Learning With Errors in Homomorphic Encryption

Kim Laine

January 21, 2015

Structure of this talk

1. What is homomorphic encryption
2. Learning With Errors
3. Example: Brakerski-Vaikuntanathan scheme
4. Key recovery for LWE in polynomial time (L.-Lauter)

What is homomorphic encryption

(Fully) Homomorphic encryption: `Enc` and `Dec` are homomorphisms with respect to "addition" and "multiplication" between plaintext and ciphertext spaces.

- Very easy to have homomorphisms with respect to one operation, e.g. RSA, ElGamal (multiplication), Paillier (addition)
- RSA example is not very good to illustrate the difficulty: Should be indeterministic
- Existence was an open problem for decades; solved in 2010 by Craig Gentry
- Gentry's scheme was very complicated and extremely impractical
- Lot of improvement since 2010; research both in academia and industry
- **Every single scheme is based on computationally hard lattice problems**
- **Quantum secure as far as we know**

What is homomorphic encryption

Somewhat homomorphic encryption: Can perform an unspecified large number of homomorphic additions on ciphertexts but only k (predetermined number) of homomorphic multiplications.

- If more multiplications are performed, impossible to decrypt
- Larger k requires bigger keys (sizes are very significant!)
- Choose parameters based on function to be evaluated; schemes are very application specific
- Still slow; impractical to encrypt large amounts of data
- Current situation: Possible to come up with light applications (Microsoft Research, CryptoExperts, etc.)
- Secure data processing in the cloud: A dream

What is homomorphic encryption

What about FHE?

- SHE can be converted to FHE using Craig Gentry's **bootstrapping theorem**
- Unfortunately: Destroys performance; completely impractical
- Requires circular security: Is the scheme secure if adversary knows some bits of encrypted secret key?
- State-of-the-art: FHE completely out of reach; most research focuses on improving SHE

Hard problems for SHE/FHE:

- Learning With Errors (Oded Regev, 2005)
- NTRU type problems
- Approximate GCD

Learning With Errors: Basic idea

Can you solve $\mathbf{s}$ from the following system?

$$\begin{cases} \langle \mathbf{a}_0, \mathbf{s} \rangle \approx b_0 \pmod{q} \\ \langle \mathbf{a}_1, \mathbf{s} \rangle \approx b_1 \pmod{q} \\ \langle \mathbf{a}_2, \mathbf{s} \rangle \approx b_2 \pmod{q} \\ \quad \vdots \qquad\qquad \vdots \\ \langle \mathbf{a}_{d-1}, \mathbf{s} \rangle \approx b_{d-1} \pmod{q} \end{cases}$$

This seems to be very hard. No hope using Gaussian elimination because errors blow up.

Learning With Errors: Definitions

Learning With Errors (Regev, 2005) is a hard lattice problem that several (homomorphic) cryptosystems are based on.

$q = 2^r$ an integer modulus

n an integer, $\mathbf{s} \in \mathbb{Z}_q^n$ a secret vector chosen uniformly at random

$D_{\mathbb{Z},\sigma}$ (**error distribution**) the discrete Gaussian distribution centered at 0, with standard deviation σ:

$$\Pr(x) \propto \exp\left(-\frac{x^2}{2\sigma^2}\right) .$$

Learning With Errors: Definitions

Definition 1 (LWE sample)

An LWE sample is a pair $(\mathbf{a}, t) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $\mathbf{a}$ is sampled uniformly at random from $\mathbb{Z}_q^n$, $e \leftarrow D_{\mathbb{Z},\sigma}$ and $t = \left[\langle \mathbf{a}, \mathbf{s} \rangle + e\right]_q = \langle \mathbf{a}, \mathbf{s} \rangle_q + e \in (-q/2, q/2)$.

Definition 2 (decision-LWE$_{n,r,d,\sigma}$)

Given d samples $(\mathbf{a}_i, t_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, the problem decision-LWE$_{n,r,d,\sigma}$ is to distinguish with non-negligible advantage whether these are LWE samples or sampled uniformly at random from the set $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Definition 3 (search-LWE$_{n,r,d,\sigma}$)

Given d LWE samples $(\mathbf{a}_i, t_i)$, the problem search-LWE$_{n,r,d,\sigma}$ is to recover the secret vector $\mathbf{s}$.

Learning With Errors: Decision-to-Search reduction

Decision-to-Search reduction (Regev): If decision-LWE can be solved with nearly perfect advantage, then search-LWE can be solved.

1. Suppose we are given one sample $(\mathbf{a}, t)$.
2. For some integer $k \in \mathbb{Z}_q$, form the sample $(\mathbf{a}', t') := (\mathbf{a} + (k, 0, \ldots, 0), t + k\mathbf{a}[0])$.
3. Use the solution to decision-LWE to decide whether $(\mathbf{a}', t')$ is an LWE sample or not.
4. If it is, then $\mathbf{s}[0] = k$. If not, try another k.
5. Similarly find all $\mathbf{s}[j]$.
6. Need to try at most $n \cdot q$ times to recover $\mathbf{s}$.

Possible if q is small, but distinguishing advantage must be extremely good.

Learning With Errors: GapSVP

GapSVP: A hard decisional lattice problem

- Λ a lattice of dimension n
- $\gamma = \gamma(n)$ a function of n (the gap); d a number
- $\text{GapSVP}_{(\gamma,d)}$ is the problem of deciding whether $\lambda_1(\Lambda)$ (shortest vector in Λ) has length less than d or greater than $\gamma \cdot d$.

If $\gamma = \text{poly}(n)$: Very hard problem; only exponential time solutions known

If $\gamma = \exp(n)$: Possibly easy; might be solvable in polynomial time using LLL

Huge open problem in theoretical quantum computing:

- Find a poly time quantum algorithm to solve GapSVP/SVP/approx-SVP

Learning With Errors: Security reductions

- Search-LWE: When $q = \text{poly}(n)$, σ large enough, polynomial time *quantum reduction* from worst-case GapSVP (Regev)
- Search-LWE: When $q = \text{poly}(n)$, σ large enough, polynomial time *classical reduction* from worst-case of an easier (less studied) variant of GapSVP (Peikert)
- Decision-LWE: When q is exponential in n, σ large enough, polynomial time *classical reduction* from worst-case GapSVP (Peikert)

In all cases the approximating factor is $\widetilde{O}(nq/\sigma)$.

Sadly in practice the parameters are never such that these reductions would apply.

Expect security problems when σ small, q large

Example: Brakerski-Vaikuntanathan scheme

Key generation:

- Choose a secret $\mathbf{s} \in \mathbb{Z}_q^n$.
- Choose a matrix $\mathbf{A} \in \mathbb{Z}_q^{d \times n}$ (each row corresponds to an $\mathbf{a}$ in our LWE samples).
- Sample an error vector $\mathbf{e}$ from $D_{\mathbb{Z},\sigma}^d$.
- Compute $\mathbf{b} := \mathbf{As} + 2\mathbf{e}$.
- Public key: $(\mathbf{A}, \mathbf{b})$
- Private key: $\mathbf{s}$

Example: Brakerski-Vaikuntanathan scheme

To encrypt:

- $m \in \mathbb{Z}_2$ the message bit
- Choose random $\mathbf{r}$ from $\{0,1\}^d$.
- Ciphertext: $c := (\mathbf{A}^\top \mathbf{r}, \langle \mathbf{b}, \mathbf{r} \rangle + m)$

To decrypt:

- Ciphertext: $(\mathbf{v}, t)$
- Compute

$$\left[t - \langle \mathbf{v}, \mathbf{s} \rangle \pmod q\right] \pmod 2 = \left[\langle \mathbf{b}, \mathbf{r} \rangle + m - \langle \mathbf{r}, \mathbf{As} \rangle \pmod q\right] \pmod 2$$

$$= \left[m + 2\langle \mathbf{r}, \mathbf{e} \rangle \pmod q\right] \pmod 2 = m$$

when $\mathbf{e}$ has small enough entries.

If the attacker can break LWE they can recover the secret $\mathbf{s}$ from the public key.

Example: (Simplified) Brakerski-Vaikuntanathan scheme

Additive homomorphism (very easy):

- $\texttt{Enc}(m_1) = (\mathbf{v}_1, t_1)$ and $\texttt{Enc}(m_2) = (\mathbf{v}_2, t_2)$
- $\texttt{Enc}(m_1 + m_2 \pmod 2) = (\mathbf{v}_1 + \mathbf{v}_2, t_1 + t_2)$ (error grows additively)
- Decryption succeeds as long as the sum of the errors does not cause reduction modulo q.

Multiplicative homomorphism (much more complicated):

- No time to discuss this in detail.
- Error grows multiplicatively.
- Leveled fully homomorphic encryption scheme: Must decide beforehand how many multiplications will be done (the level)
- Gentry's bootstrapping theorem: Can be converted into a fully homomorphic scheme but quite difficult

Polynomial time key recovery: appr-CVP

LWE as an instance of the lattice problem appr-CVP:

- appr-CVP = approximate Closest Vector Problem
- This means: Given a point $\mathbf{u}$ in $\mathbb{R}^D$, find a lattice point $\mathbf{v} \in \Lambda$ such that

$$||\mathbf{v} - \mathbf{u}|| \leq \gamma \cdot \min_{\mathbf{w} \in \Lambda} ||\mathbf{w} - \mathbf{u}|| .$$

- Easy in low dimensions (can possibly even enumerate)
- If a "good" (short, orthogonal) basis for Λ is known, can use Babai's nearest planes algorithm
- If no "good" basis is known, need to first find one using lattice reduction
- LLL: Polynomial time; approximation factor guaranteed $2^{D/2}$ (exponentially bad closest vector)
- BKZ-2.0: Exponential time; approximation factor poly(D) (can be arbitrarily good but very expensive)

Polynomial time key recovery: Recall the definitions

Definition 4 (LWE sample)

An LWE sample is a pair $(\mathbf{a}, t) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $\mathbf{a}$ is sampled uniformly at random from $\mathbb{Z}_q^n$, $e \leftarrow D_{\mathbb{Z},\sigma}$ and $t = \left[\langle \mathbf{a}, \mathbf{s} \rangle + e\right]_q = \langle \mathbf{a}, \mathbf{s} \rangle_q + e \in (-q/2, q/2)$.

Definition 5 (search-LWE$_{n,r,d,\sigma}$)

Given d LWE samples $(\mathbf{a}_i, t_i)$, the problem search-LWE$_{n,r,d,\sigma}$ is to recover the secret vector $\mathbf{s}$.

Polynomial time key recovery: LWE as appr-CVP

Let Λ be the $(n+d)$-dimensional lattice generated by the rows of the matrix

$$\begin{pmatrix} q & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & q & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & q & 0 & 0 & \cdots & 0 \\ \mathbf{a}_0[0] & \mathbf{a}_1[0] & \cdots & \mathbf{a}_{d-1}[0] & 1/2^N & 0 & \cdots & 0 \\ \mathbf{a}_0[1] & \mathbf{a}_1[1] & \cdots & \mathbf{a}_{d-1}[1] & 0 & 1/2^N & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \mathbf{a}_0[n-1] & \mathbf{a}_1[n-1] & \cdots & \mathbf{a}_{d-1}[n-1] & 0 & 0 & \cdots & 1/2^N \end{pmatrix}.$$

Easy to see:

$$\mathbf{v} = \left[\langle \mathbf{a}_0, \mathbf{s} \rangle_q, \langle \mathbf{a}_1, \mathbf{s} \rangle_q, \ldots, \langle \mathbf{a}_{d-1}, \mathbf{s} \rangle_q, \mathbf{s}[0]/2^N, \mathbf{s}[1]/2^N, \ldots, \mathbf{s}[n-1]/2^N\right] \in \Lambda$$

$$\mathbf{u} = \left[t_0, t_1, \ldots, t_{d-1}, 0, \ldots, 0\right] \notin \Lambda \text{ but is close to } \mathbf{v} \text{ if } N \text{ is big}$$

Polynomial time key recovery: LWE as appr-CVP

Recovering the secret key s:

- Solve good enough appr-CVP with **u** as input to recover **v**
- Read **s** from **v**
- Suppose we have access to any number of LWE samples: d can be anything we want
- Difficulty I: Lattice has very high dimension
- Difficulty II: Naive analysis indicates that a very good basis for Λ is needed (must use BKZ-2.0)

Polynomial time key recovery: New observation

Very surprising claim (L.-Lauter): If q is large enough and d is chosen appropriately, need to solve CVP only up to exponential factor (in n)!

- So in certain cases: Run LLL (polynomial time in n) and use Babai's method to find candidate for $\mathbf{v}$.
- Recover the correct vector $\mathbf{v}$ almost certainly

Explanation:

- If $q = 2^{O(n)}$ and d chosen appropriately, probability of having a lattice vector within some large radius $< q = 2^{O(n)}$ of $\mathbf{u}$ can be rigged to be very small.
- But we know there is one, namely $\mathbf{v}$.
- So even if LLL-Babai performs exponentially poorly, it will still find a close vector within this radius, which is very likely to be $\mathbf{v}$.

Polynomial time key recovery: New observation

Slightly more precise claim:

Suppose $r := \log_2 q > 7n/2$. Let

$$d = \left\lceil \sqrt{(r+1)n/2} \right\rfloor .$$

Solve ℓ_σ from

$$nd\sqrt{e}\, 2^{r-\ell_\sigma} = \sigma \exp\left(\frac{2^{2r-2\ell_\sigma-1}}{\sigma^2} \right) .$$

If

$$n/2 + 2\sqrt{(r+1)n/2} < \ell_\sigma ,$$

i.e. σ is small enough, then *search*-LWE$_{n,r,d,D_{\mathbb{Z},\sigma}}$ can be solved in probabilistic polynomial time in n by computing an LLL-reduced basis of the given basis of Λ and using Babai's nearest planes method to $\mathbf{u}$. The algorithm successfully returns $\mathbf{v}$ with very high probability. The running time is polynomial in n.

Polynomial time key recovery: Practical performance

Remarks:

- Idea of proof: Generalize Boneh-Venkatesan approach for solving Hidden Number Problem to higher dimensions
- In the proof many very strong approximations are made to guarantee success.
- Theorem does not tell much about practical performance.
- Input basis to LLL has very special form: Performance of LLL-Babai is very unclear
- How to measure practical performance and extrapolate results to bigger examples?

Polynomial time key recovery: Practical performance

Here is one way of measuring practical performance:

- Guaranteed approx factor in LLL-Babai is $2^{(n+d)/2}$ (used in proof)
- Instead write $2^{\mu(n+d)}$ and go through the proof; get a formula for value of μ needed to succeed.
- Run examples and compute the required μ ...

  Failed: effectively μ was larger

  Succeeded: effectively μ was smaller
- Gives an idea of how big μ can effectively be expected to be
- Extrapolate behavior of μ to larger examples and predict whether they succeed or fail

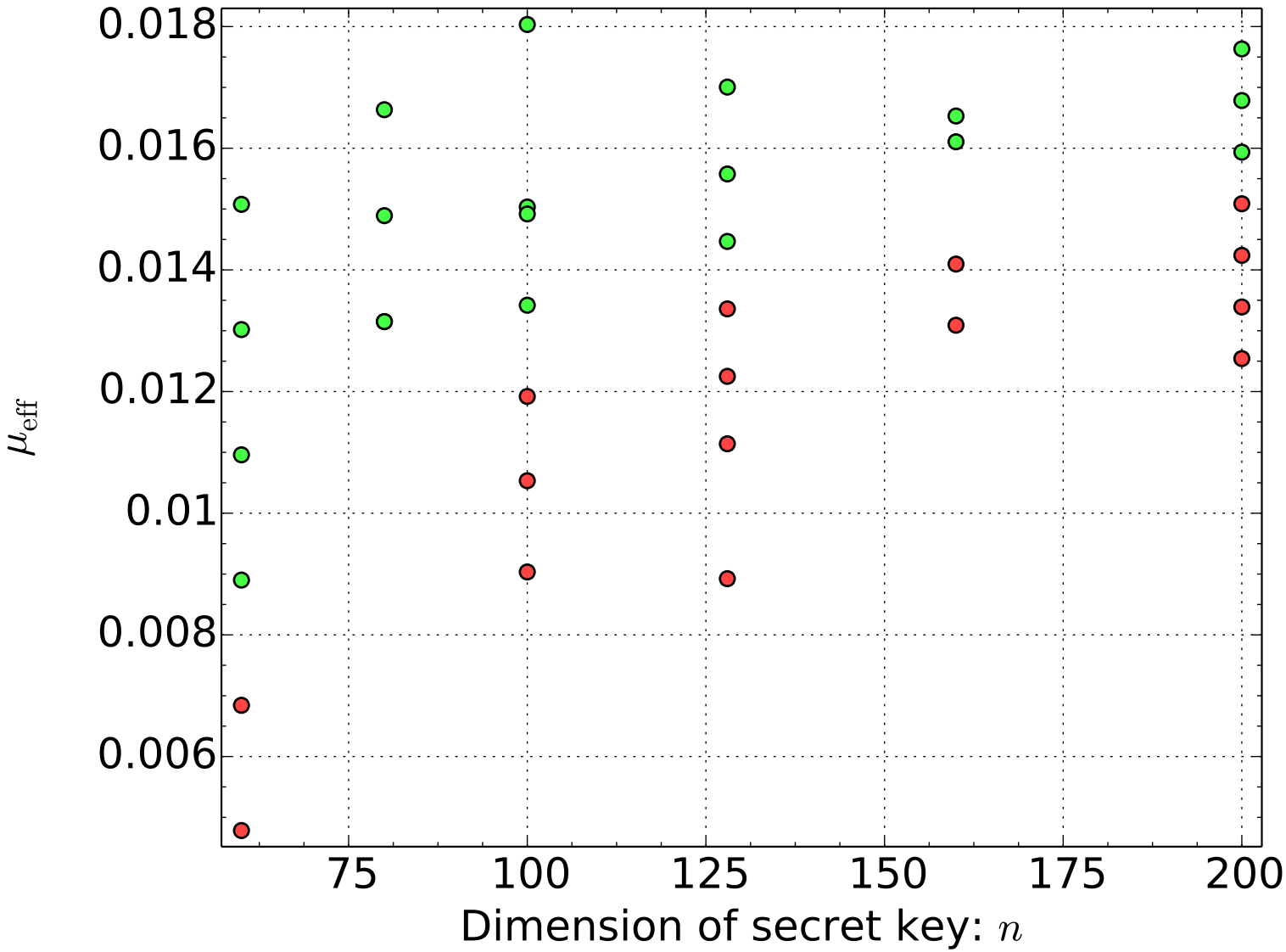
Polynomial time key recovery: Experimental results

For a particular experiment to succeed, the effective lower bound for μ that needs to be realized is

$$\mu_{\text{eff}} := \frac{1}{n+d}\left[-\frac{rn}{d} + r - 1 - \log_2(\lceil\sigma\rceil\sqrt{n+d})\right]$$

Here are the results as a function of n:

Polynomial time key recovery: Experimental results



Polynomial time key recovery: Experimental results

- There seems to be a trend, but exactly what?
- Could plausibly be used to predict what happens in big examples.
- Theoretical understanding of performance seems to be surprisingly non-trivial.
- More theoretical work and experimental results needed.

Security implications

Ok, so what can be broken? Here are some examples:

n	d	r	σ	Time
80	320	16	3.233	1.6h
100	400	20	6.346	6.8h
128	572	23	3.097	24h
160	540	27	3.077	1d 5h
200	550	31	3.049	2d 13h

Security implications

What happens for larger n? Here $\sigma = 8/\sqrt{2\pi}$.

n	d	r	μ_{eff}
512	1388	65	0.0171
1024	2576	120	0.0176
2048	5152	235	0.0184
4096	10104	465	0.0188

Will these succeed or fail?

Distinguishing attack

- Usually (e.g. Lindner-Peikert, van de Pol-Smart, Lepoint-Naehrig) recommended parameters based on hardness of a well-known probabilistic attack against decision-LWE (distinguishing of valid LWE samples from random data).
- Turns out: For the kinds of parameters that we can break, distinguishing is not hard.
- Search-to-Decision reduction!
- So none of this is surprising: *Distinguishing implies key recovery* but in time proportional to q (huge!).
- The interesting result is that key recovery can actually be done so easily in these cases, in polynomial time.

Security implications?

Therefore:

- At least with pure LLL the attack does not threaten commonly recommended secure parameters, but how significantly does performance improve if we improve the basis using other methods?
- In some special applications might want to evaluate very deep circuits homomorphically and need very large r, small σ.
- E.g. evaluating some block ciphers homomorphically.
- In these cases you really do need a large enough n.

Security of LWE based cryptosystems depends very strongly on the parameters.

Thank you!