Fully Homomorphic Encryption

Zvika Brakerski
Weizmann Institute of Science
Outsourcing Computation

Email, web-search, navigation, social networking...

What if $x$ is private?

Search query, location, business information, medical information...
The Situation Today

We promise we won't look at your data. Honest!

We want real protection.
Outsourcing Computation – Privately

\[ x \rightarrow Enc(x) \rightarrow y \rightarrow Dec(y) = f(x) \]

WANTED
Homomorphic Evaluation function:
\[ Eval: f, Enc(x) \rightarrow Enc(f(x)) \]

Learns nothing on \( x \).
Fully Homomorphic Encryption (FHE)

\[ E_{pk}(x) \]

\[ y = \text{Eval}_{evk}(f, Enc(x)) \]

Correctness:
\[ \text{Dec}_{sk}(y) = f(x) \]

Input privacy:
\[ Enc(x) \approx Enc(0) \]

Fully Homomorphic = Correctness for any efficient \( f \)

= Correctness for universal set

- NAND.
- \((+,\times)\) over \(\mathbb{Z}_2\) (= binary XOR, AND)
Trivial FHE?

PKE $\Rightarrow$ “FHE”:

- $\text{Keygen}$ and $\text{Enc}$: Same as PKE.
- $\text{Eval}^{\text{FHE}}(f, c)$
- $\text{Dec}^{\text{FHE}}(f, c) \triangleq f(\text{Dec}_{sk}(c)) = f\left(\text{Dec}_{sk}(\text{Enc}(x))\right) = f(x)$

NOT what we were looking for...

All work is relayed to receiver.

Compact FHE: $Dec$ time does not depend on ciphertext.

$\Rightarrow$ ciphertext length is globally bounded.

In this talk (and in literature) $\text{FHE} \triangleq \text{Compact-FHE}$
Trivial FHE?

PKE ⇒ “FHE”:
- $Keygen$ and $Enc$: Same as PKE.
- $Eval^{FHE} (f, c)$
- $Dec_{sk}^{FHE} (f, c) \triangleq f(Dec_{sk} (c))$

This “scheme” also completely reveals $f$ to the receiver.
Can be a problem.

Circuit Privacy: Receiver learns nothing about $f$ (except output).

Compactness ⇒ Circuit Privacy (by complicated reduction) [GHV10]

Circuit private FHE is not trivial to achieve – even non-compact.

In this talk: Only care about compactness, no more circuit privacy.
Applications

In the cloud:

• Private outsourcing of computation.
• Near-optimal private outsourcing of storage (single-server PIR). [G09,BV11b]
• Verifiable outsourcing (delegation). [GGP11,CKV11]
• Private machine learning in the cloud. [GLN12,HW13]

Secure multiparty computation:

• Low-communication multiparty computation. [AJLTVW12,LTV12]
• More efficient MPC. [BDOZ11,DPSZ12,DKLPSS12]

Primitives:

• Succinct argument systems. [GLR11,DFH11,BCCT11,BC12,BCCT12,BCGT13,...]
• General functional encryption. [GKPVZ12]
• Indistinguishability obfuscation for all circuits. [GGHRSW13]
Verifiable Outsourcing (Delegation)

What if the server is cheating?

Can send wrong value of $f(x)$. 

Need proof!
FHE ⇒ Verifiable Outsourcing

FHE ⇒ Verifiability and Privacy.

1. Verifiability with preprocessing under “standard” assumptions: [GGP10, CKV10].

2. Less standard assumptions but without preprocessing via SNARGs/SNARKs [DCL08, BCCT11,...] (uses FHE or PIR).

Pre-FHE solutions: multiple rounds [K92] or random oracles [M94].
FHE $\Rightarrow$ Verifiable Outsourcing [V10]

Preprocessing:
\[ c_0 = Enc(0) \]
\[ z_0 = Eval(f, c_0) \]

Verification:
Check \( y_0 = z_0 \)?

Yes $\Rightarrow$ output \( Dec(y_x) \)
No $\Rightarrow$ output $\perp$

Idea: “Cut and choose”

\( c_x, c_0 \) look the same $\Rightarrow$ cheating server will be caught w.p. \( \frac{1}{2} \)
(easily amplifiable)

\[ f \]

Server executes
\[ y = Eval(f, c) \]

But preprocessing is as hard as computation!
FHE $\Rightarrow$ Verifiable Outsourcing [CKV10]

**Preprocessing:**
- $c_0 = Enc(0)$
- $z_0 = Eval(f, c_0)$

$(evk'', Enc''(c_x)), (evk', Enc'(c_0))$

**Verification:**

Check $Dec'(y'_0) = z_0$?

- Yes $\Rightarrow$ output $Dec''(Dec(y_x))$
- No $\Rightarrow$ output $\perp$

**Server executes**
- $y' = Eval'(Eval(f, \cdot), c')$
- $y'' = Eval''(Eval(f, \cdot), c'')$

**Idea:** Outer layer keeps server “oblivious” of $z_0$.

$\Rightarrow$ Can recycle $z_0$ for future computations.
FHE Timeline

**Basic scheme:** Ideal cosets in polynomial rings. ⇒ Bounded-depth homomorphism.

- **Assumption:** hardness of (quantum) apx. short vector in ideal lattice.

**Bootstrapping:** bounded-depth HE ⇒ full HE.

But bootstrapping doesn’t apply to basic scheme...

- **Need additional assumption:** hardness of sparse subset-sum.

... is it even possible?
The FHE Challenge

Make it simpler.
Simplified basic scheme [vDGHV10,BV11a]
- Under similar assumptions.

Make it more secure.
?

Make it practical.
Optimizations [SV10,SS10,GH10]
FHE without Ideals [BV11b]

Linear algebra instead of polynomial rings

**Assumption:** Apx. short vector in arbitrary lattices (via LWE).

**Shortest-vector Problem (SVP):**

Fundamental algorithmic problem – extensively studied.

[LLL82,K86,A97,M98,AKS03,MR04,MV10]
FHE without Ideals [BV11b]

Linear algebra instead of polynomial rings

**Assumption:** Apx. short vector in arbitrary lattices (via LWE).

- **Basic scheme:** noisy linear equations over $\mathbb{Z}_q$.
  - Ciphertext is a linear function $c(x)$ s.t. $c(sk) \approx m$.
  - Add/multiply functions for homomorphism.
  - Multiplication raises degree ⇒ use relinearization.

- **Bootstrapping:** Use dimension-modulus reduction to shrink ciphertexts.

- Simpler: straightforward presentation.
- More secure: based on a standard assumption.
- Efficiency improvements.

Concurrently [GH11]: Ideal lattice based scheme without squashing.
FHE without Ideals

Follow-ups:

• [BGV12]: Improved parameters.
  – Even better security.
  – Improved efficiency in ring setting using “batching”.
  – Batching without ideals in [BGH13].

• [B12]: Improved security.
  – Security based on classical lattice assumptions.
  – Explained in blog post [BB12].

Various optimizations, applications and implementations:

[LNV11, GHS12a, GHS12b, GHS12c, GHPS12, AJLTVW12, LTV12,
DSPZ12, FV12, GLN12, BGHWW12, HW13 ...]
The “Approximate Eigenvector” Method [GSW13]

Ciphertexts = Matrix

Same assumption and keys as before – ciphertexts are different

- **Basic scheme:** Approximate eigenvector over $\mathbb{Z}_q$.
  - Ciphertext is a matrix $C$ s.t. $C \cdot sk \approx m \cdot sk$.
  - Add/multiply matrices for homomorphism*.

- **Bootstrapping:** Same as previous schemes.

- Simpler: straightforward presentation.
- New and exciting applications “for free”! IB-FHE, AB-FHE.
- Same security as [BGV12, B12].
- Unclear about efficiency: some advantages, some drawbacks.
What is the best way to evaluate a product of $k$ numbers?

Sequentialization [BV13]

Parallel vs. Sequential

Conventional wisdom

Actually better (if done right)
Sequentialization [BV13]

Barrington’s Theorem [B86]: Every depth $d$ computation can be transformed into a width-5 depth $4^d$ branching program.

- Better security – breaks barrier of [BGV12, B12, GSW13].
- Using dimension-modulus reduction (from [BV11b]) ⇒ same hardness assumption as non homomorphic encryption.
- Short ciphertexts.
Efficiency

Standard benchmark: AES128 circuit

Implementations of [BGV12] by [GHS12c,CCKLLTY13] $\approx 5$ min/input

Limiting factors:

- Circuit representation.
- Bootstrapping.
- Key size.

2-years ago it was 3 min/gate [GH10]

New works [GSW13,BV13] address some of these issues, but have other drawbacks

⇒ To be practical, we need to improve the theory.

See also HElib

https://github.com/shaih/HElib
Hybrid FHE

- In known FHE encryption is slow and ciphertexts are long.
- In symmetric encryption (e.g., AES) these are better.

Best of both worlds?
Hybrid FHE

\[ \text{Dec}_{sk}(y) = f(x) \]

Easy to encrypt, ciphertext is short... But how to do Eval?

Define: \( h(z) = SYM\_Dec_z(c) \)

Server Computes: \( y' = Eval_{evk}(h, Enc_{pk}(sym)) \)

\[ \Rightarrow y' = Enc(h(sym)) = Enc\left(SYM\_Dec_{sym}(c)\right) = Enc_{pk}(x) \]
Approximate Eigenvector Method [GSW13]

**Observation:** Let $C_1, C_2$ be matrices with the same eigenvector $\tilde{s}$, and let $m_1, m_2$ be their respective eigenvalues w.r.t $\tilde{s}$. Then:

1. $C_1 + C_2$ has eigenvalue $(m_1 + m_2)$ w.r.t $\tilde{s}$.
2. $C_1 \cdot C_2$ (and also $C_2 \cdot C_1$) has eigenvalue $m_1 m_2$ w.r.t $\tilde{s}$.

**Idea:** $\tilde{s} = \text{secret key}, C = \text{ciphertext},$ and $m = \text{message}.$

⇒ Homomorphism for addition and multiplication.

⇒ Full homomorphism!

Insecure! Eigenvectors are easy to find.

What about **approximate** eigenvectors?
Approximate Eigenvector Method [GSW13]

\[ C \cdot \hat{s} = m\hat{s} + \hat{e} \approx m\hat{s} \]

How to decrypt? Must have restriction on \( \|\hat{e}\| \)

Suppose \( \hat{s}[1] = \frac{q}{2} \), and \( m \in \{0,1\} \)

\[ (C \cdot \hat{s})[1] = \frac{q}{2} m + \hat{e}[1] \]

Find \( m \) by rounding

Condition for correct decryption: \( \|\hat{e}\| < \frac{q}{4} \).
Approximate Eigenvector Method [GSW13]

\[ C_1 \cdot \hat{s} = m_1 \hat{s} + \hat{e}_1 \]
\[ \|\hat{e}_1\| \ll q \]

\[ C_2 \cdot \hat{s} = m_2 \hat{s} + \hat{e}_2 \]
\[ \|\hat{e}_2\| \ll q \]

**Goal:** \( C_1, C_2 \Rightarrow C_{add} = Enc(m_1 + m_2), C_{mult} = Enc(m_1m_2). \)

\[ C_{add} = C_1 + C_2: \]
\[ (C_1 + C_2) \cdot \hat{s} = C_1 \hat{s} + C_2 \hat{s} \]
\[ = m_1 \hat{s} + \hat{e}_1 + m_2 \hat{s} + \hat{e}_2 \]
\[ = (m_1 + m_2) \hat{s} + (\hat{e}_1 + \hat{e}_2) \]

\[ \hat{e}_{add} \]

Noise grows a little
Approximate Eigenvector Method [GSW13]

\[
C_1 \cdot \hat{s} = m_1 \hat{s} + \hat{e}_1
\]
\[
\|\hat{e}_1\| \ll q
\]

\[
C_2 \cdot \hat{s} = m_2 \hat{s} + \hat{e}_2
\]
\[
\|\hat{e}_2\| \ll q
\]

**Goal:** \(C_1, C_2 \Rightarrow C_{add} = Enc(m_1 + m_2), C_{mult} = Enc(m_1 m_2).\)

\[
C_{mult} = C_1 \cdot C_2:
\]

\[
(C_1 \cdot C_2) \cdot \hat{s} = C_1(m_2 \hat{s} + \hat{e}_2)
\]
\[
= m_2 C_1 \hat{s} + C_1 \hat{e}_2
\]
\[
= m_2 (m_1 \hat{s} + \hat{e}_1) + C_1 \hat{e}_2
\]
\[
= m_2 m_1 \hat{s} + m_2 \hat{e}_1 + C_1 \hat{e}_2
\]

Can also use \(C_2 \cdot C_1\)

Noise grows. But by how much?
Plan for Technical Part

1. Constructing approximate eigenvector scheme.
2. Sequentialization.
4. Open problems and limits on FHE.
Learning with Errors (LWE) [R05]

Random noisy linear equations ≈ uniform

As hard as \((n/\alpha)\)-apx. short vector in **worst case** \(n\)-dim. lattices [R05, P09]
Encryption Scheme from LWE

[ACPS09]

\[
\begin{align*}
\mathbf{A} & \times \mathbf{s} + \mathbf{r} = \mathbf{b} \\
\mathbf{s} & \times \mathbf{r} \cdot \mathbf{\eta} + \mathbf{g} \times \mathbf{s} = \mathbf{\tilde{c}_g} \\
\mathbf{g} \cdot \mathbf{s} & \text{(without knowing } \mathbf{s}) \quad [\text{ACPS09}]
\end{align*}
\]
Encryption Scheme from LWE
[R05,ACPS09]

\[
\begin{align*}
A \cdot s + \eta &= b \\
\gamma = \eta + G \\n\gamma_{0,1} \times (n+1) &\text{uniform}
\end{align*}
\]
Approx. Eigenvector Encryption

**Goal:** Encrypt message $m \in \{0, 1\}$

**Idea:** $Enc(m) = C_m \cdot I$

$$\Rightarrow C_m \cdot \hat{s} = \hat{e} + mI \hat{s} = m \cdot \hat{s} + \hat{e}$$

As we saw:

$$C_1 \cdot C_2 \cdot \hat{s} = C_1 \cdot (\hat{e}_2 + m_2 \hat{s})$$

$$= C_1 \cdot \hat{e}_2 + m_2 \cdot C_1 \cdot \hat{s}$$

$$= C_1 \cdot \hat{e}_2 + m_2 \hat{e}_1 + m_1 m_2 \hat{s}$$

- **HUGE** noise
- **small** noise
- **desired** output

**Need to reduce the norm of $C_1$**

**Solution:** binary decomposition
Binary Decomposition

Break each entry in $C$ to its binary representation

$$C = \begin{bmatrix} 3 \\ 1 \\ 4 \\ 5 \end{bmatrix} \pmod{8} \implies \text{bits}(C) = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \pmod{8}$$

Small entries like we wanted!
But product with $\hat{s}$ now meaningless

Consider the “reverse” operation:

$$\text{bits}(C) \cdot \begin{bmatrix} 4 & 0 \\ 2 & 0 \\ 1 & 0 \\ 0 & 4 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} = C \implies C \cdot \hat{s} = \text{bits}(C) \cdot G \cdot \hat{s} = \text{bits}(C) \cdot \hat{s}^*$$

$$\hat{s}^* = G \cdot \hat{s}$$

“powers of 2” vector
Contains $q/2$ as an element
Approx. Eigenvector Encryption

\[ Enc(m) = C_{m \cdot G} \]
\[ \in \mathbb{Z}_q^{((n+1) \log q) \times (n+1)} \]

\[ C_{\text{nand}} = G - \text{bits}(C_1) \cdot C_2 \]

\[ \Rightarrow C_{m \cdot G} \cdot \tilde{s} = \tilde{e} + m \cdot G \cdot \tilde{s} \]

\[ C_{\text{mult}} = \text{bits}(C_1) \cdot C_2 \]

\[ \text{bits}(C_1) \cdot C_2 \cdot \tilde{s} \]
\[ = \text{bits}(C_1) \cdot (\tilde{e}_2 + m_2 G \tilde{s}) \]
\[ = \text{bits} (C_1) \cdot \tilde{e}_2 + m_2 \cdot \text{bits}(C_1) \cdot G \cdot \tilde{s} \]
\[ = \text{bits} (C_1) \cdot \tilde{e}_2 + m_2 \cdot C_1 \cdot \tilde{s} \]
\[ = \text{bits} (C_1) \cdot \tilde{e}_2 + m_2 \cdot \tilde{e}_1 + m_1 \cdot m_2 \cdot G \cdot \tilde{s} \]

\[ \| \tilde{e}_{\text{nand}} \| \leq N \cdot \| \tilde{e}_2 \| + m_2 \cdot \| \tilde{e}_1 \| \leq (N + 1) \cdot \max\{\| \tilde{e}_1 \|, \| \tilde{e}_2 \|\} \]
Homomorphic Circuit Evaluation

Noise grows during homomorphic evaluation

\[ \| \hat{e}_{\text{output}} \| \leq (N + 1)^d \cdot M\alpha q \approx N^d \alpha q \]

\[ \Rightarrow \text{Decryption succeeds if } \alpha \ll 1/N^d. \]

\[ \| \hat{e}_{i+1} \| \leq (N + 1) \| \hat{e}_i \| \]

\[ \| \hat{e}_{\text{input}} \| \leq M\alpha q \]
Full Homomorphism

\[ \alpha \leq N^{-d} \]
\[ d_{hom} \approx \log(1/\alpha) \]

1. If depth upper-bound is known ahead of time.
   
   Set \( N \geq d^2 \); \( \alpha = 2^{-\sqrt{N}} \) \( \Rightarrow \) \( \log(1/\alpha) = d \)

   Leveled FHE: Parameters (evk) grow with \( d \).

2. Single scheme for any poly depth.

Undesirable:
- Huge parameters.
- Low security.
- Inflexible.

Bootstrap!
The Bootstrapping Theorem

(Proof to come)

Homomorphic $\Rightarrow$ fully homomorphic

when $d_{dec} < d_{hom}$

- $d_{dec}$ = depth of the decryption circuit.
- $d_{hom}$ = maximal homomorphic depth.

In our scheme: $d_{dec} = \log N \Rightarrow \text{FHE if } \alpha < N^{-\log N}$

Quasi-polynomial approximation for short vector problems
(same factor as [BGV12,B12])

Non-homomorphic schemes only need $N^{O(1)}$ approximation
A Taste of Sequentialization [BV13]

\[ \hat{e}_{\text{mult}} = \text{bits}(C_1) \cdot \hat{e}_2 + m_2 \cdot \hat{e}_1 \]

Asymmetric!

Important observations:

1. \( \hat{e}_1 \) gets multiplied by 0/1; \( \hat{e}_2 \) can get multiplied by \( N \).
2. \( m_2 = 0 \Rightarrow \hat{e}_1 \) has no effect!

Conclusion: The order of multiplication matters.

Want to multiply \( C_A, C_B \) s.t. \( \hat{e}_A \gg \hat{e}_B \).

Which is better: \( \text{bits}(C_A) \cdot C_B \) or \( \text{bits}(C_B) \cdot C_A \)?
A Taste of Sequentialization [BV13]

\[
\hat{e}_{\text{mult}} = \text{bits } (C_1) \cdot \hat{e}_2 + m_2 \cdot \hat{e}_1
\]

**Task:** Multiply 4 ciphertexts \(C_1, \ldots, C_4\)

**Multiplication Tree**

\[
\|\hat{e}\| = E_0(N + 1)^2
\]

\[
\|\hat{e}\| = E_0(N + 1)
\]

\[
\|\hat{e}\| = E_0
\]

**Sequential Multiplier**

\[
E_0(3N + 1)
\]

\[
E_0(2N + 1)
\]

\[
E_0(N + 1)
\]

\[
E_0
\]

Winner!
Bootstrapping

Homomorphic $\Rightarrow$ fully homomorphic when

\[ d_{dec} < d_{hom} \]

- $d_{dec}$ = depth of the decryption circuit.
- $d_{hom}$ = maximal homomorphic depth.
Bootstrapping

Given scheme with bounded $d_{\text{hom}}$
How to extend its homomorphic capability?

Idea: Do a few operations, then "switch" to a new instance

Switch keys
"cost" in homomorphism
How to Switch Keys

We have seen this before!

Hybrid FHE
Hybrid FHE

\[
\text{Define: } h(z) = \text{SYM}_z( Dec_z(c) )
\]

Server Computes:
\[
y' = \text{Eval}_{evk}( h, \text{Enc}_{pk}(\text{sym}) )
\]
\[
\Rightarrow y' = \text{Enc}( h(\text{sym}) ) = \text{Enc} \left( \text{SYM} \_ \text{Dec}_{sym}(c) \right) = \text{Enc}_{pk}(x)
\]
How to Switch Keys

Decryption circuit:

\[ \text{Dec}_{sk}(\cdot) \]

\[ c \quad \Rightarrow \quad m \]

Dual view:

\[ \text{Dec}(\cdot)(c) \equiv h_c(\cdot) \]

\[ sk \quad \Rightarrow \quad m \]

\[ h_c(sk) = \text{Dec}_{sk}(c) = m \]

Key switching procedure \((sk_1, pk_1) \rightarrow (sk_2, pk_2)\):

**Input:** \( c = \text{Enc}_{pk_1}(m) \)

**Server aux info:** \( aux = \text{Enc}_{pk_2}(sk_1) \) (ahead of time)

**Output:** \( \text{Eval}_{pk_2}(h_c, aux) \)

\[ \text{Eval}_{pk_2}(h_c, aux) = \text{Eval}_{pk_2}(h_c, \text{Enc}_{pk_2}(sk_1)) \]

\[ = \text{Enc}_{pk_2}(h_c(sk_1)) = \text{Enc}_{pk_2}\left(\text{Dec}_{sk_1}(c)\right) \]

\[ = \text{Enc}_{pk_2}(m) \]

Eval depth = \( d_{dec} \)
Bootstrapping

Given scheme with bounded $d_{hom}$

How to extend its homomorphic capability?

**Idea:** Do a few operations, then “switch” to a new instance ($pk_2, sk_2$)

Switch keys

“cost” of $d_{dec}$ hom. operations

**Conclusion:** Bootstrapping if $d_{hom} \geq d_{dec} + 1$

Need to generate many keys...
Bootstrapping

Given scheme with bounded $d_{hom}$.
How to extend its homomorphic capability?

**Idea:** Do a few operations, then “switch” to a new instance.

**Server aux info:**
$$aux = Enc_{pk}(sk)$$

Switch from the key to itself!
Key switching works.
Circular Security

Is it secure to publish $aux = Enc_{pk}(sk)$

**Intuitively:** Yes, encryption hides the message.

**Formally:** Security does not extend.

What can we do about it?

**Option 1:** Assume it’s secure – no attack is known.

**Option 2:** Use a sequence of keys.

$\Rightarrow$ No. of keys proportional to computation depth (leveled FHE).

**Short keys without circular assumption?**

[BV11a]: Circular secure “somewhat” homomorphic scheme.
Diversity

- Other (older) schemes with similar properties [AD97, GGH97, R03, R05, …] ⇒ homomorphism
  But all are lattice based

  [B13]: Homomorphically “clean up” the noise ⇒ break security.
  ⇒ “Too much” homomorphism is a bad sign.
What We Saw Today

• Definition of FHE.
• Applications.
• Historical perspective and background.
• Constructing HE using the approximate eigenvector method.
• Sequentialization.
• Bootstrapping.
• Limits on HE.
Open Problems

• Short keys without circular security.
• FHE from different assumptions.
• CCA1 secure FHE.
• Bounded malleability.
• Improved efficiency.
Thank You