MIT 6.875

Foundations of Cryptography
Lecture 19

TODAY (and the next lecture):
Lattice-based Cryptography

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}.

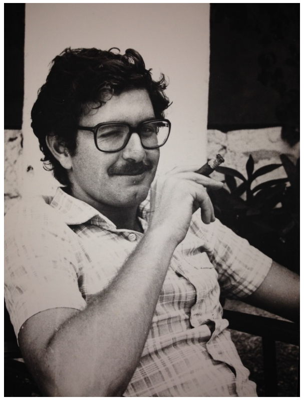
Why Lattice-based Crypto?

- ☐ **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

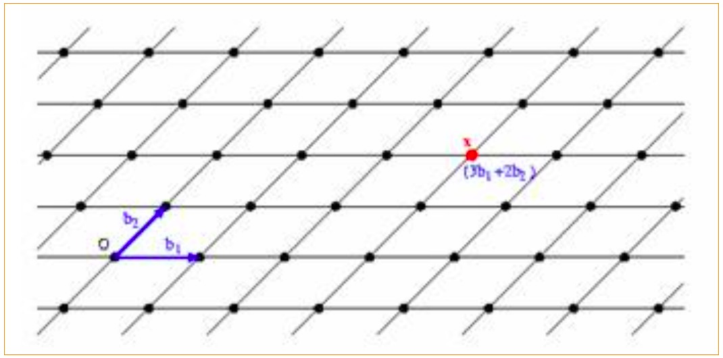
Cryptography that is (believed to be) secure against quantum attacks.

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)

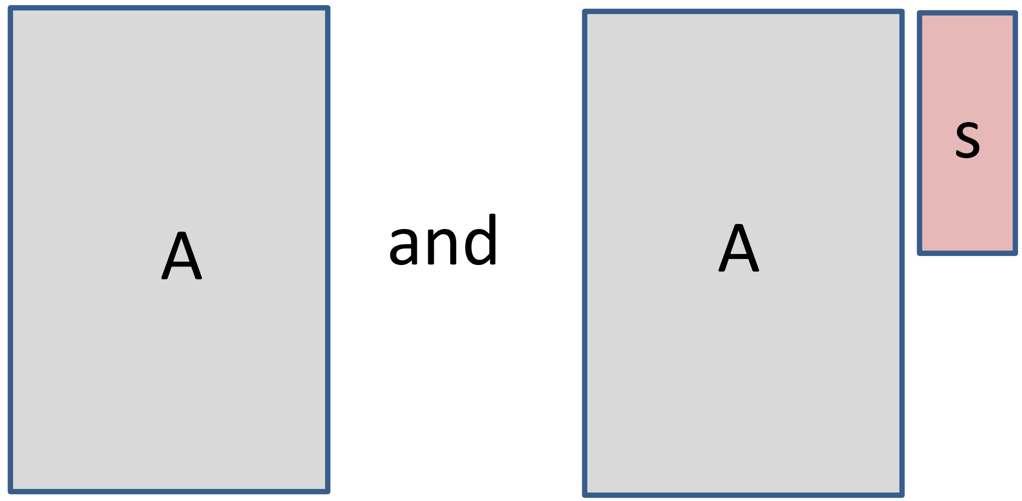
Solving Linear Equations

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

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

Solving Linear Equations

Given:

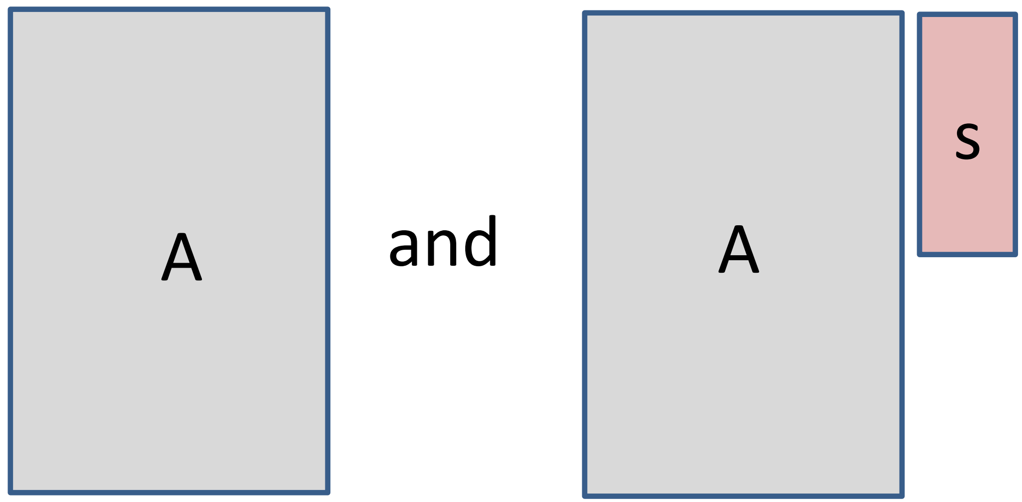


GOAL: Find s.

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

Solving Linear Equations

Given:

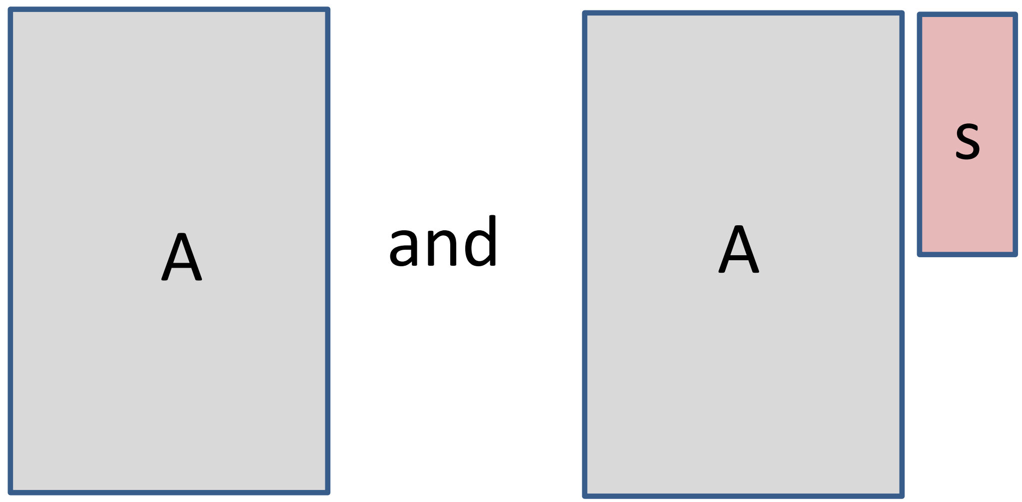


GOAL: Find s.

EASY! For example, by Gaussian Elimination

Solving Linear Equations

Given:



GOAL: Find s.

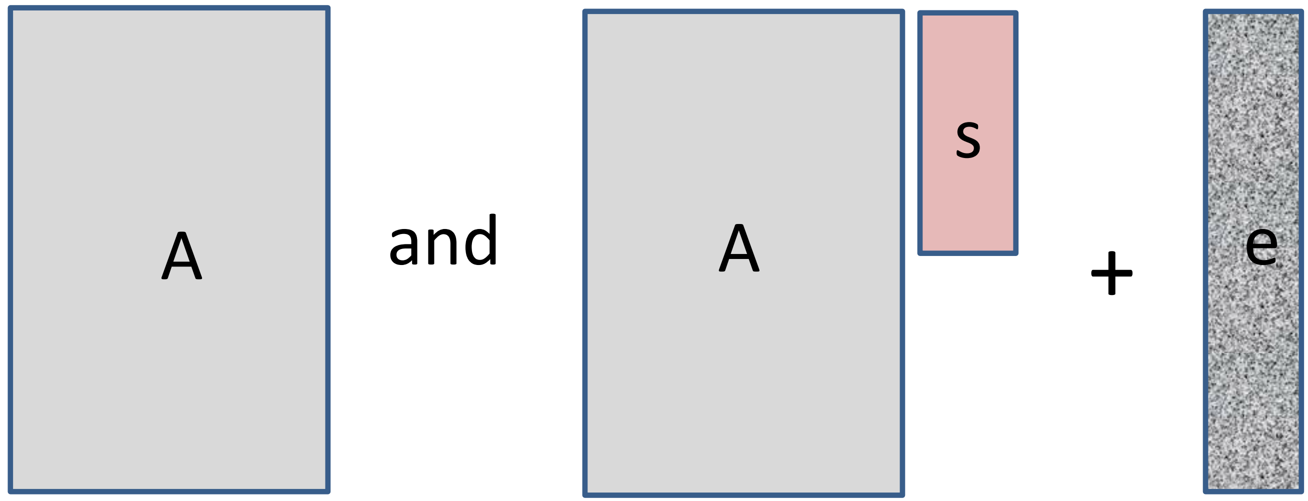
How to make it hard: Chop the head?

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

Still EASY! Gaussian Elimination mod q

Solving Linear Equations

Given:



GOAL: Find s.

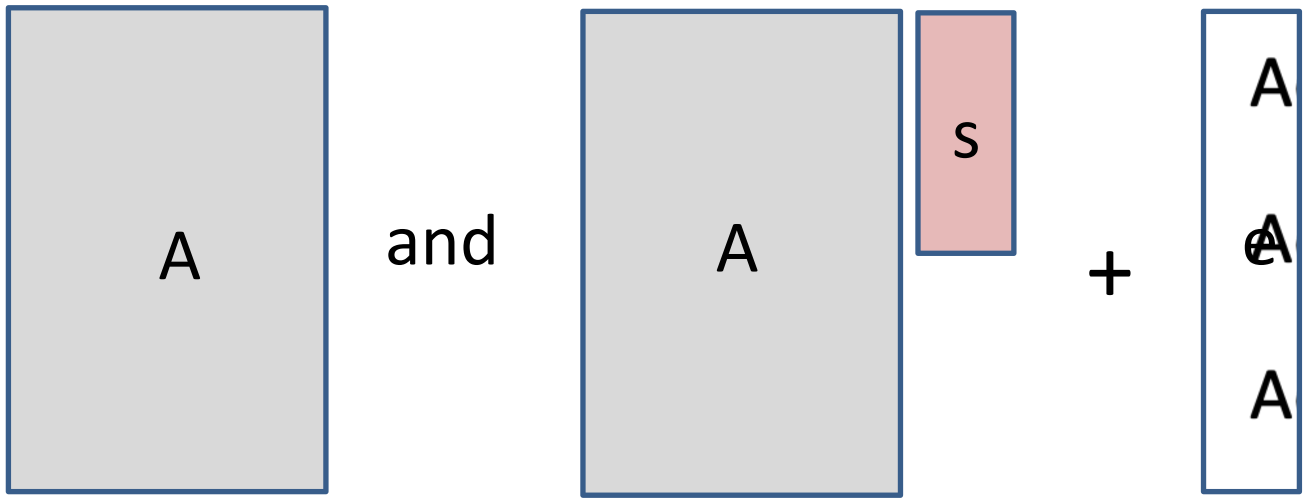
How to make it hard: Chop the tail?

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?

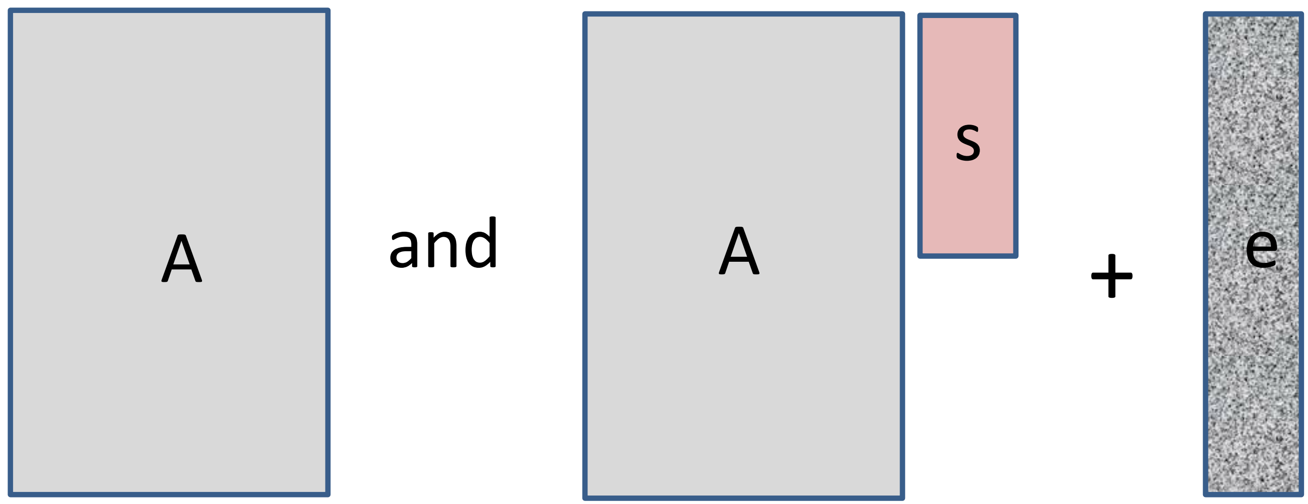
Add a small error to each equation and work mod q.

Turns out to be very HARD!

Learning with Errors (LWE)

Solving Noisy Modular Linear Equations

Given:



GOAL: Find s.

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 $\mathbb{Z}_q^n$ and **e** from χ^m.

Learning with Errors (LWE)

=

- **Decoding Random Linear Codes**

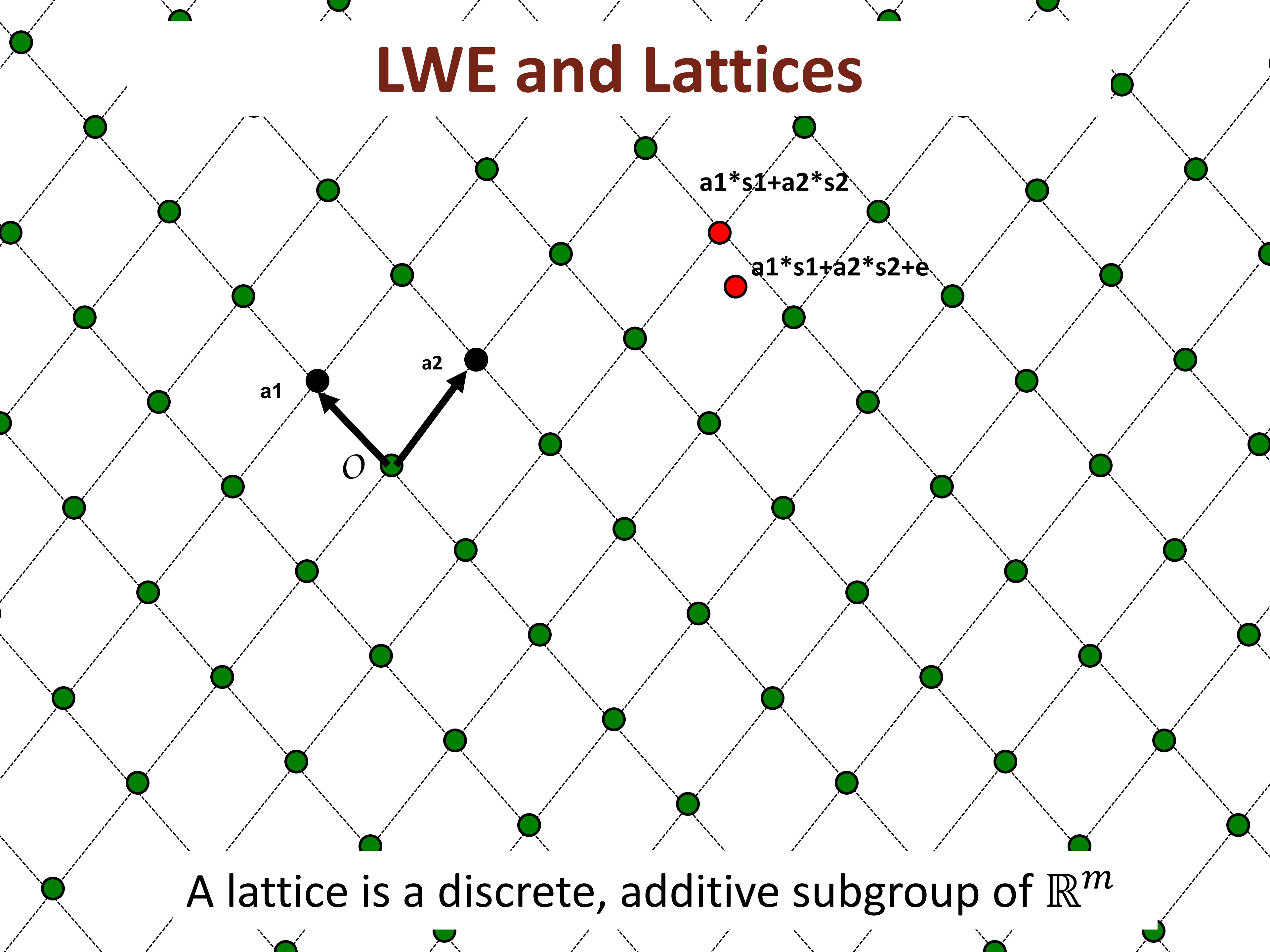
  (over F_q with L_1 errors)

- **Learning Noisy Linear Functions**

- **Worst-case hard Lattice Problems**

  [Regev'05, Peikert'09]

LWE and Lattices



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

Setting Parameters

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

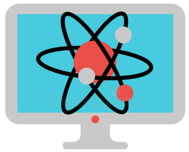
$n =$ security parameter ($\approx 1 - 10\text{K}$)

$m =$ 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* $2^{n^{0.99}}$

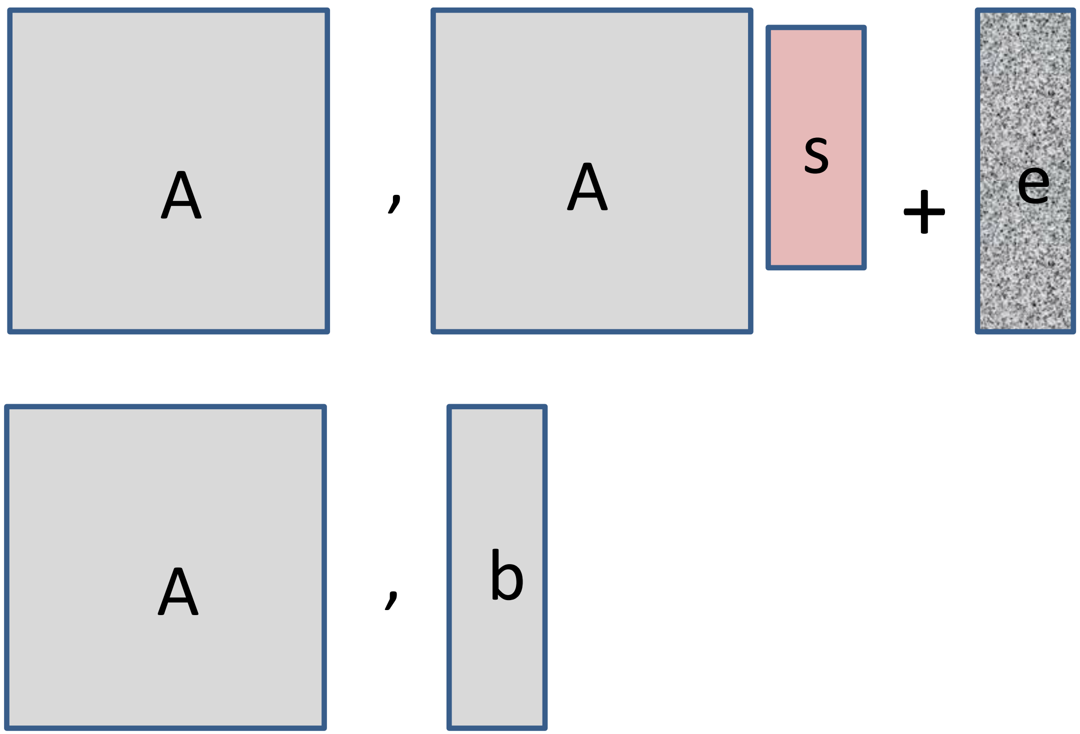
even from quantum computers, AFAWK!



QUANTUM COMPUTER

Decisional LWE

Can you distinguish between:



and

Theorem: “Decisional LWE is as hard as LWE”.

Information-Computation Gap

Fix n, q, B.

(Search) LWE:

easy

$m = 0$

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

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

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…
- also a homomorphic commitment…

Basic (Secret-key) Encryption

[Regev05]

n = security parameter, q = “small” modulus

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

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

Basic (Secret-key) Encryption

[Regev05]

This scheme is additively homomorphic.

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

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

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

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.

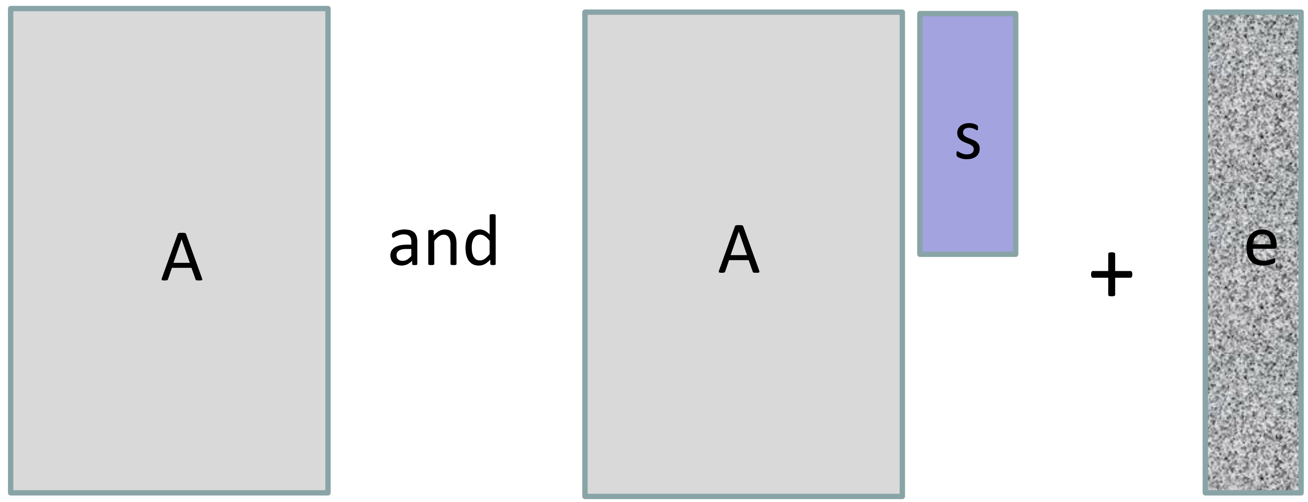
NEXT UP:

1. Public-key Encryption from LWE and

2. Fully Homomorphic Encryption

LWE with Small Secrets

Given:



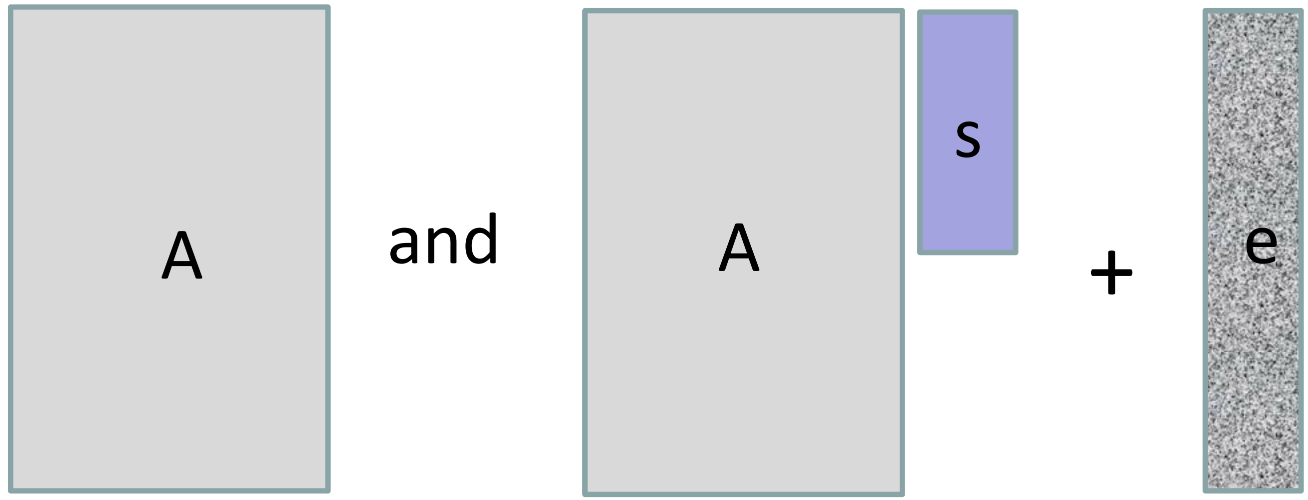
GOAL: Find s.

<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 $\boldsymbol{\chi^n}$** 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

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

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

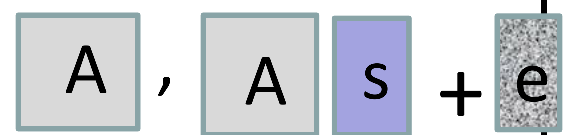
$$\boldsymbol{c_i} = (\boldsymbol{a_i}, \langle \boldsymbol{a_i}, \boldsymbol{s} \rangle + e_i)$$

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.

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:

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

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

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

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

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).

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

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.

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

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.

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

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

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

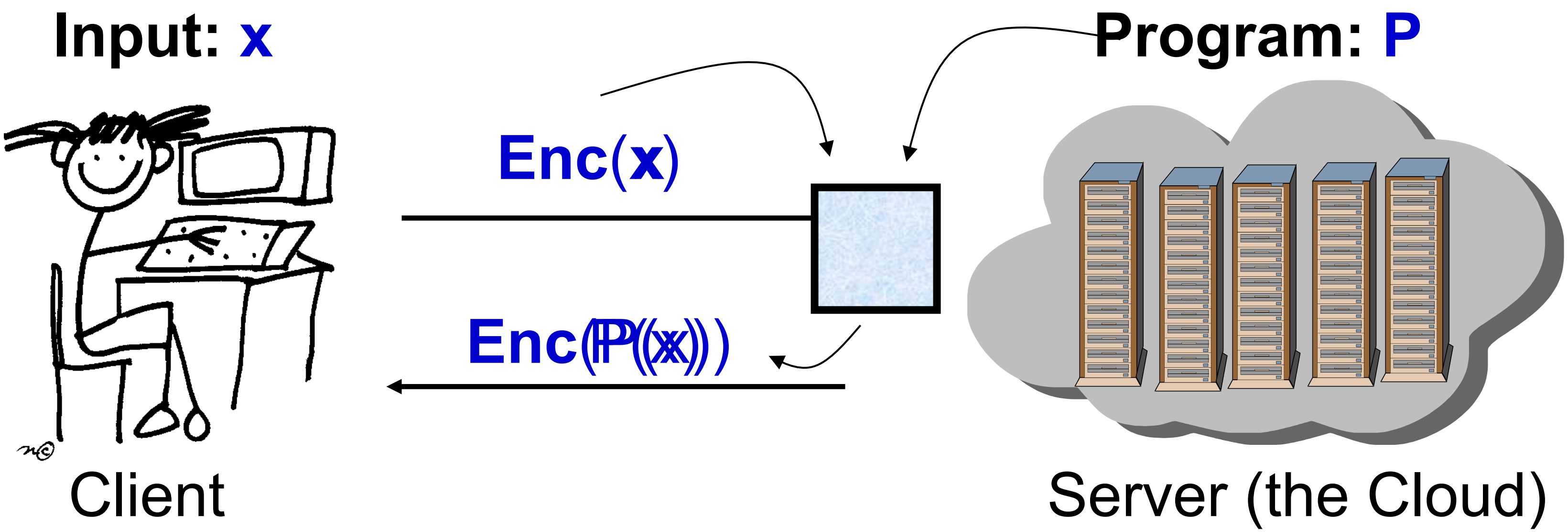
- Decryption: compute

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

and round to nearest multiple of q/2.

Homomorphic Encryption

Application 1. Secure Outsourcing



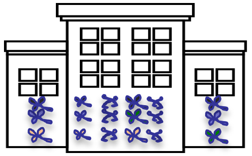
A Special Case: Encrypted Database Lookup

– also called “private information retrieval” (we’ll see in two lectures)

Application 2. Secure Collaboration

ID	Genome

Hospital



ID	Phenotype

“Parties learn the genotype-phenotype correlations and nothing else”

Homomorphic Encryption: Syntax

(can be either secret-key or public-key enc)

4-tuple of PPT algorithms $(Gen, Enc, Dec, Eval)$ s.t.

- $(sk, ek) \leftarrow Gen(1^n)$.

PPT Key generation algorithm generates a secret key as well as a (public) evaluation key.

- $c \leftarrow Enc(sk, m)$.

Encryption algorithm uses the secret key to encrypt message m.

- $c' \leftarrow Eval(ek, f, c)$.

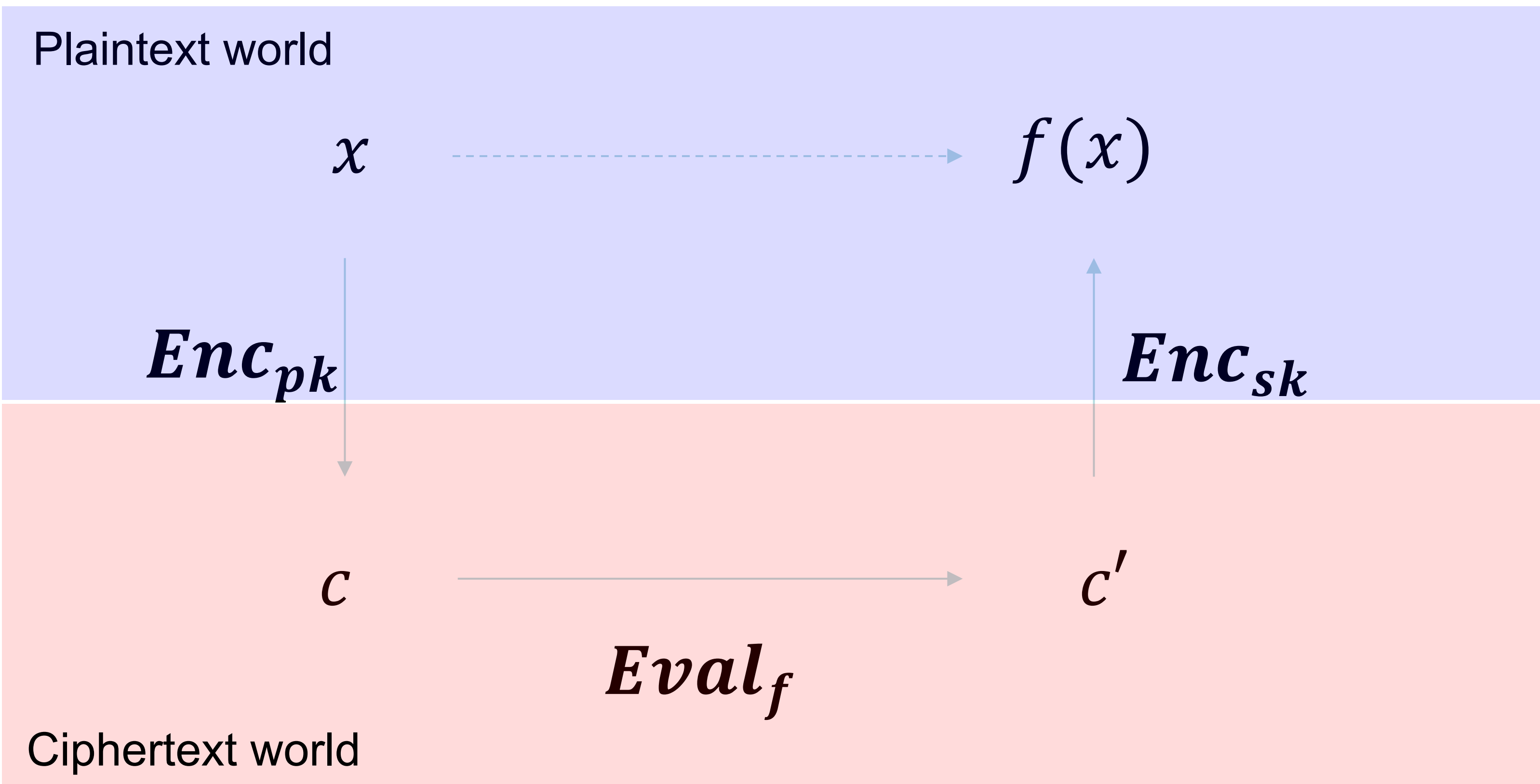
Homomorphic evaluation algorithm uses the evaluation key to produce an "evaluated ciphertext" c'.

- $m \leftarrow Dec(sk, c)$.

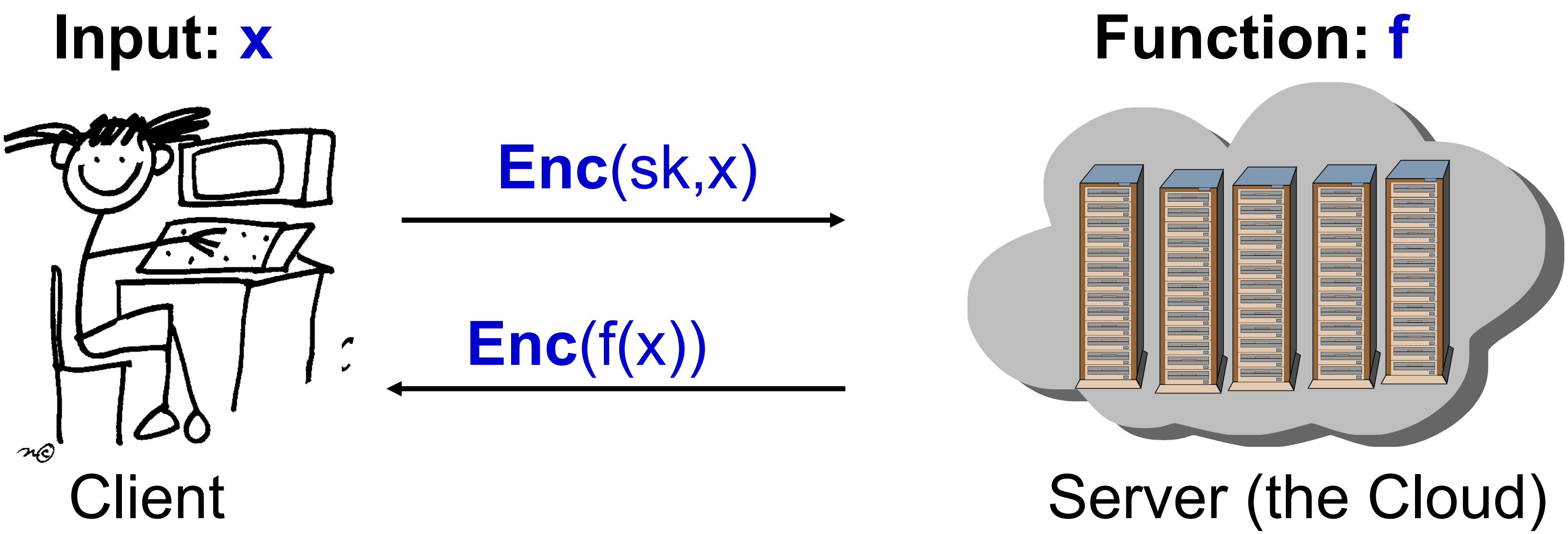
Decryption algorithm uses the secret key to decrypt ciphertext c.

Homomorphic Encryption: Correctness

$$Dec(sk, Eval(ek, f, Enc(x))) = f(x).$$



Homomorphic Encryption: Security



Security against the “curious cloud” = standard **IND-security** of secret-key encryption

Key Point: Eval is an entirely public algorithm with public inputs.

Here is a homomorphic encryption scheme…

- $(sk, -) \leftarrow Gen(1^n)$.

Use any old secret key enc scheme.

- $c \leftarrow Enc(sk, m)$.

Just the secret key encryption algorithm…

- $c' \leftarrow Eval(ek, f, c)$.

Output c' = c || f. So Eval is basically the identity function!!

- $m \leftarrow Dec(sk, c')$.

Parse $c' = c||f$ as a ciphertext concatenated with a function description. Decrypt c and compute the function f.

This is correct and it is IND-secure.

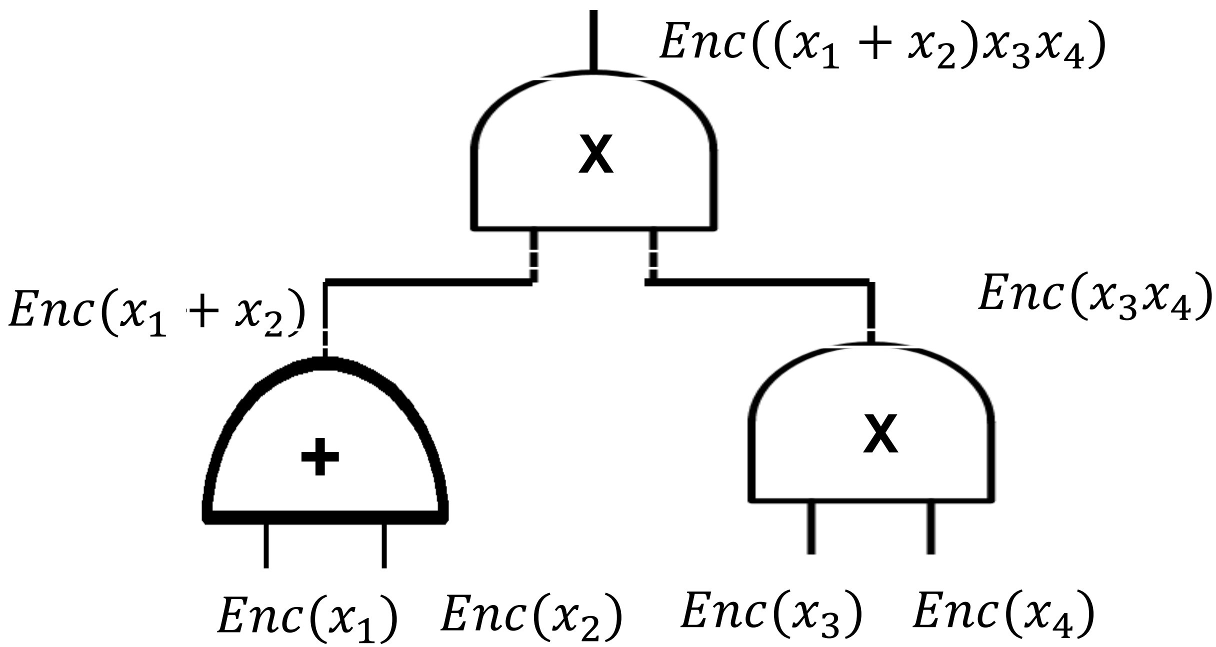
Homomorphic Encryption: Compactness

The size (bit-length) of the evaluated ciphertext and the runtime of the decryption is *independent of* the complexity of the evaluated function.

A Relaxation: The size (bit-length) of the evaluated ciphertext and the runtime of the decryption *depends sublinearly on* the complexity of the evaluated function.

How to Compute Arbitrary Functions

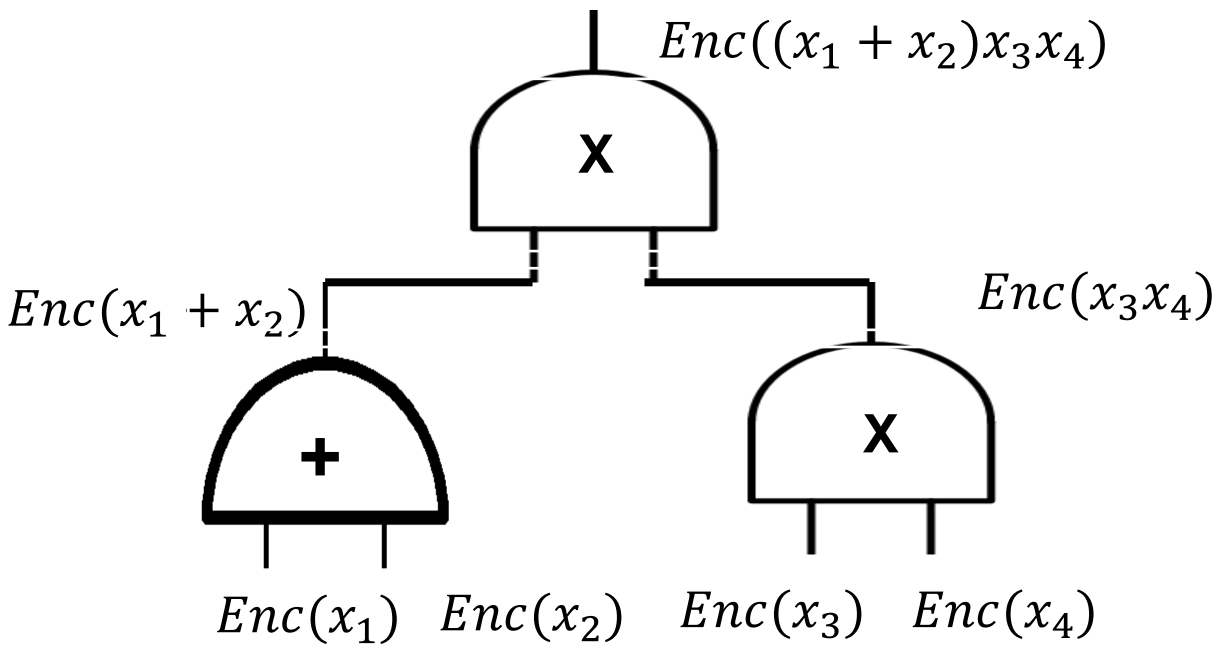
For us, programs = functions = Boolean circuits with XOR ($+\ mod\ 2$) and AND ($\times\ mod\ 2$) gates.



Takeaway: If you can compute XOR and AND on encrypted bits, you can compute everything.

How to Compute Arbitrary Functions

For us, programs = functions = Boolean circuits with XOR ($+\ mod\ 2$) and AND ($\times\ mod\ 2$) gates.



We already know how to add (XOR), can we multiply?? Next lecture…