, the problem can be states as given a set of pairs (a_i, b_i), determine whether b_i originated from the $A_{s,X}$ oracle for some s or whether they follow the uniform distribution on $R_q \times R_q$.

Generally more efficient to compute and store elements related to this problem over LWE. For example, the samples $(a,b) \in R_q \times R_q$ of the R-LWE problem can replace n samples of $(x, y) \in Z_q^n \times Z_q$ of the LWE problem. This often results in a reduction of public and private keys by a factor of n over a system using LWE.

Security is still somewhat an open problem but there is a decent amount of confidence that there is not much of a security trade-off.

The Basic Idea: Encryption and Decryption

Parameters:

- Based on some secret element p and small, random distributions (Gaussian), denoted X
- m ∈ {0, 1}
- e, q ← X;

Encryption:

$c = Enc(m) = m + 2e + pq$

Decryption:

$m = Dec(c) = (c \bmod p) \bmod 2$

Correctness:

$Dec(c) = (c \bmod p) \bmod 2$

$Dec(c) = (m + 2e + pq \bmod p) \bmod 2$

$Dec(c) = (m + 2e) \bmod 2$

$Dec(c) = m$

Decryption works if $(m + 2e) < p/2$

The Basic Idea: Addition

Elements:

$c_1 = Enc(m_1)$

$c_2 = Enc(m_2)$

Addition:

$c_3 = c_1 + c_2$

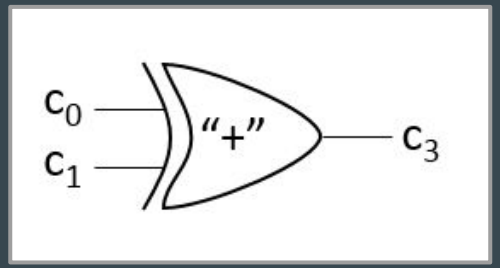
Correctness:

$c_3 = c_1 + c_2$

$c_3 = m_1 + 2e_1 + pq_1 + m_2 + 2e_2 + pq_2$

$c_3 = (m_1 + m_2) + 2(e_1 + e_2) + p(q_1 + q_2)$

$c_3 = (m_1 + m_2) + 2e_3 + pq_3$



The Basic Idea: Multiplication

Elements:

$c_1 = Enc(m_1)$

$c_2 = Enc(m_2)$

Addition:

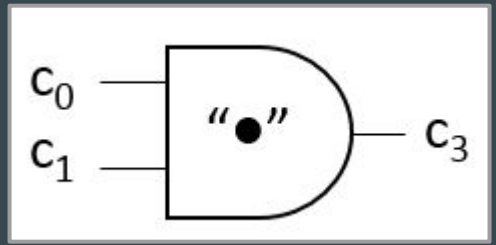
$c_3 = c_1 c_2$

Correctness:

$c_3 = c_1 c_2$

$c_3 = (m_1 + 2e_1 + pq_1)(m_2 + 2e_2 + pq_2)$

$c_3 = m_1 m_2 + 2(m_1 e_2 + m_2 e_1 + 2e_1 e_2) + kp$



Example Scheme: YASHE

YASHE.ParamsGen(λ): Given the security parameter λ, fix a positive integer d that determines R, moduli q and t with $1 < t < q$, distributions $\chi_{\text{key}}, \chi_{\text{err}}$ on R, and an integer base $w > 1$. Output $(d, q, t, \chi_{key}, \chi_{err}, w)$.

YASHE.KeyGen$(d, q, t, \chi_{\text{key}}, \chi_{\text{err}}, w)$: Sample $f', g \leftarrow \chi_{\text{key}}$ and let $f = [tf' + 1]_q$. If f is not invertible modulo q, choose a new f'. Compute the inverse $f^{-1} \in R$ of f modulo q and set $h = [tgf^{-1}]_q$. Sample $\mathrm{e}, \mathrm{s} \leftarrow \chi_{err}^{\ell_{w,q}}$, compute $\gamma = [\mathsf{PowersOf}_{w,q}(f) + \mathrm{e} + h \cdot \mathrm{s}]_q \in R^{\ell_{w,q}}$ and output $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) = (h, f, \gamma)$.

YASHE.Encrypt(h, m): The message space is R/tR. For a message $m + tR$, sample $s, e \leftarrow \chi_{\text{err}}$, and output the ciphertext $c = [\Delta[m]_t + e + hs]_q \in R$.

YASHE.Decrypt(f, c): Decrypt a ciphertext c by $m = \left[\left\lfloor \frac{t}{q} \cdot [fc]_q \right\rceil\right]_t \in R$.

YASHE.Add(c_1, c_2): Output $c_{\text{add}} = [c_1 + c_2]_q$.

YASHE.KeySwitch$(\tilde{c}_{\text{mult}}, \mathsf{evk})$: Output the ciphertext$[\langle \mathsf{WordDecomp}_{w,q}(\tilde{c}_{\text{mult}}), \mathsf{evk} \rangle]_q$.

YASHE.Mult(c_1, c_2, evk): Output the ciphertext

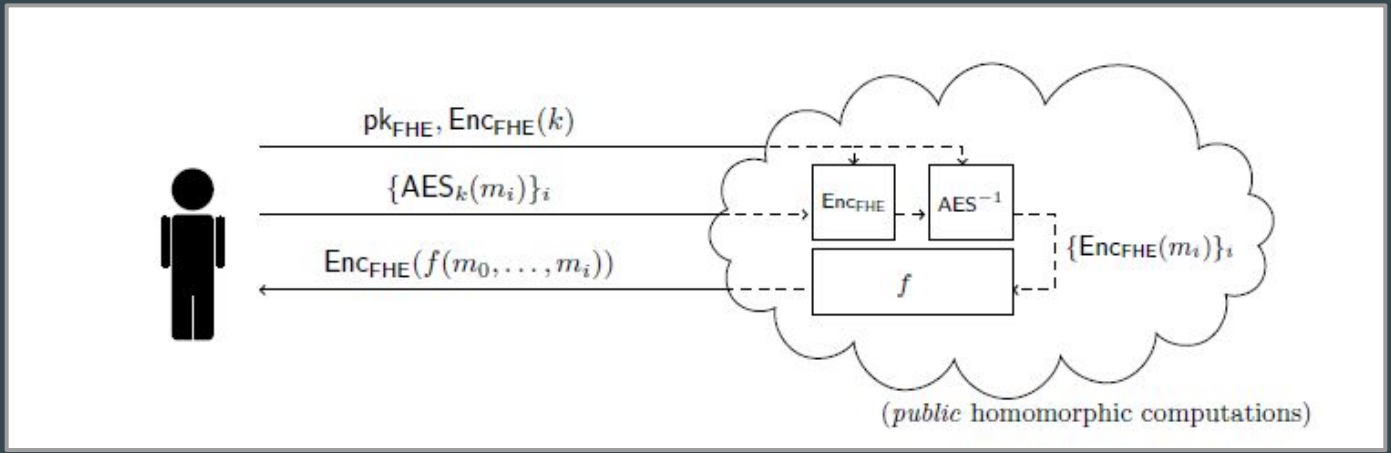
$$c_{\text{mult}} = \mathsf{YASHE.KeySwitch}(\tilde{c}_{\text{mult}}, \mathsf{evk}), \text{ where } \tilde{c}_{\text{mult}} = \left[\left\lfloor \frac{t}{q} c_1 c_2 \right\rceil\right]_q.$$

Recommended Security Parameters for RLWE Based Schemes

distribution	n	security level	logq	uSVP	dec	dual
(-1, 1)	1024	128	22	134.3	184.0	149.9
		192	16	200.9	306.7	227.5
		256	13	256.1	428.6	296.4
	2048	128	44	131.4	156.5	139.3
		192	33	193.0	241.2	206.2
		256	26	260.5	344.7	280.0
	4096	128	88	129.9	143.1	134.6
		192	66	192.7	217.4	199.7
		256	53	256.7	297.3	269.5
	8192	128	177	128.5	134.3	130.2
		192	132	192.1	205.0	196.5
		256	106	257.0	278.5	263.1
	16384	128	353	128.5	131.4	129.9
		192	264	192.1	198.3	194.2
		256	213	256.1	266.6	260.5
	32768	128	706	128.5	129.6	129.1
		192	527	192.7	195.6	194.2
		256	426	256.1	261.3	261.6

Application Example: Overview

- Craig Gentry, Shai Halevi, and Nigel Smart implemented a homomorphically evaluated AES circuit using the BGV scheme under the guise of HElib in 2012. Updated and optimized in 2015.
- AES is a non-trivial, “real world” function composed of a combination of complex and simple operations.
- Has practical application uses for real-world cloud computing environments.
    - Cut down communication cost by only sending AES encrypted data and reshaping the data in the cloud scope.



Application Example: Results

- Initial implementation (2012) had discouraging results
  - AES rounds took several hours to compute and required 256GB of RAM to store public keys
- Recent implementation implemented optimized HE techniques
  - Key Switching: Switch key that a ciphertext is valid with to another key
  - Modulus Switching: Reduce the norm of the noise (reduce noise term)
  - Batching: Pack multiple pieces of data in a ciphertext for SIMD operations
  - Dynamic Noise Management: Keep an estimate of the noise stored with the ciphertext to determine when to refresh
- Implementation ran on a Lenovo X230 laptop
  - Ubuntu 13.03 VM
  - Intel Core i5-3320M at 2.6GHz
  - 4GB of RAM
  - g++ 4.9.2 compiler
  - Entire program is single-threaded

Application Example: Results

- No boostrapping
  - 120 AES blocks packed in each ciphertext
  - Computations based on 120 AES cleartext blocks
  - ~2s per block throughput
- Bootstrapping
  - 60 AES blocks packed in each ciphertext
  - Computations based on 180 AES cleartext blocks
  - ~5.8s per block throughput

Test	m	$\phi(m)$	lvls	$\|Q\|$	security	params/key-gen	Encrypt	Decrypt	memory
no bootstrap	53261	46080	40	886	150-bit	26.45 / 73.03	245.1	394.3	3GB
bootstrap	28679	23040	23	493	123-bit	148.2 / 37.2	1049.9	1630.5	3.7GB

References

[1] Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., ... Sahai, A. (2018). Homomorphic Encryption Standard.

[2] Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2017). A Survey on Homomorphic Encryption Schemes: Theory and Implementation, 1–35. https://doi.org/10.1145/0000000.0000000

[3] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. Proceedings of the 41st Annual ACM Symposium on Symposium on Theory of Computing - STOC '09, 169. https://doi.org/10.1145/1536414.1536440

[4] Lepoint, T. E., & Naehrig, M. (n.d.). A Comparison of the Homomorphic Encryption Schemes FV and YASHE, 1–18. Retrieved from https://eprint.iacr.org/2014/062.pdf

[5] Gentry, C., & Smart, N. P. (2015). Homomorphic Evaluation of the AES Circuit. Retrieved from http://eprint.iacr.org/2012/099.pdf

[6] Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2011). Fully Homomorphic Encryption without Bootstrapping, 1–35. Retrieved from http://eprint.iacr.org/2011/277%5Cnhttp://eprint.iacr.org/2011/277.pdf

[7]Van Dijk, M., Gentry, C., Halevi, S., & Vaikuntanathan, V. (2010). Fully homomorphic encryption over the integers. Advances in Cryptology– EUROCRYPT '10, 24–43. https://doi.org/10.1007/978-3-642-38348-9_20