MIT 6.875

Foundations of Cryptography
Lecture 10

Lectures 7-10

Constructions of Public-key Encryption

✅ Diffie-Hellman/El Gamal

✅ Trapdoor Permutations (RSA)

✅ Quadratic Residuosity/Goldwasser-Micali

4: Post-Quantum Security & Lattice-based Encryption

Why Lattice-based Crypto?

☐ **Exponentially Hard** (so far)

While factoring and discrete log can be solved in time $2^{\sqrt[3]{n}}$ for problems of size n, the best algorithms for lattice-based crypto run in time nearly 2^n.

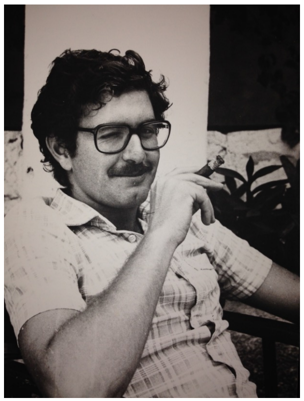
Why Lattice-based Crypto?

- ☐ **Exponentially Hard** (so far)
- ☐ **Quantum-Resistant** (so far)

(Very large scale) (if they exist)

Quantum Computers Break Crypto

Shor's Algorithm for Factoring and Discrete Logarithms.



“Cryptographers seldom sleep well”.

[Silvio Micali, 1988]

Post-Quantum Cryptography

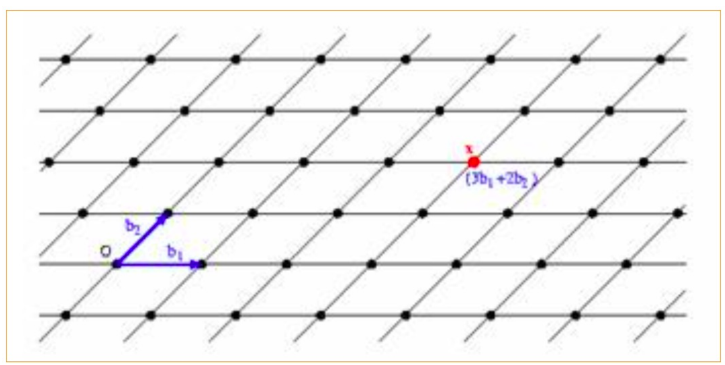
Cryptography that is (believed to be) secure against quantum attacks.

NEWS

NIST Announces First Four Quantum-Resistant Cryptographic Algorithms

Federal agency reveals the first group of winners from its six-year competition.

July 05, 2022



3 out of 4: Lattice-based Cryptography

Why Lattice-based Crypto?

- ☐ **Exponentially Hard** (so far)
- ☐ **Quantum-Resistant** (so far)
- ☐ **Worst-case hardness**

  (unique feature of lattice-based crypto)
- ☐ **Simple and Efficient**
- ☐ **Enabler of Surprising Capabilities**

  (Fully Homomorphic Encryption)

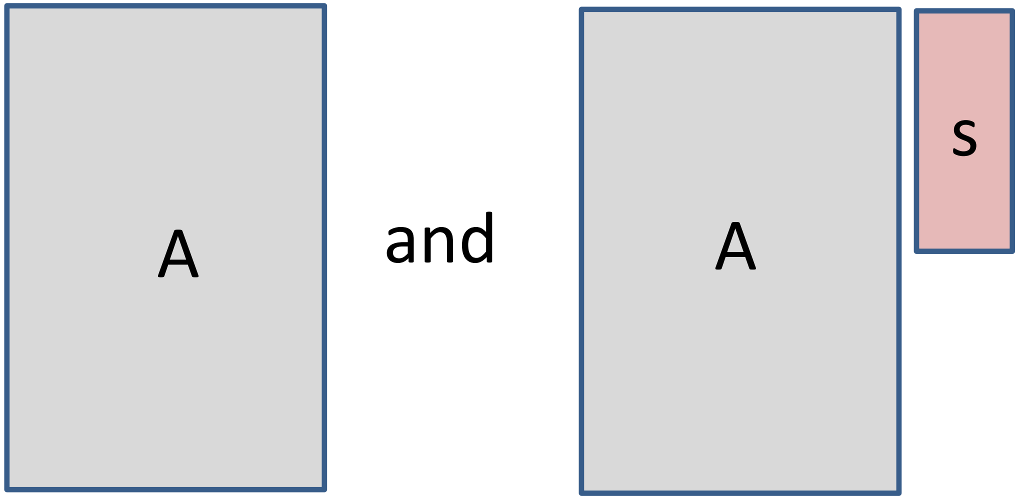
Solving Linear Equations

$$\begin{aligned} 5s_1 + 11s_2 &= 2 \\ 2s_1 + s_2 &= 6 \\ 7s_1 + s_2 &= 26 \end{aligned}$$

where all equations are over $\mathbb{Z}$, the integers

Solving Linear Equations

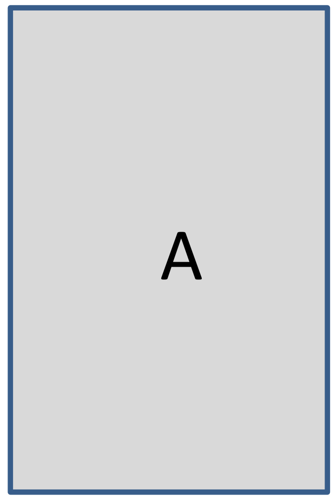
Given:

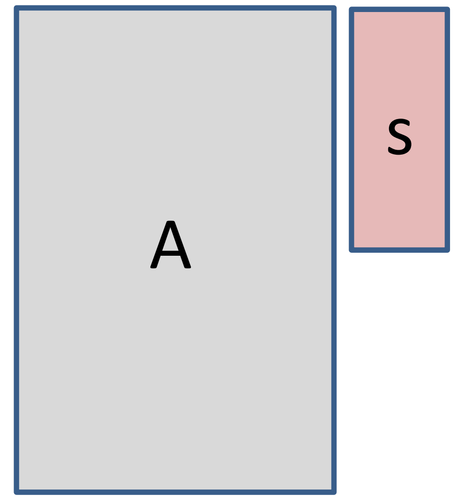


GOAL: Find s.

More generally, n variables and $m \gg n$ equations.

Solving Linear Equations

Given: 

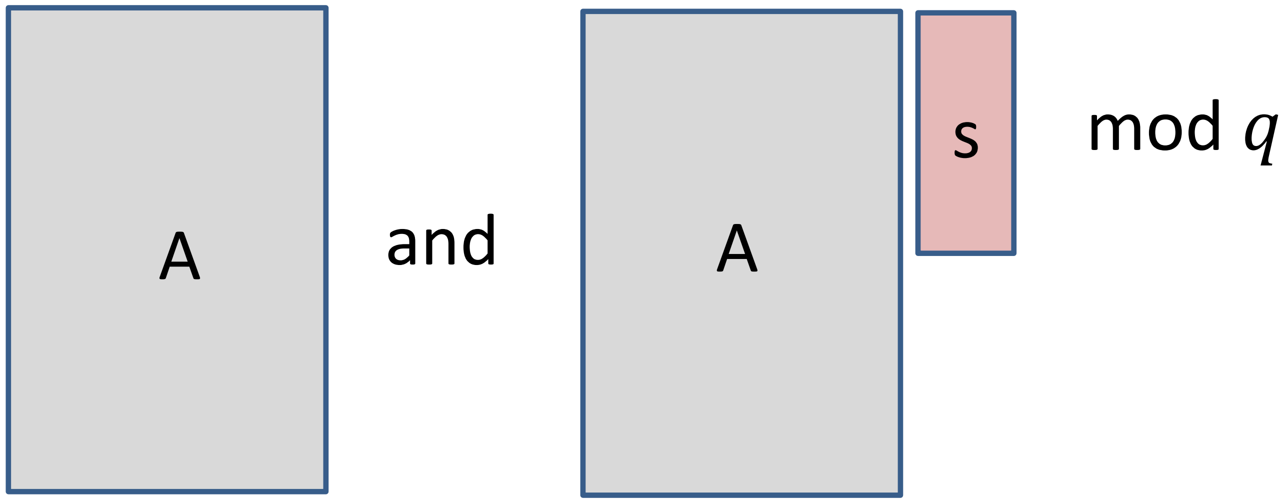
and 

GOAL: Find s.

EASY! For example, by Gaussian Elimination 

Solving Linear Equations

Given:



GOAL: Find s.

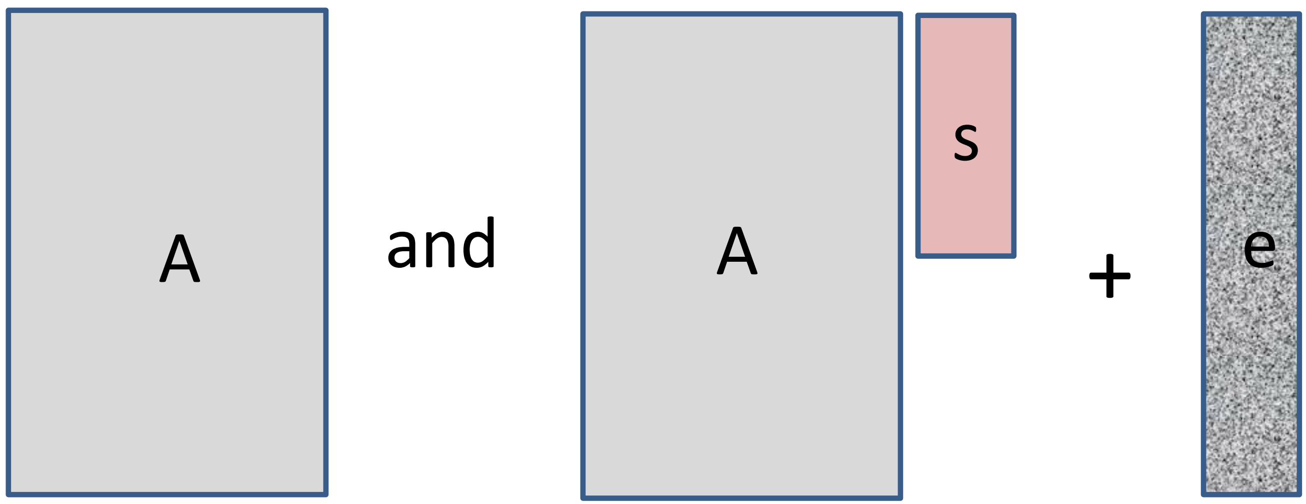
How to make it hard: Chop the head?

That is, work modulo some q. $(1121 \; mod \; 100 = 21)$

Still EASY! Gaussian Elimination mod q

Solving Linear Equations

Given:



GOAL: Find s.

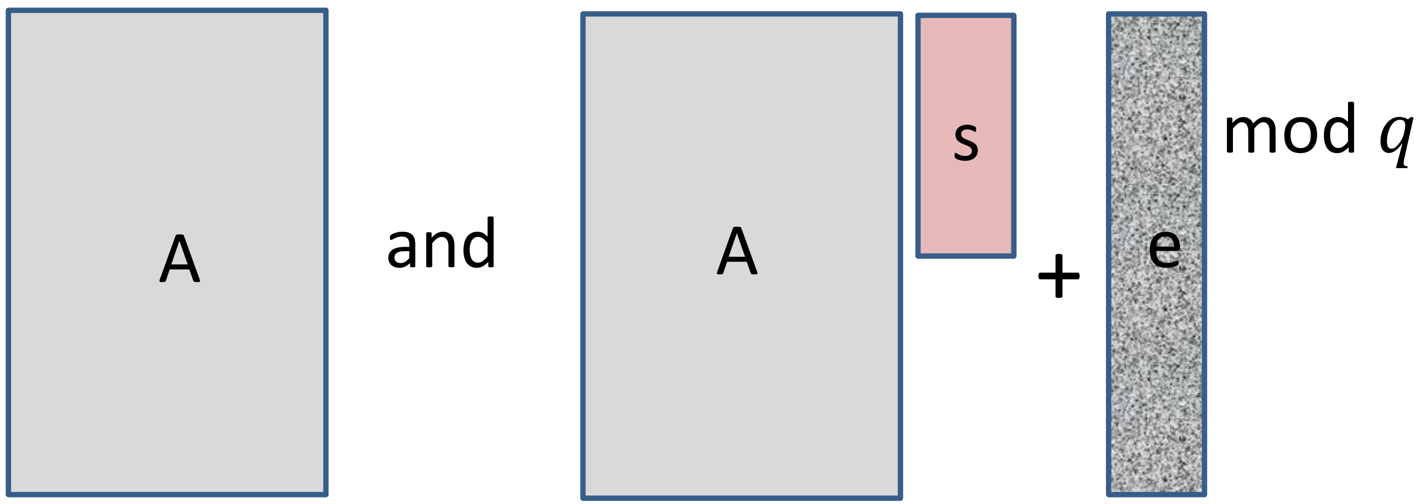
How to make it hard: Chop the tail?

Add a small error to each equation.

Still EASY! Linear regression.

Solving Linear Equations

Given:



GOAL: Find s.

How to make it hard: Chop the head *and* the tail?

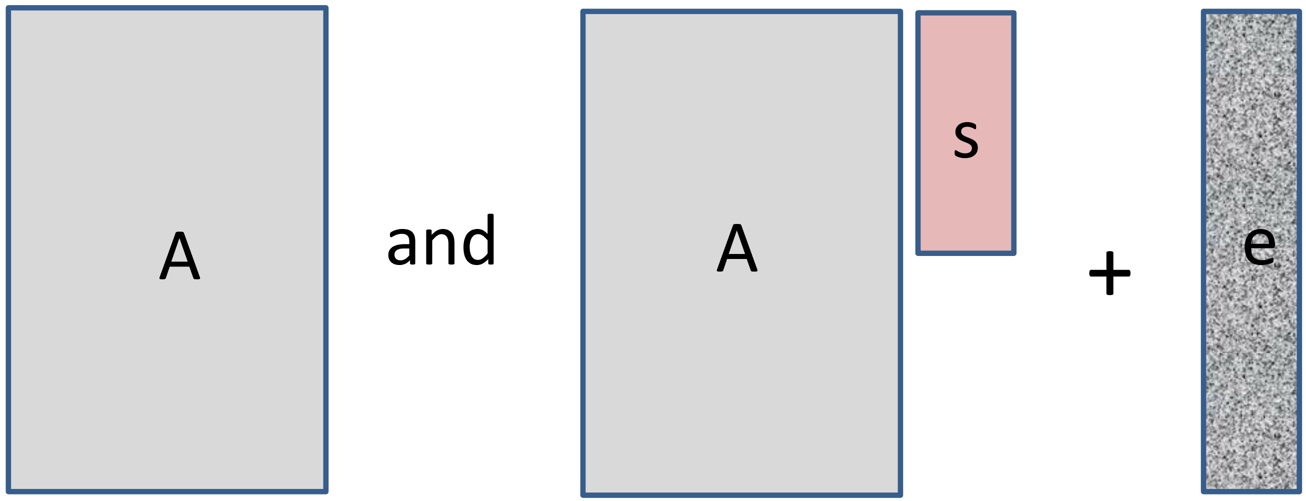
Add a small error to each equation and work mod q.

Turns out to be very HARD!

Learning with Errors (LWE)

Solving Noisy Modular Linear Equations

Given:



GOAL: Find s.

<u>Parameters</u>: dimensions $\boldsymbol{n}$ and m, modulus $\boldsymbol{q}$, error distribution χ = uniform in some interval $[-\boldsymbol{B}, \ldots, \boldsymbol{B}]$.

A is chosen at random from $\mathbb{Z}_q^{m\times n}$, **s** from $\mathbb{Z}_q^n$ and **e** from χ^m.

Learning with Errors (LWE)

=

- **Decoding Random Linear Codes**
  (over $\mathbb{Z}_q$ with ℓ_∞ errors)

- **Learning Noisy Linear Functions**

- **Worst-case hard Lattice Problems**
  [Regev'05, Peikert'09]

Attack 1: *Linearization*

<u>Given $\boldsymbol{A}, \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$, find $\boldsymbol{s}$</u>.

Idea (a) *Each noisy linear equation is an exact polynomial eqn.*

Consider $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i + e$.

Imagine for now that the error bound $B = 1$. So, $e \in \{-1,0,1\}$. In other words, $\mathrm{b} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i \in \{-1,0,1\}$.

So, here is a noiseless polynomial equation on s_i:

$$(\mathrm{b} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i - 1)\ (\mathrm{b} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i)(\mathrm{b} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i + 1) = 0$$

Attack 1: *Linearization*

<u>Given $\boldsymbol{A}, \boldsymbol{As} + \boldsymbol{e}$, find $\boldsymbol{s}$</u>.

BUT: *Solving (even degree 2) polynomial equations is NP-hard.*

$$(\mathrm{b} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i - 1)\,(\mathrm{b} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i)(\mathrm{b} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i + 1) = 0$$

Attack 1: *Linearization*

$$(\mathrm{b} - \textstyle\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i - 1)\,(\mathrm{b} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i)(\mathrm{b} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} a_i s_i + 1) = 0$$

Idea (b) *Easy to solve given* <u>*sufficiently many*</u> *equations.*

(using a technique called '

$$\sum a_{ijk} s_i s_j s_k + \sum a_{ij} s_i s_j + \sum a_i$$

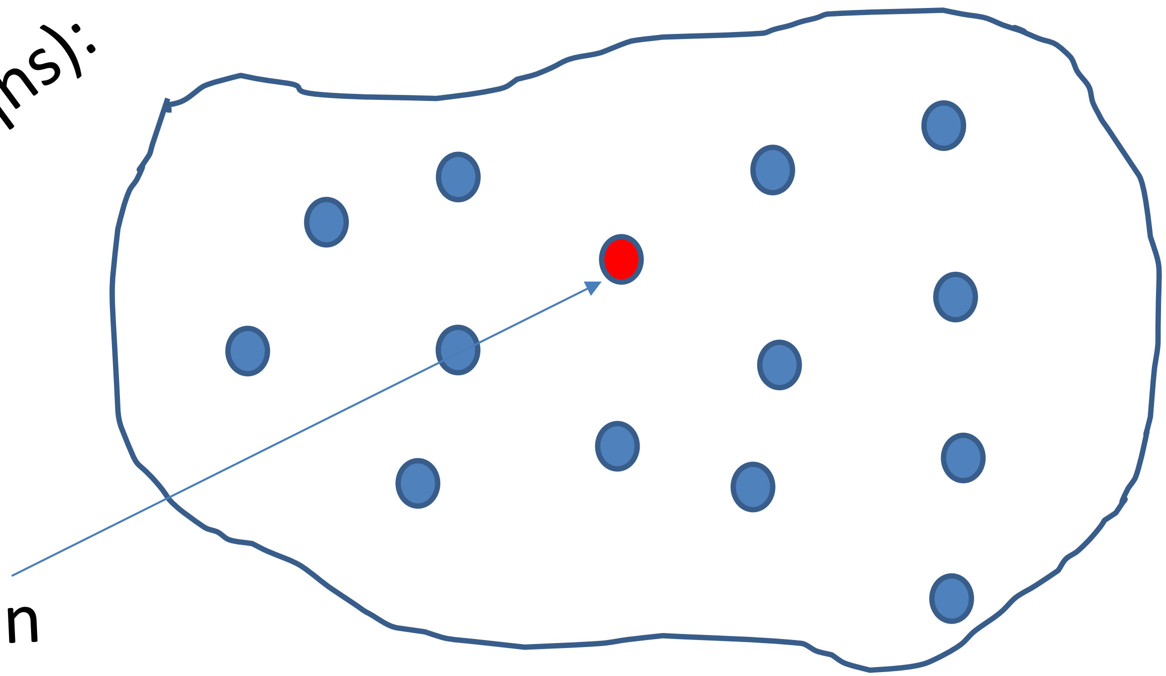
Treat each "monomial", e.g. $\mathrm{s_i s}$
variable, e.g. $\mathrm{t_{ijk}}$.

Now, you have a noiseless linear equation in $\mathrm{t_{ijk}}$!!!

Attack 1: *Linearization*

$$\sum a_{ijk} t_{ijk} + \sum a_{ij} t_{ij} + \sum a_i t_i + (b-1)b(b+1) = 0$$

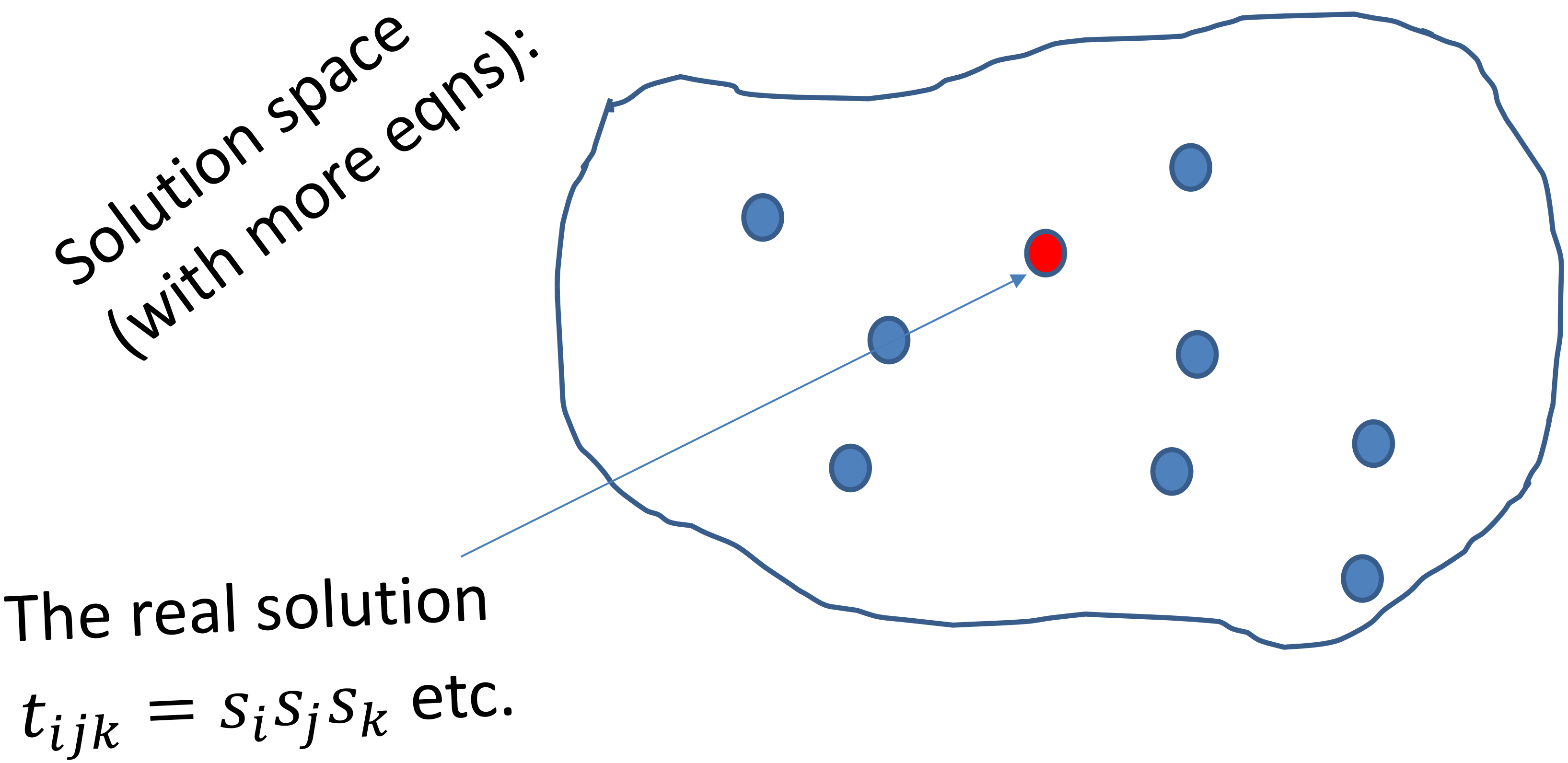
Solution space
(with some eqns):



The real solution
$t_{ijk} = s_i s_j s_k$ etc.

Attack 1: *Linearization*

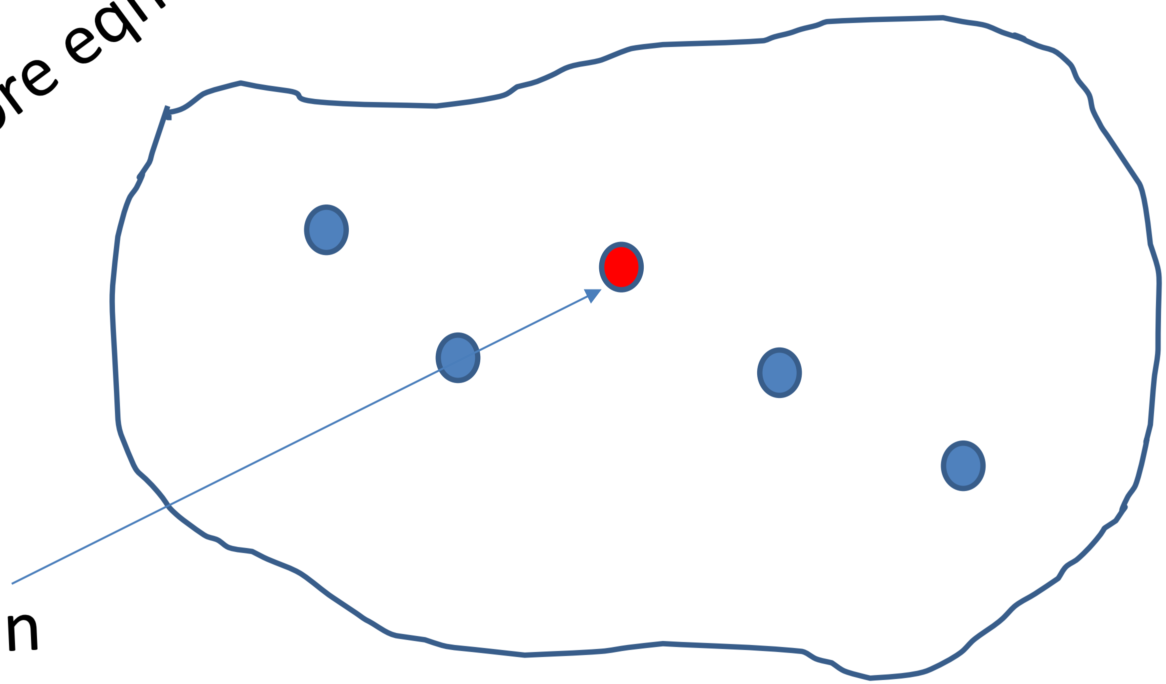
$$\sum a_{ijk}t_{ijk} + \sum a_{ij}t_{ij} + \sum a_i t_i + (b-1)b(b+1) = 0$$



Attack 1: *Linearization*

$$\sum a_{ijk} t_{ijk} + \sum a_{ij} t_{ij} + \sum a_i t_i + (b-1)b(b+1) = 0$$

Solution space
(with even more eqns):

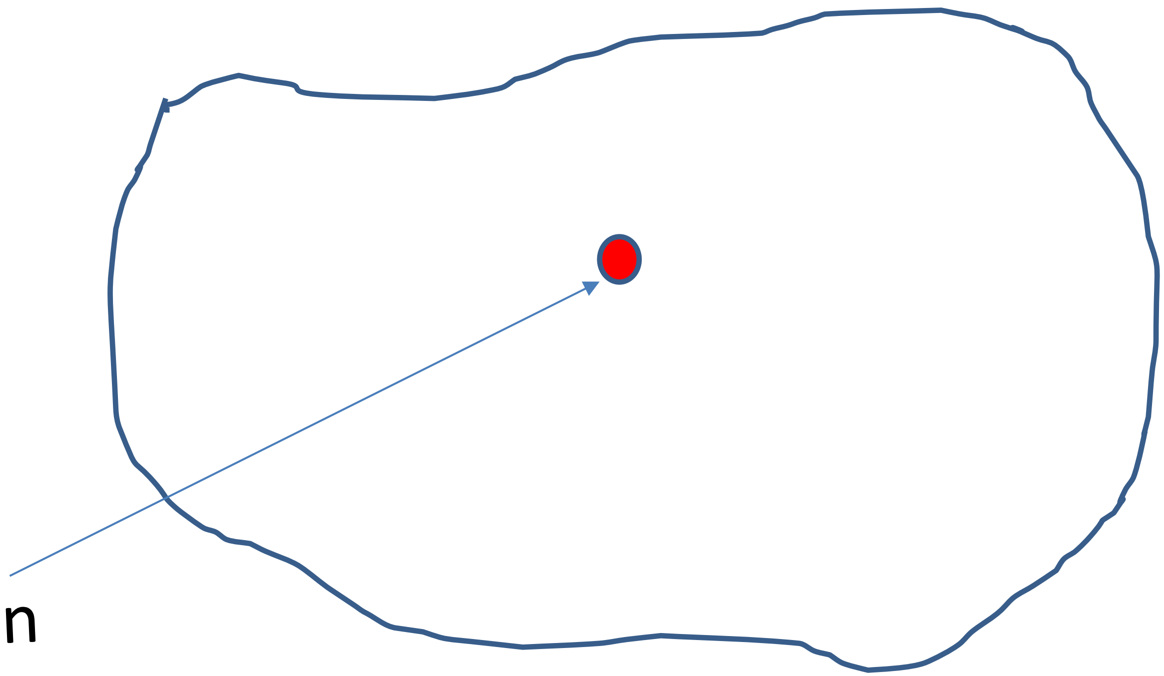


The real solution
$t_{ijk} = s_i s_j s_k$ etc.

Attack 1: *Linearization*

$$\sum a_{ijk}t_{ijk} + \sum a_{ij}t_{ij} + \sum a_i t_i + (b-1)b(b+1) = 0$$

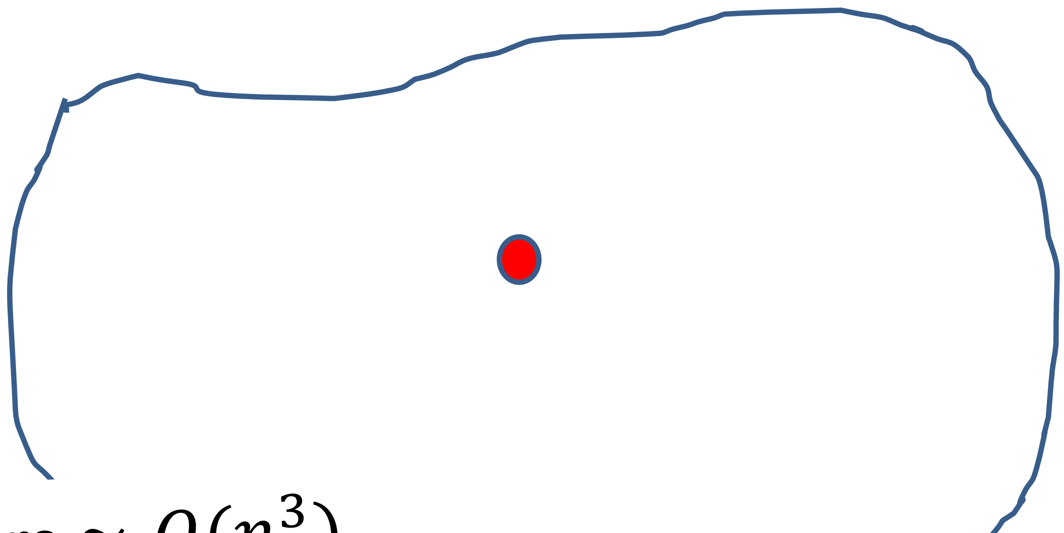
Solution space
(keep going):



The real solution
$t_{ijk} = s_i s_j s_k$ etc.

Attack 1: *Linearization*

$$\sum a_{ijk} t_{ijk} + \sum a_{ij} t_{ij} + \sum a_i t_i + (b-1)b(b+1) = 0$$



When #eqns = #vars $\approx O(n^3)$
the only surviving solution to the linear system is the real solution.

Attack 1: *Linearization*

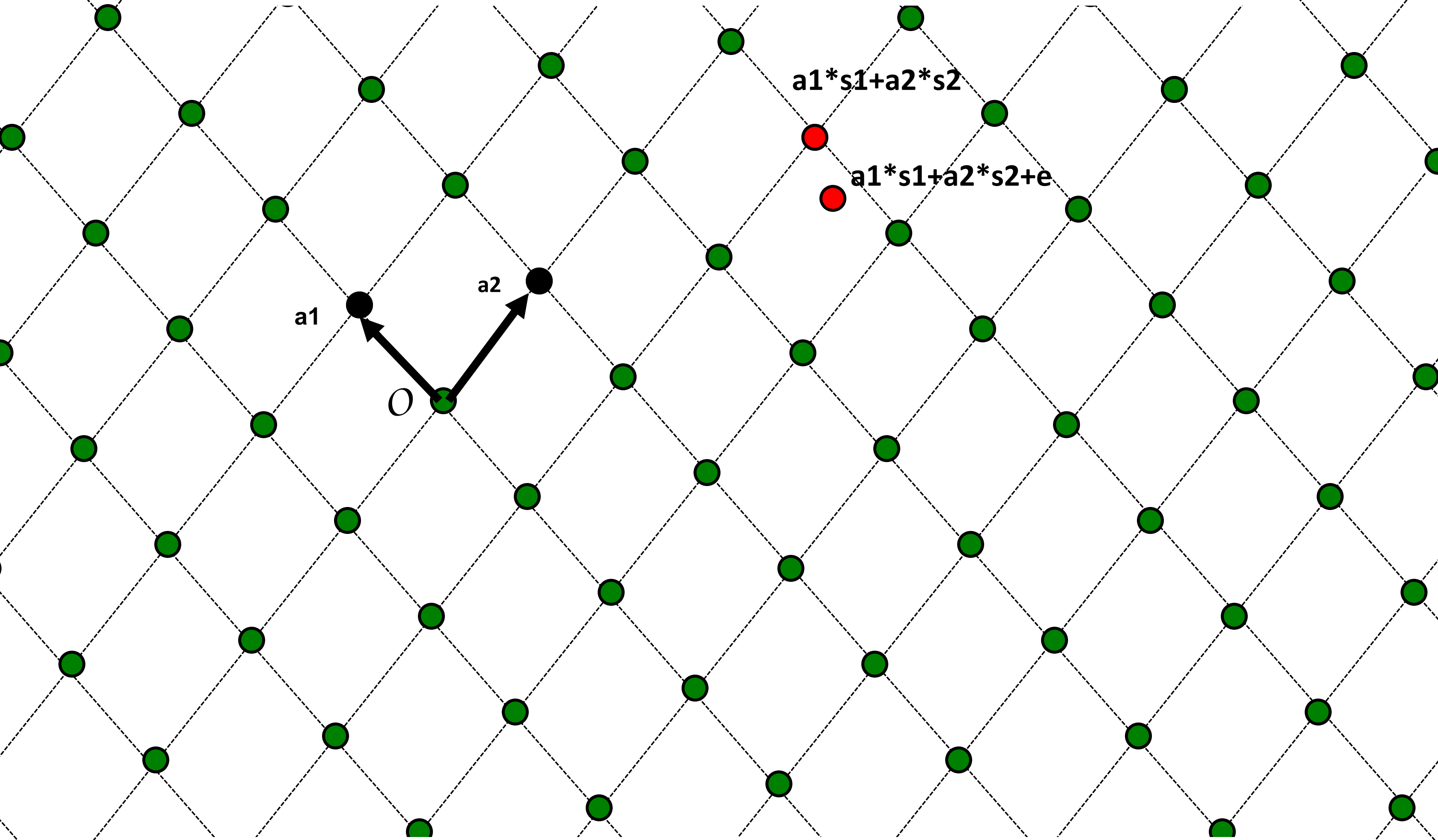
Given $\boldsymbol{A}, \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$, find $\boldsymbol{s}$.

Can solve/break as long as

$$\boldsymbol{m} \gg \boldsymbol{n}^{\boldsymbol{2B+1}}$$

We will set $B = n^{\Omega(1)}$, in other words polynomial in n so as to blunt this attack.

LWE and Lattices



A lattice is a discrete, additive subgroup of $\mathbb{R}^m$

Attack 2: *Lattice Reduction*

Lenstra-Lenstra-Lovasz (LLL) Algorithm

Say $q/B = 2^{n^{\varepsilon}}$ for a constant $\varepsilon > 0$. LLL solves LWE in time $2^{\tilde{O}(n^{1-\varepsilon})} \cdot \mathrm{poly}(n, \log q)$.

This is polynomial in n and $\log q$ when $\frac{q}{B} = 2^{\Omega(n)}$.

Setting Parameters

Cryptanalysis over three decades suggests we are safe with the following parameters:

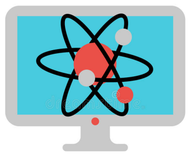
$n =$ security parameter ($\approx 1 - 10\text{K}$)

$m =$ arbitrary poly in n

$B =$ small poly in n, say $\sqrt{n}$

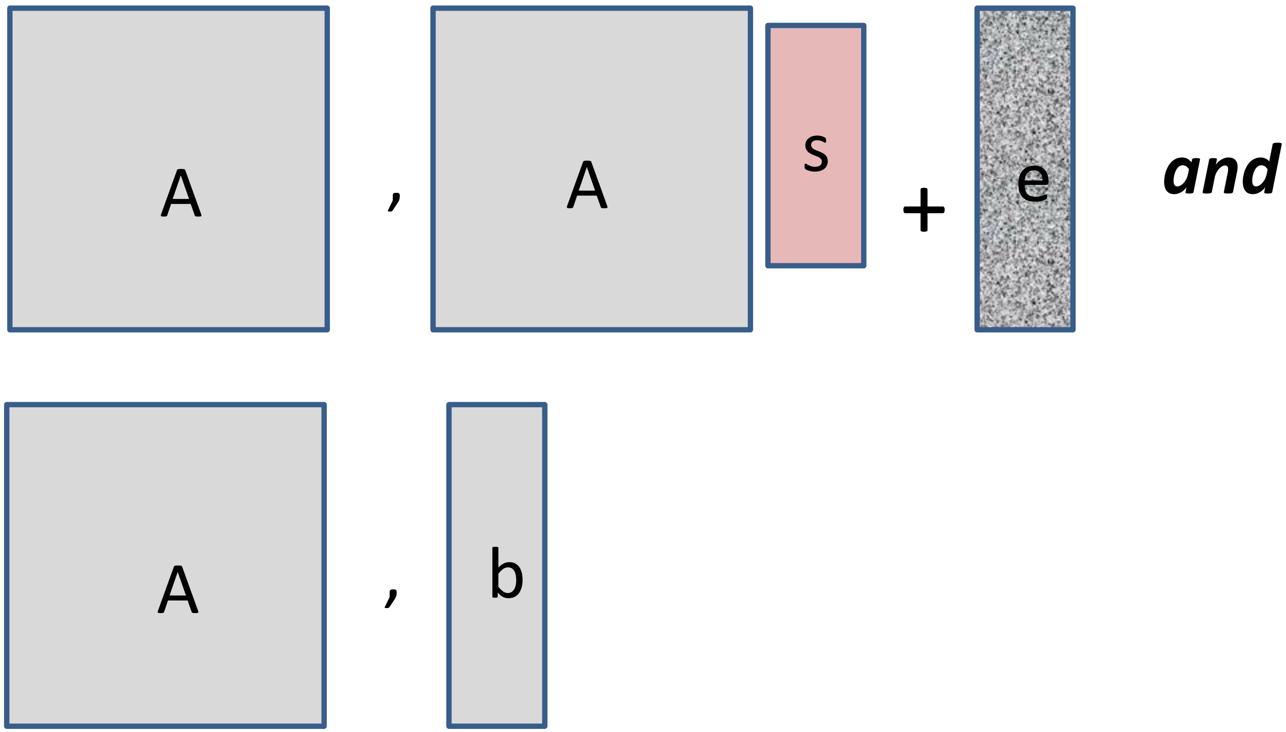
$q =$ poly in n, larger than B, and could be as large as sub-exponential, *say* $2^{n^{0.99}}$

even from quantum computers, AFAWK!



Decisional LWE

Can you distinguish between:



Theorem: "Decisional LWE is as hard as LWE".

Information-Computation Gap

Fix n, q, B.

(Search) LWE:

easy

$m = 0$

$$m \approx \frac{\boldsymbol{n}}{\left(1 - \frac{\log 2B + 1}{\log q}\right)}$$

$$m \approx \boldsymbol{2}^{\frac{\boldsymbol{n}}{\log\left(\frac{q}{2B+1}\right)}}$$

Information-theoretically impossible to recover s.

s uniquely determined given $(A, As + e)$. computationally hard to recover.

OWF and PRG

$$g_A(s,e) = \mathbf{As} + e$$

($\mathbf{A} \in Z_q^{nXm}$
$\mathbf{s} \in Z_q^n$ random “small” secret vector
$\boldsymbol{e} \in Z_q^n$: random “small” error vector)

- g_A is a one-way function (assuming LWE)
- g_A is a pseudo-random generator (decisional LWE)
- g_A is also a trapdoor function… (this is not obvious and we won’t see how in this class)

Basic (Secret-key) Encryption

[Regev05]

n = security parameter, q = “small” modulus

- Secret key sk = Uniformly random vector $\mathbf{s} \in Z_q^n$
- Encryption $\mathrm{Enc}_{\mathbf{s}}(\mu)$: // $\mu \in \{0,1\}$
  - Sample uniformly random $\mathbf{a} \in Z_q^n$, “small” noise $e \in Z$
  - The ciphertext $\mathbf{c} = (\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e + \mu \qquad)$
- Decryption $\mathrm{Dec}_{sk}(\mathbf{c})$: Output $(b - \langle \mathbf{a}, \mathbf{s} \rangle \bmod q)$

  // correctness as long as $|e| < q/4$

Basic (Secret-key) Encryption

[Regev05]

This scheme is additively homomorphic.

$\boldsymbol{c} = (\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e + \mu \lfloor q/2 \rceil)$ $+$ ← $Enc_s(m)$

$\boldsymbol{c}' = (\mathbf{a}', b' = \langle \mathbf{a}', \mathbf{s} \rangle + e' + \mu' \lfloor q/2 \rceil)$ ← $Enc_s(m')$

$\boldsymbol{c} + \boldsymbol{c}' = (\mathbf{a}+\mathbf{a}', b+b') = \langle \mathbf{a}+\mathbf{a}', \mathbf{s} \rangle + (e+e') + (\mu + \mu') \lfloor q/2 \rceil)$

In words: $c + c'$ is an encryption of $\mu + \mu'$ (mod 2)

Basic (Secret-key) Encryption

[Regev05]

You can also negate the encrypted bit easily.

We will see how to make this scheme into a fully homomorphic scheme.

For now, note that the error increases when you add two ciphertexts. That is, $|e_{add}| \approx |e_1| + |e_2| \leq 2B$.

Setting $q = n^{\log n}$ and $B = \sqrt{n}$ (for example) lets us support any polynomial number of additions.

NEXT UP:
Public-key Encryption from LWE

Public-key Encryption

[Regev05]

Here is a crazy idea. Public key has an encryption of 0 (call it c_0) and an encryption of 1 (call it c_1).
If you want to encrypt 0, output c_0 and if you want to encrypt 1, output c_1.

Well, turns out to be a crazy *bad* idea.

If only we could produce *fresh* encryptions of 0 or 1 given just the pk…

Public-key Encryption

[Regev05]

Here is another crazy idea.

Public key has *many* encryptions of 0 and an encryption of 1 (call it c_1).

If you want to encrypt 0, output a random linear combination of the 0-encryptions.

If you want to encrypt 1, output a random linear combination of the 0-encryptions plus c_1.

This one turns out to be a crazy ***good*** idea.

Regev's Public-key Encryption

- Secret key sk = Uniformly random vector **s** $\in Z_q^n$
- Public key pk: for $i\ from\ 1\ to\ m = poly(n)$

$$(\boldsymbol{c_i} = (\boldsymbol{a_i}, \langle \boldsymbol{a_i}, \boldsymbol{s} \rangle + e_i))$$

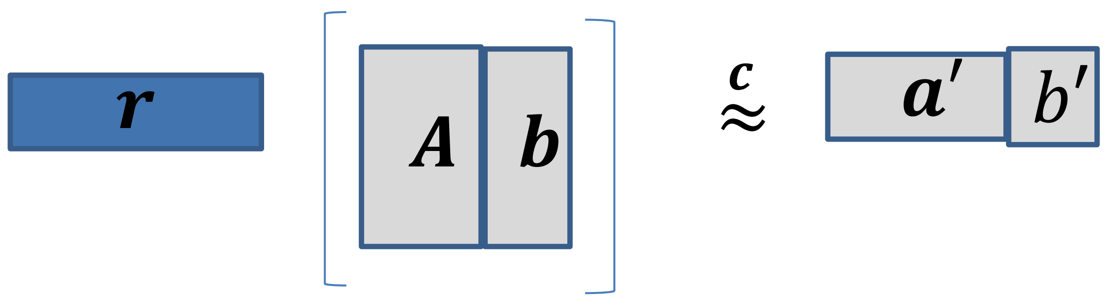
- Encrypting a bit μ: pick m random bits $r_1, \ldots, r_m$

$$\sum_{i=1}^{m} r_i \boldsymbol{c_i} + \mu \cdot \left\lfloor \frac{q}{2} \right\rfloor$$

Correctness: as long as $|\sum r_i e_i| < q/4$ is small enough.

Security: Leftover Hash Lemma
[Impagliazzo-Levin-Luby'90]

We want to understand how $\boldsymbol{rA}, \boldsymbol{rb} = \boldsymbol{r}[\boldsymbol{A} \mid \boldsymbol{b}]$ is distributed when A, b is random (and public).



If $\boldsymbol{r}$ is truly random, so is $\boldsymbol{r}[\boldsymbol{A} \mid \boldsymbol{b}]$.

But r is NOT truly random! It has small entries.

Nevertheless, $\boldsymbol{r}$ has entropy. Leftover hash lemma tells us that matrix multiplication turns (sufficient) entropy into true randomness. We need $m \gg (n+1)\log q$.

Security Proof

Theorem: under decisional LWE, the scheme is IND-secure. In fact, even more: a ciphertext together with the public key is pseudorandom.

Hybrid 1. Change the public key to random (from LWE).

$$\widetilde{\boldsymbol{pk}} = (\boldsymbol{A}, \boldsymbol{b}), \tilde{\boldsymbol{c}} = \boldsymbol{Enc}(\widetilde{\boldsymbol{pk}}, \mu) = \boldsymbol{rA}, \boldsymbol{rb} + \mu \lfloor q/2 \rfloor)$$

Hybrids 0 and 1 are comp. indist. by decisional LWE.

Security Proof

Theorem: under decisional LWE, the scheme is IND-secure. In fact, even more: a ciphertext together with the public key is pseudorandom.

Hybrid 2. Change $\boldsymbol{rA}, \boldsymbol{rb}$ to random (using Leftover hash lemma or LHL).

$$\widetilde{\boldsymbol{pk}} = (\boldsymbol{A}, \boldsymbol{b}), \tilde{\boldsymbol{c}} = \boldsymbol{Enc}\left(\widetilde{\boldsymbol{pk}}, \mu\right) = \boldsymbol{u}, u' + \mu \lfloor q/2 \rfloor)$$

Hybrids 1 and 2 are stat. indist. by LHL.

Security Proof

Theorem: under decisional LWE, the scheme is IND-secure. In fact, even more: a ciphertext together with the public key is pseudorandom.

Hybrid 3. Change $u' + \mu \lfloor q/2 \rceil$ to a random bit.

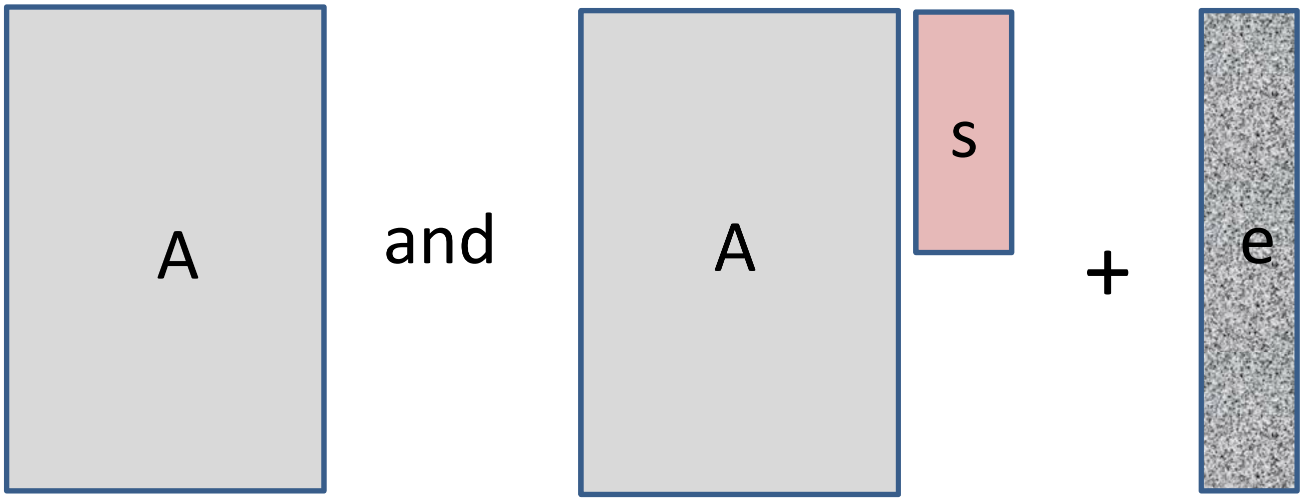
$$\widetilde{\boldsymbol{pk}} = (\boldsymbol{A}, \boldsymbol{b}), \tilde{\boldsymbol{c}} = \boldsymbol{Enc}\big(\widetilde{\boldsymbol{pk}}, \mu\big) = \boldsymbol{u}, u')$$

Hybrids 1 and 2 are perfectly indist.

NEXT UP:
Public-key Encryption from LWE

LWE with Small Secrets

Given:



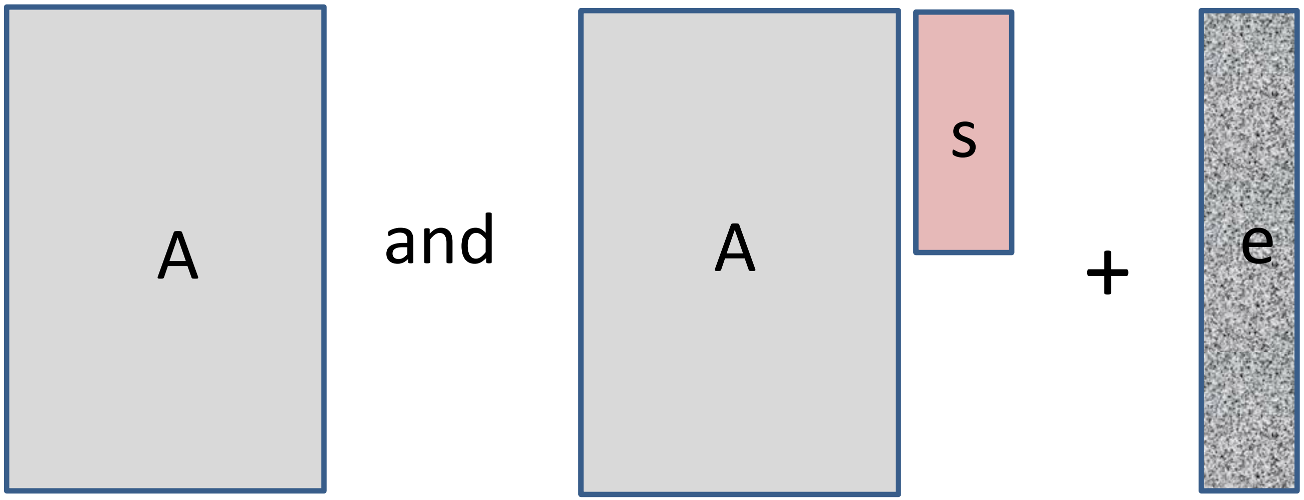
GOAL: Find s.

Parameters: dimensions $\boldsymbol{n}$ and m, modulus $\boldsymbol{q}$, error distribution χ = uniform in some interval $[-\boldsymbol{B}, \ldots, \boldsymbol{B}]$.

A is chosen at random from $\mathbb{Z}_q^{m\times n}$, **s from $\boldsymbol{\chi^n}$** and **e** from χ^m.

LWE with Small Secrets

Given:



GOAL: Find (the small secret) s.

Theorem: LWE with small secrets is as hard as LWE.

Proof on the board.

Public-key Encryption

[Lyubashevsky-Peikert-Regev'10]

- Secret key sk = Small secret **s** from χ^n
- Public key pk: for $i\ from\ 1\ to\ n$

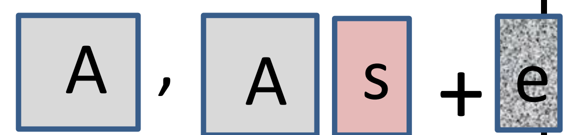
$$\boldsymbol{c_i} = (\boldsymbol{a_i}, \langle \boldsymbol{a_i}, \boldsymbol{s} \rangle + e_i)$$

Public-key Encryption

[Lyubashevsky-Peikert-Regev'10]

- Secret key sk = Small secret **s** from χ^n
- Public key pk: for $i\ from\ 1\ to\ n$

$$(\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e})$$



- Encrypting a message bit μ: pick a random vector $\boldsymbol{r}$ from χ^n

$$(\boldsymbol{r}\boldsymbol{A} + \boldsymbol{e}', \boldsymbol{r}\boldsymbol{b} + e'' + \mu \lfloor q/2 \rfloor)$$

- Decryption: compute

$$(\boldsymbol{r}\boldsymbol{b} + e'' + \mu \lfloor q/2 \rfloor) - (\boldsymbol{r}\boldsymbol{A} + \boldsymbol{e}')\boldsymbol{s}$$

and round to nearest multiple of q/2.

Correctness

- Encrypting a message bit μ: pick a random vector $\boldsymbol{r}$ from χ^n

$$(\boldsymbol{rA} + \boldsymbol{e}', \boldsymbol{rb} + e'' + \mu \lfloor q/2 \rfloor)$$

- Decryption:

$$\mathbf{(}\boldsymbol{rb} + e'' + \mu \lfloor q/2 \rfloor) - (\boldsymbol{rA} + \boldsymbol{e}')\boldsymbol{s}$$

$$= \boldsymbol{r}(\boldsymbol{As} + \boldsymbol{e}) + e'' + \mu \lfloor q/2 \rfloor - (\boldsymbol{rA} + \boldsymbol{e}')\boldsymbol{s}$$

$$= \boldsymbol{re} + e'' - \boldsymbol{e}'\boldsymbol{s} + \mu \lfloor q/2 \rfloor$$

Decryption works as long as $|\boldsymbol{re} - \boldsymbol{e}'\boldsymbol{s} + e''| < \boldsymbol{\frac{q}{4}}$.

Security

Theorem: under decisional LWE, the scheme is IND-secure. In fact, even more: a ciphertext together with the public key is pseudorandom.

We show this by a hybrid argument.

Let's stare at a public key, ciphertext pair.

$$\boldsymbol{pk} = (\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{As} + \boldsymbol{e}), \boldsymbol{c} = \boldsymbol{Enc}(\boldsymbol{pk}, \mu) = \boldsymbol{rA} + \boldsymbol{e}', \boldsymbol{rb} + e'' + \mu \lfloor q/2 \rceil$$

Call this distribution **Hybrid 0**.

Security

Theorem: under decisional LWE, the scheme is IND-secure. In fact, even more: a ciphertext together with the public key is pseudorandom.

Hybrid 1. Change the public key to random (from LWE).

$$\widetilde{\boldsymbol{pk}} = (\boldsymbol{A}, \boldsymbol{b}), \tilde{\boldsymbol{c}} = \boldsymbol{Enc}(\widetilde{\boldsymbol{pk}}, \mu) = \boldsymbol{rA} + \boldsymbol{e}', \boldsymbol{rb} + e'' + \mu \lfloor q/2 \rfloor)$$

$$= \boldsymbol{r}[\boldsymbol{A}|\boldsymbol{b}] + [\boldsymbol{e}'|e] + [0|\mu \left\lfloor \frac{q}{2} \right\rfloor]$$

Hybrids 0 and 1 are comp. indist. by decisional LWE.

Security

Theorem: under decisional LWE, the scheme is IND-secure. In fact, even more: a ciphertext together with the public key is pseudorandom.

Hybrid 2. Change $\boldsymbol{rA} + \boldsymbol{e}', \boldsymbol{rb} + e''$ into random.

$$\widetilde{\boldsymbol{pk}} = (\boldsymbol{A}, \boldsymbol{b}), \tilde{\boldsymbol{c}} = \boldsymbol{Enc}\left(\widetilde{\boldsymbol{pk}}, \mu\right) = \boldsymbol{a}', b' + \mu \lfloor q/2 \rfloor)$$

Hybrids 1 and 2 are comp. indist. by LWE.

Security

Theorem: under decisional LWE, the scheme is IND-secure. In fact, even more: a ciphertext together with the public key is pseudorandom.

Hybrid 2. Change $\boldsymbol{rA} + \boldsymbol{e}', \boldsymbol{rb} + e''$ into random.

$$\widetilde{\boldsymbol{pk}} = \ (\boldsymbol{A}, \boldsymbol{b}), \tilde{\boldsymbol{c}} = \boldsymbol{Enc}\big(\widetilde{\boldsymbol{pk}}, \mu\big) = \boldsymbol{a}', b' + \mu\ \lfloor q/2 \rfloor)$$

Now, we have the message μ encrypted with a one-time pad which perfectly hides μ.

Public-key Encryption

[Regev05, Micciancio'10, Lyubashevsky-Peikert-Regev'10]

- Secret key sk = Small secret **s** from χ^n
- Public key pk: for $i\ from\ 1\ to\ n$

$$(\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e})$$

- Encrypting a message bit μ: pick a random vector $\boldsymbol{r}$ from χ^n

$$(\boldsymbol{r}\boldsymbol{A} + \boldsymbol{e}', \boldsymbol{r}\boldsymbol{b} + e'' + \mu\lfloor q/2 \rfloor)$$

- Decryption: compute

$$(\boldsymbol{r}\boldsymbol{b} + e'' + \mu\lfloor q/2 \rfloor) - (\boldsymbol{r}\boldsymbol{A} + \boldsymbol{e}')\boldsymbol{s}$$

and round to nearest multiple of q/2.

Epilogue

A Big Open Question

Public-key Encryption from One-way Functions?

Impagliazzo-Rudich: Black-box separations.

Roughly speaking, says that any construction of a public-key encryption scheme in a "OWF-oracle-model" can be broken with $O(Q^2)$ queries if the honest parties make at most Q queries.

[Barak-Mahmoody'09]

This is tight w.r.t. Merkle puzzles!

Practical Considerations

I want to encrypt to Bob. How do I know his public key?

Public-key Infrastructure: a directory of identities together with their public keys.

<u>Needs to be “authenticated”:</u>

otherwise Eve could replace Bob’s pk with her own.

Practical Considerations

Public-key encryption is orders of magnitude slower than secret-key encryption.

1. We mostly showed how to encrypt bit-by-bit! Super-duper inefficient.
2. Exponentiation takes $O(n^2)$ time as opposed to typically linear time for secret key encryption (AES).
3. The n itself is large for PKE (RSA: $n \geq 2048$) compared to SKE (AES: $n = 128$).

(For Elliptic Curve El-Gamal, it's 320 bits)

Can solve problem 1 and minimize problems 2&3 using **hybrid encryption**.

Hybrid Encryption

To encrypt a long message m (think 1 GB):

Pick a random key K (think 128 bits) for a secret-key encryption

Encrypt K with the PKE: $PKE.Enc(pk, K)$

Encrypt m with the SKE: $\mathrm{SKE}.Enc(K, m)$

To decrypt: recover K using sk. Then using K, recover m