MIT 6.875

Foundations of Cryptography
Lecture 22

Complexity of the 2PC protocols

Number of OT protocol invocations $= 2 * \#AND$ gates

Can be made into O(#inputs · λ): Yao's garbled circuits

Number of rounds = AND-depth of the circuit

Can be made into O(1) rounds: Yao's garbled circuits

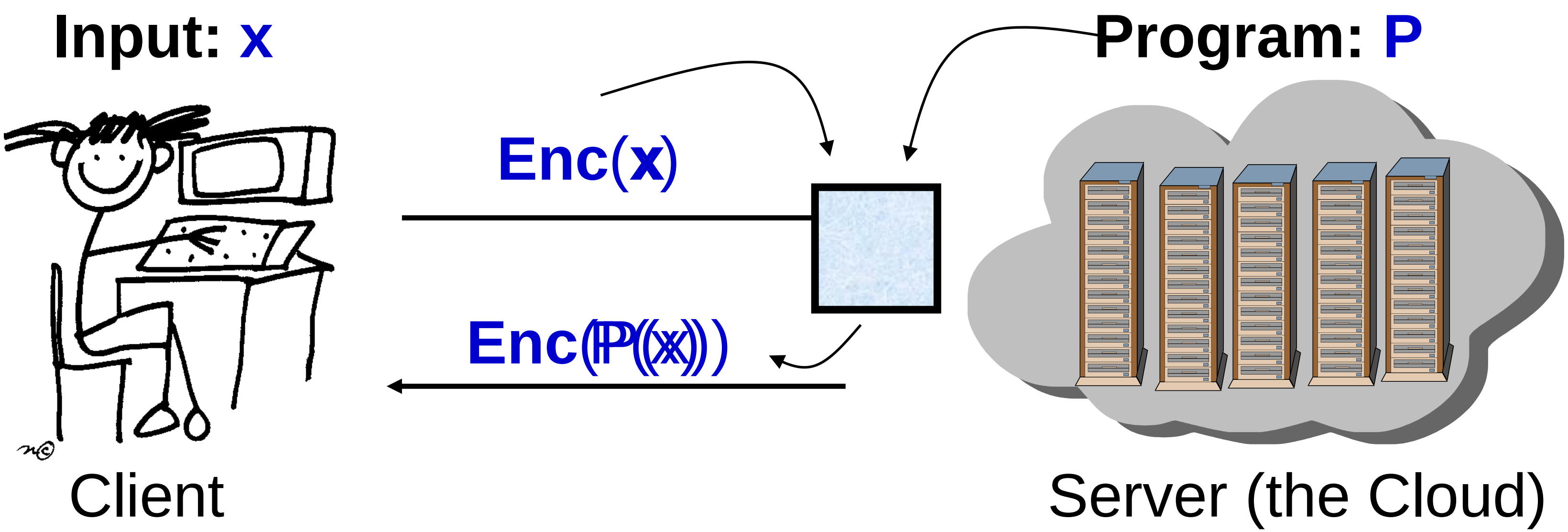
Communication in bits =

$$O(\#AND \cdot \lambda + \#outputs)$$

Can be made into O(λ #inputs) using FHE: but FHE is computationally more expensive concretely.

Homomorphic Encryption

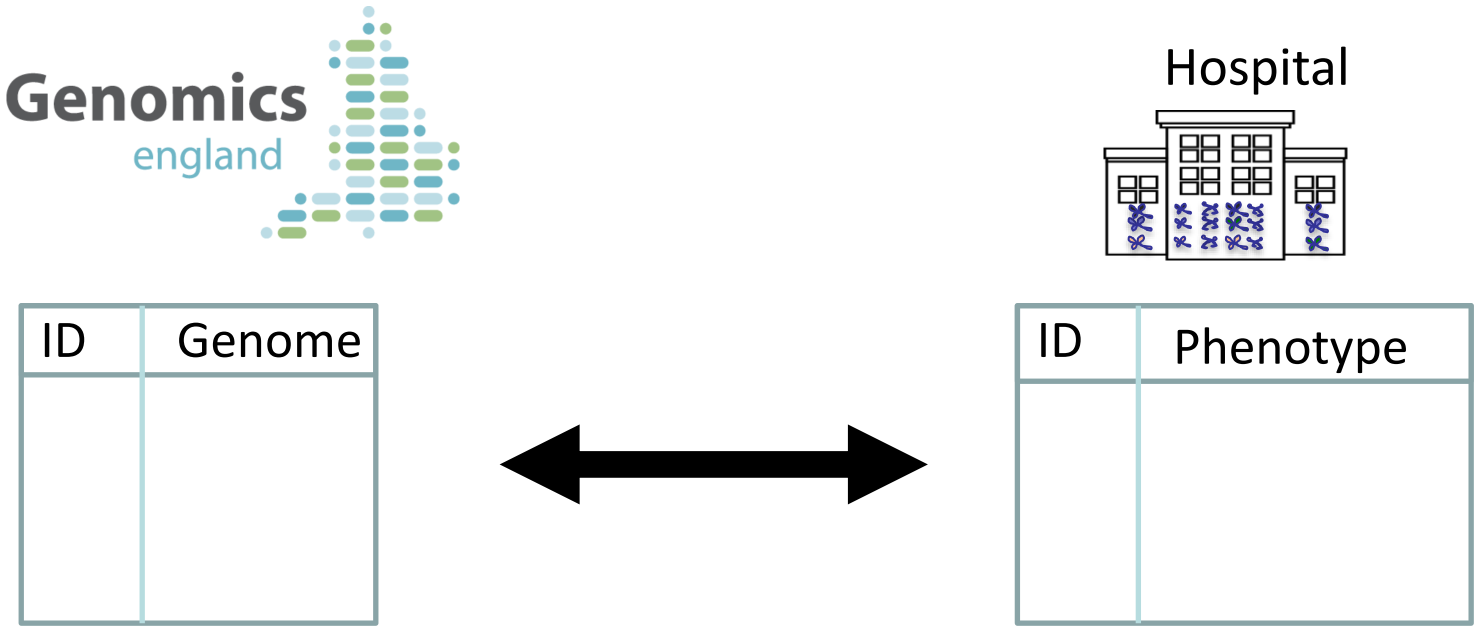
Application 1. Secure Outsourcing



A Special Case: Encrypted Database Lookup

– also called "private information retrieval" (we'll see in two lectures)

Application 2. Secure Collaboration



"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.

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

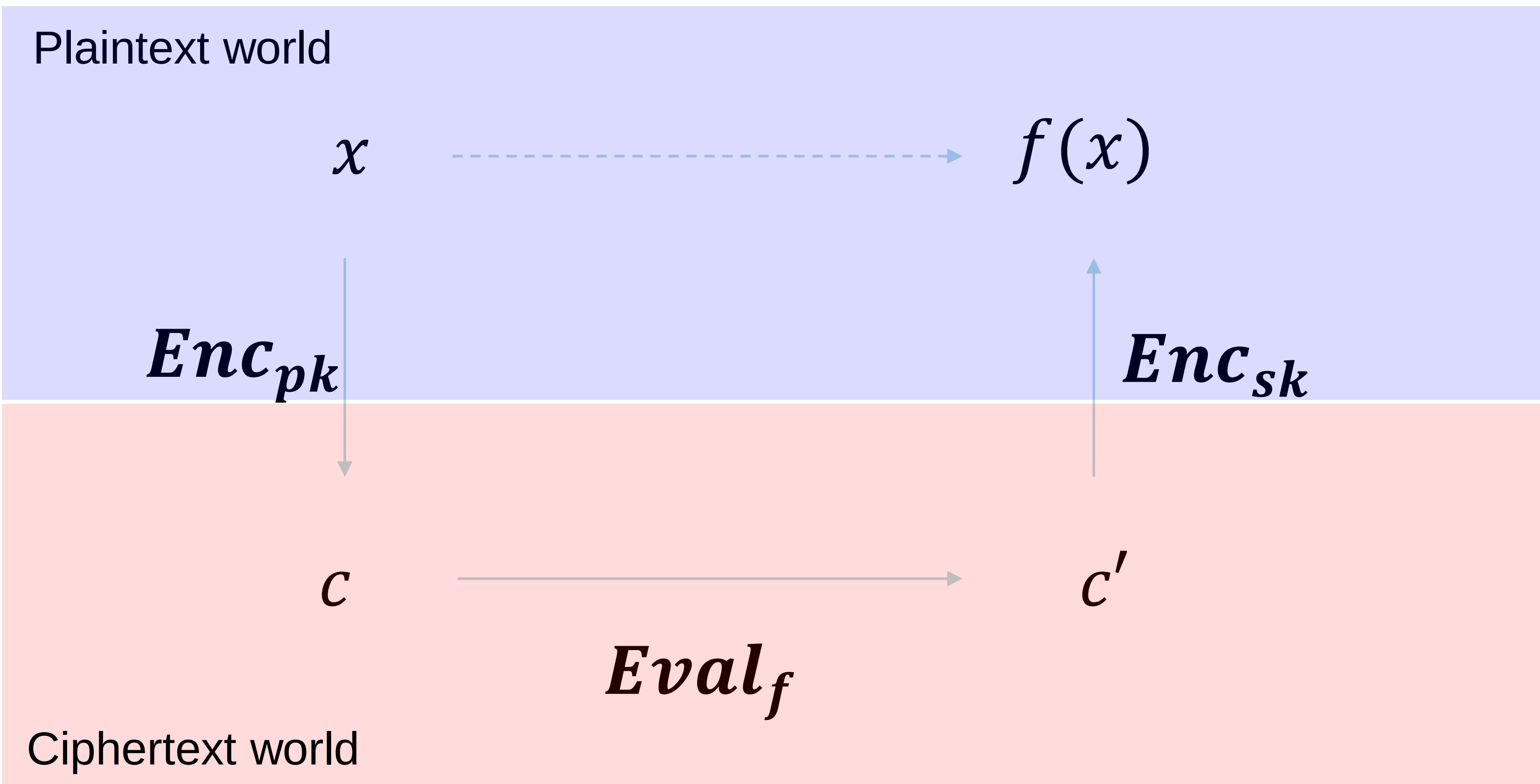
Homomorphic evaluation algorithm uses the evaluation key to produce an "evaluated ciphertext" c'.

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

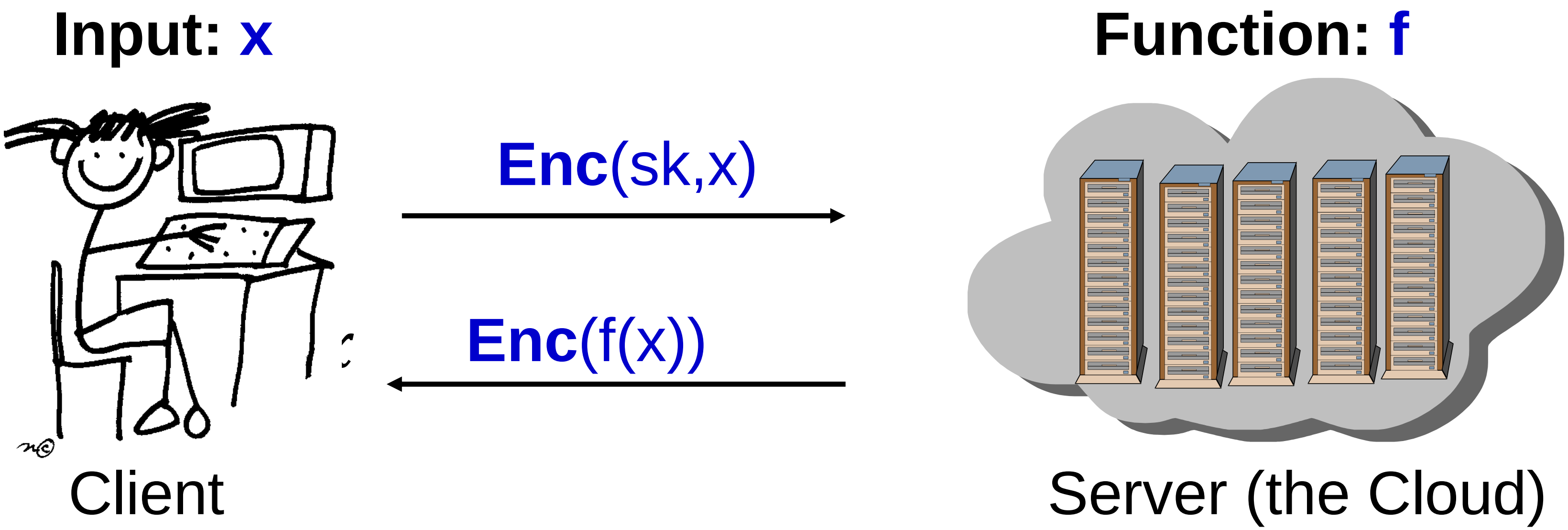
Decryption algorithm uses the secret key to decrypt ciphertext c.

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 $c' = c \mid\mid f$. 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.

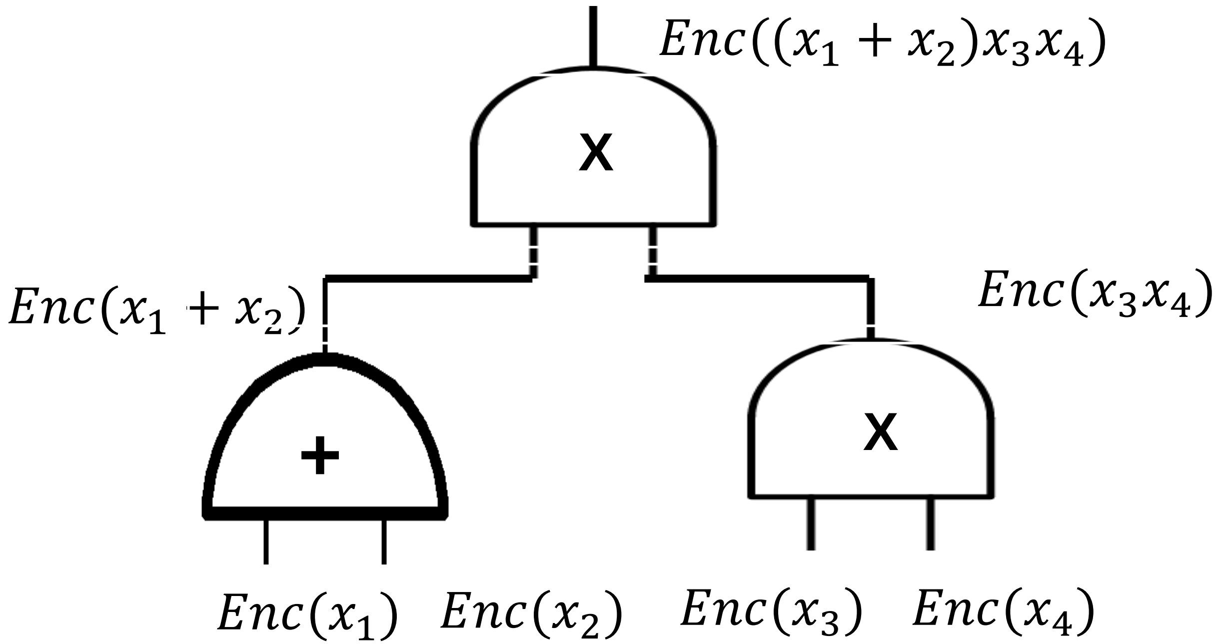
Homomorphic Encryption: Compactness

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

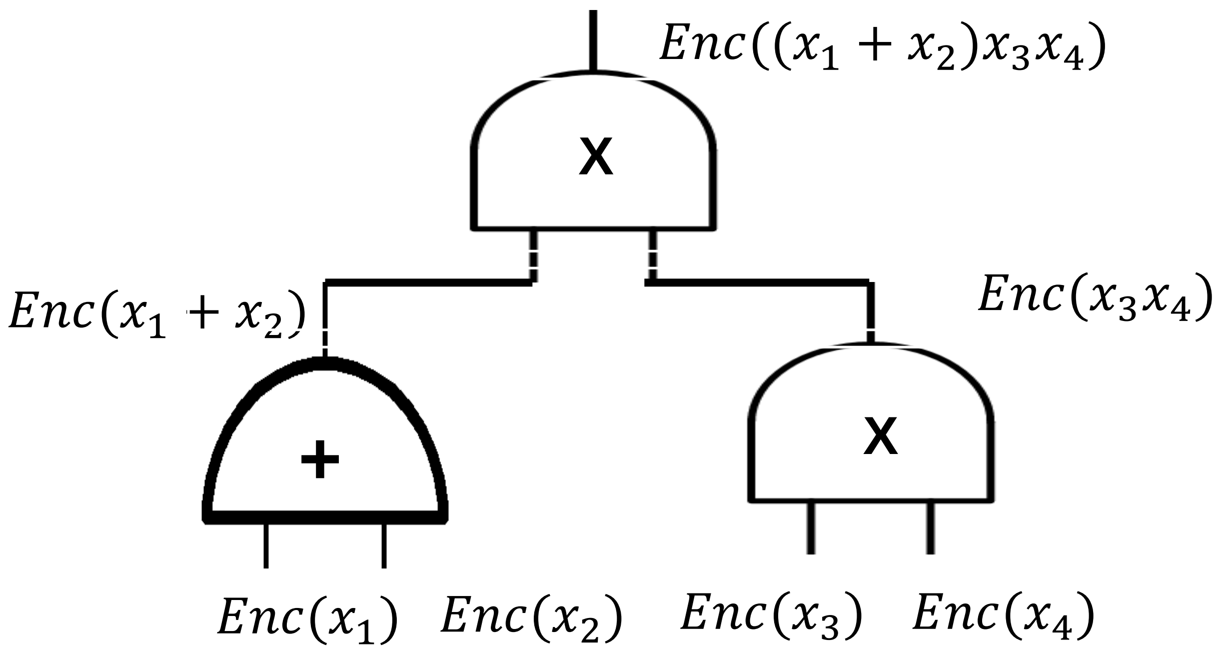
For us, programs = functions = Boolean circuits with XOR ($+ \, mod\ 2$) and AND ($\times \, mod\ 2$) gates.



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…

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.

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

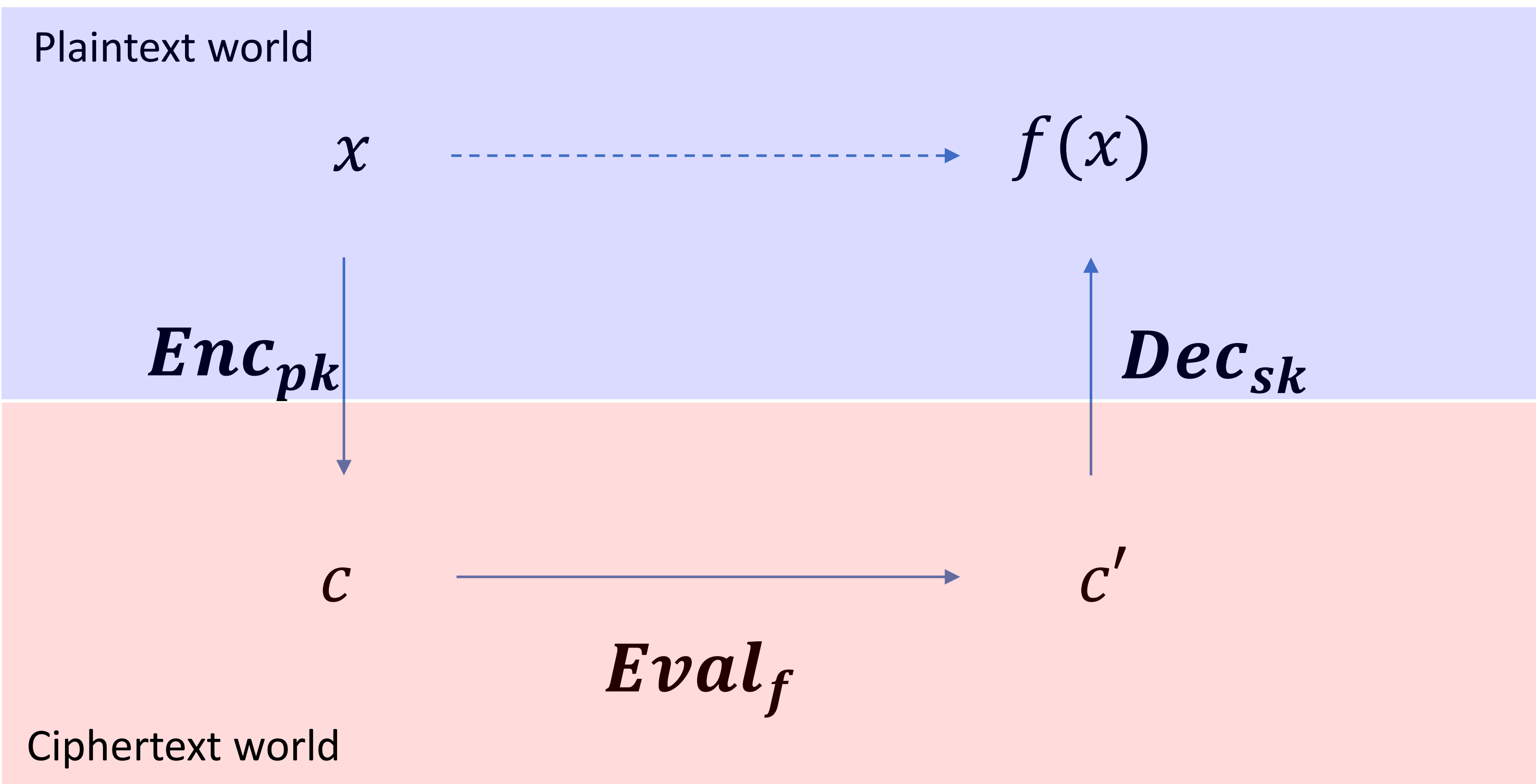
Homomorphic evaluation algorithm uses the evaluation key to produce an "evaluated ciphertext" c'.

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

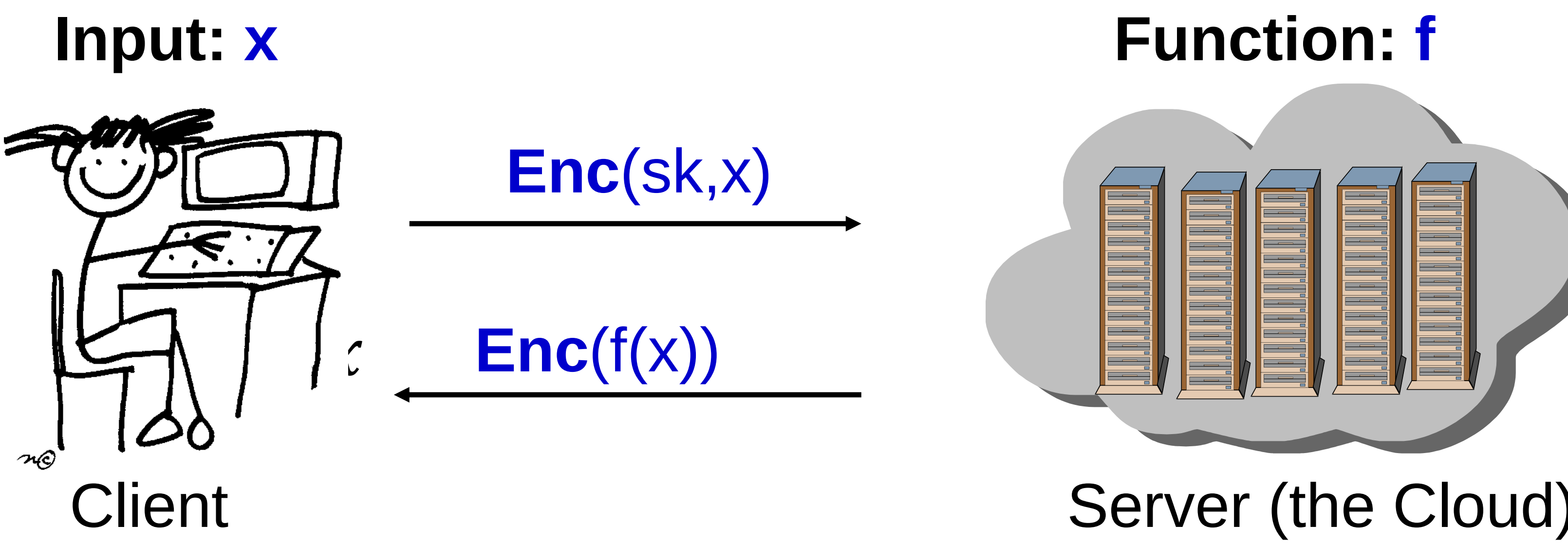
Decryption algorithm uses the secret key to decrypt ciphertext c.

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 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.

Big Picture: Two Steps to FHE

Leveled Secret-key Homomorphic Encryption:
Evaluate circuits of a-priori bounded depth *d*

"you give me a depth bound d, I will give you a homomorphic scheme that handles depth-d circuits…"

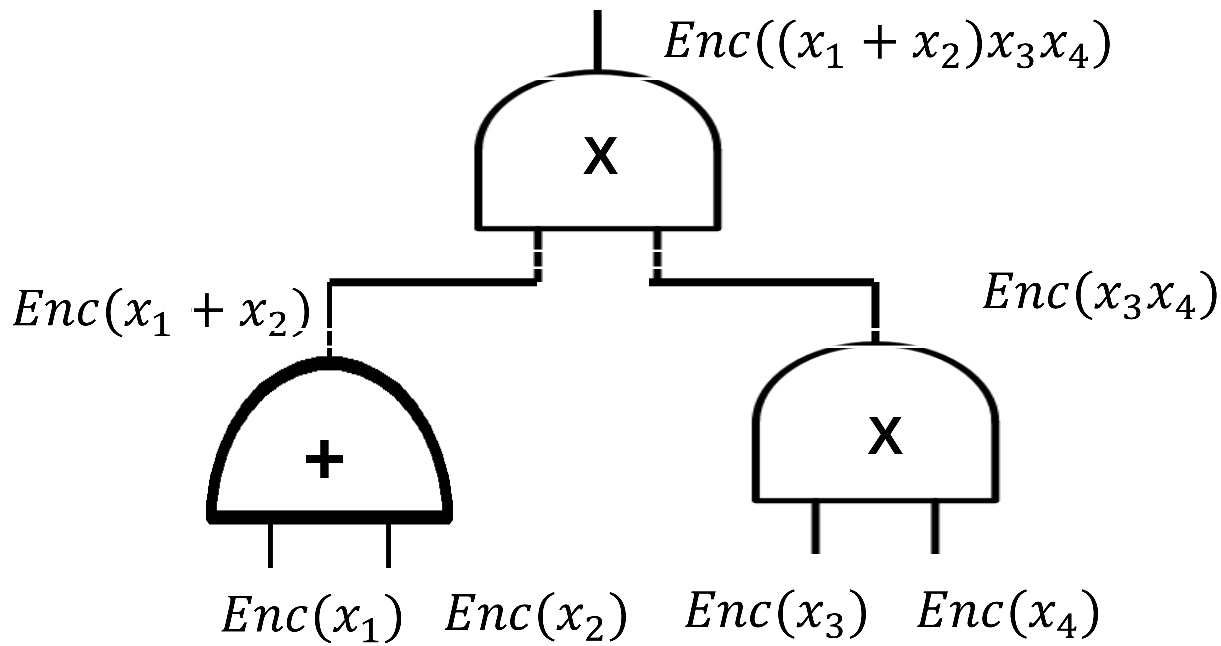
Bootstrapping Theorem:

From "circular secure" Leveled FHE to Pure FHE (at the cost of an additional assumption)

"I will give you homomorphic scheme that handles circuits of ANY size/depth"

How to Compute Arbitrary Functions

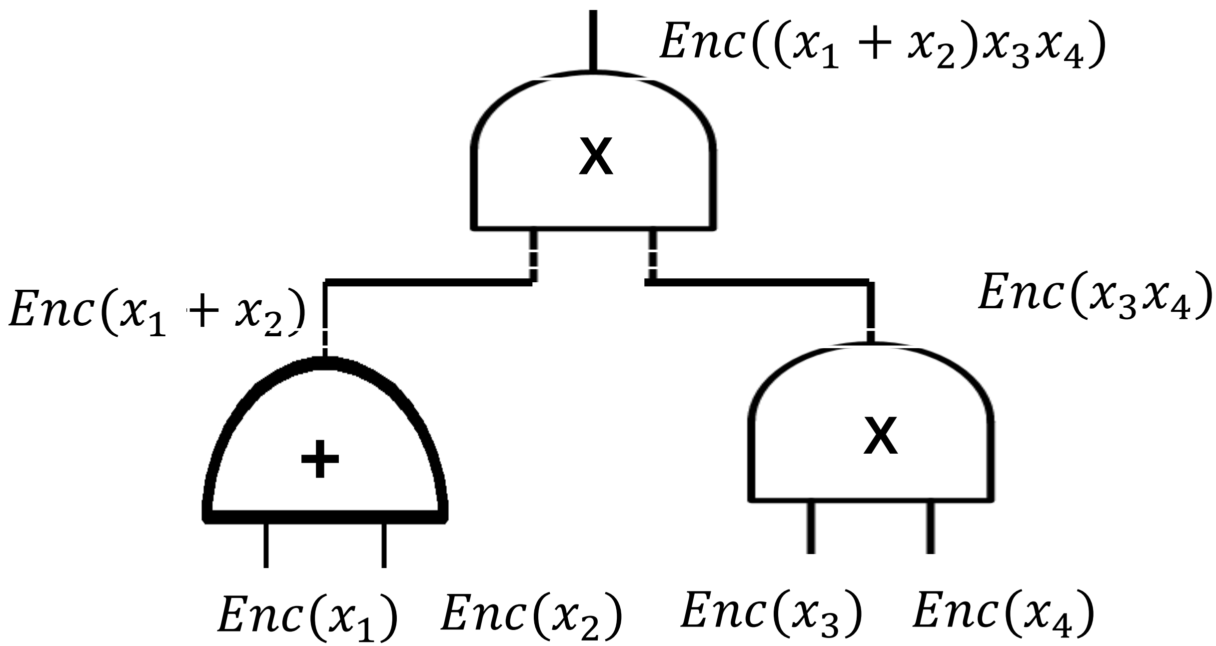
For us, programs = functions = Boolean circuits with XOR ($+\ mod\ 2$) and AND ($\times\ mod\ 2$) gates.



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??

New (Secret-key) Encryption: Take 1

- Private key: a vector **s** $\in Z_q^n$

- Private-key Encryption of a bit $m \in \{0, 1\}$:

$$\mathbf{C} = \begin{bmatrix} A \\ sA \end{bmatrix} + m\, I \qquad (A \text{ is random (n) X (n+1) matrix})$$

- Decryption:

$$\underbrace{[\mathbf{s} \,||\, -1]}_{\text{Priv key = Eigenvector}} \quad \underbrace{\mathbf{C}}_{\text{Ciphertext matrix}} \quad = \quad \underbrace{m}_{\text{Message = Eigenvalue}} \; [\mathbf{s} \,||\, -1] \quad (\text{mod } q)$$

☹ **INSECURE!** Easy to solve linear equations.

New (Secret-key) Encryption: Take 1

$$\mathbf{t} \mathbin{.} \mathbf{C} = m \mathbin{.} \mathbf{t} \pmod q$$

$t = [s \,||\, -1]$

- ► Homomorphic addition: $\mathbf{C_1} + \mathbf{C_2}$
  - – t is an eigenvector of C_1+C_2 with eigenvalue m_1+m_2
- ► Homomorphic multiplication: $\mathbf{C_1 C_2}$
  - – t is an eigenvector of C_1C_2 with eigenvalue m_1m_2

Proof: $\mathbf{t} \mathbin{.} \mathbf{C_1} \, \mathbf{C_2} = (m_1 \mathbin{.} \mathbf{t}) \mathbin{.} \mathbf{C_2} = m_1 \mathbin{.} m_2 \mathbin{.} \mathbf{t}$

But, remember, the scheme is insecure?

Key idea: fix insecurity while retaining homomorphism.

New (Secret-key) Encryption: Take 2

- Private key: a vector **s** $\in \boldsymbol{Z_q^n}$

- Private-key Encryption of a bit $m \in \{\mathbf{0}, \mathbf{1}\}$:

$$\mathbf{C} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{sA} + \boldsymbol{e} \end{bmatrix} + m\,\boldsymbol{I}$$

($\boldsymbol{A}$ is random (n+1) X n matrix)

- Decryption:

$$\underbrace{[\mathbf{s} \,||\, \mathbf{-1}]}_{\text{Priv key = Approx Eigenvector}} \; \underbrace{\mathbf{C}}_{\text{Ciphertext matrix}} \approx \underbrace{m}_{\text{Message = Approx Eigenvalue}} [\mathbf{s} \,||\, \mathbf{-1}] \pmod q$$

☺ **CPA-secure by LWE.**

New (Secret-key) Encryption: Take 2

t . C = m . t + e (mod q)

t = [s || -1]

► Homomorphic addition: $\mathbf{C_1} + \mathbf{C_2}$

$$
\begin{aligned}
\vec{t} \cdot (C_1 + C_2) &= \vec{t}C_1 + \vec{t}C_2 \\
&= m_1\vec{t} + \vec{e}_1 + m_2\vec{t} + \vec{e}_2 \\
&= (m_1 + m_2)\vec{t} + (\vec{e}_1 + \vec{e}_2) \\
&\approx (m_1 + m_2)\vec{t}
\end{aligned}
$$

Noise grows a little

New (Secret-key) Encryption: Take 2

t . C = m . t + e (mod q)

t = [s || -1]

► Homomorphic multiplication: $\mathbf{C_1\, C_2}$

Can also use $C_2 C_1$

$$
\begin{aligned}
\vec{t}\cdot(C_1\cdot C_2) &= (m_1\vec{t}+\vec{e}_1)C_2 \\
&= m_1\vec{t}C_2+\vec{e}_1C_2 \\
&= m_1(m_2\vec{t}+\vec{e}_2)+\vec{e}_1C_2 \\
&= m_1m_2\vec{t}+\underbrace{m_1\vec{e}_2+\vec{e}_1C_2}_{\vec{e}_{mult}}
\end{aligned}
$$

Noise grows.

Need C_2 to be small! How?!

Aside: Binary Decomposition

Break each entry in C into its binary representation

$$C = \begin{bmatrix}3 & 5\\ 1 & 4\end{bmatrix} \ (mod\ 8) \Longrightarrow bits(C) = \begin{bmatrix}0 & 1\\ 1 & 0\\ 1 & 1\\ 0 & 1\\ 0 & 0\\ 1 & 0\end{bmatrix} \ (mod\ 8)$$

Small entries like we wanted!

Consider the "reverse" operation:

$$k \updownarrow \overbrace{\left[\qquad\qquad\right]}^{k \log q} \cdot bits(C) = C \Longrightarrow \boxed{\vec{t} \cdot C = \vec{t} \cdot G \cdot G^{-1}(C)}$$

G

Denote: $G^{-1}(C)$ which has "small" entries

New (Secret-key) Encryption: Take 3

- Private key: a vector **s** $\in \mathbb{Z}_q^n$

- Private-key Encryption of a bit $m \in \{0, 1\}$:

$$\mathbf{C} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{sA} + \boldsymbol{e} \end{bmatrix} + m\, G$$

(**A** is random (n+1) X n log q matrix)

- Decryption:

$$\underbrace{[\mathbf{s} \,\|\, -1]}_{\text{Priv key = Approx Eigenvector}} \quad \underbrace{\mathbf{C}}_{\text{Ciphertext matrix}} \quad \approx \quad \underbrace{m}_{\text{Message = Approx “Eigenvalue”}} \, [\mathbf{s} \,\|\, -1] \, \mathbf{G} \pmod q$$

☺ **Still CPA-secure by LWE.**

New (Secret-key) Encryption: Take 3

t . C = m . t . G + e (mod q)

t = [s || -1]

► Homomorphic multiplication: $C_{mult} = C_1 \cdot G^{-1}(C_2)$

$$\begin{aligned}
\vec{s} \cdot C_1 \cdot G^{-1}(C_2) &= (\vec{e}_1 + m_1 \cdot \vec{s} \cdot G) \cdot G^{-1}(C_2) \\
&= \vec{e}_1 \cdot G^{-1}(C_2) + m_1 \cdot \vec{s} \cdot G \cdot G^{-1}(C_2) \\
&= \vec{e}_1 \cdot G^{-1}(C_2) + m_1 \cdot \vec{s} \cdot C_2 \\
&= \vec{e}_1 \cdot G^{-1}(C_2) + m_1 \cdot (\vec{e}_2 + m_2 \cdot \vec{s} \cdot G) \\
&= \underbrace{(\vec{e}_1 \cdot G^{-1}(C_2) + m_1 \cdot \vec{e}_2)}_{\vec{e}_{mult}} + m_1 m_2 \cdot \vec{s} \cdot G
\end{aligned}$$

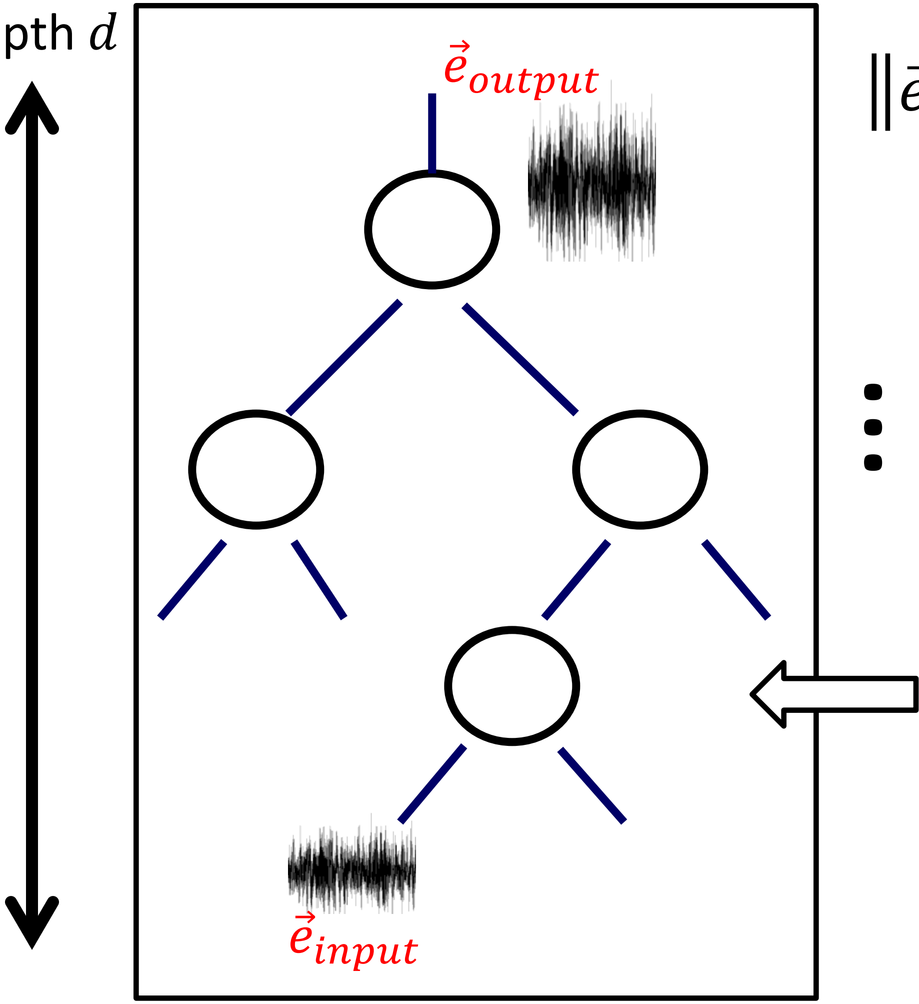
$$\|\vec{e}_{mult}\| \leq n \log q \cdot \|\vec{e}_1\| + m_1 \cdot \|\vec{e}_2\| \leq (n \log q + 1) \cdot \max\{\|\vec{e}_1\|, \|\vec{e}_2\|\}$$

Let $N = n \log q$

Homomorphic Circuit Evaluation

Noise grows during homomorphic eval

Depth d



$$\|\vec{e}_{output}\| \leq (N+1)^d \cdot B_0 \approx N^d B_0$$

$\Rightarrow$ Decryptable if $q \gg N^d B_0$.

(for security: $q \ll 2^n$)

So this can support $d \approx n^{0.99}$

$\vdots$

$$\|\vec{e}_{i+1}\| \leq (N+1)\|\vec{e}_i\|$$

$$\|\vec{e}_{input}\| \leq B_0$$

Big Picture: Two Steps to FHE

Leveled Secret-key Homomorphic Encryption:
Evaluate circuits of a-priori bounded depth *d*

"you give me a depth bound d, I will give you a homomorphic scheme that handles depth-d circuits…"

Bootstrapping Theorem:

From "circular secure" Leveled FHE to Pure FHE (at the cost of an additional assumption)

"I will give you homomorphic scheme that handles circuits of ANY size/depth"

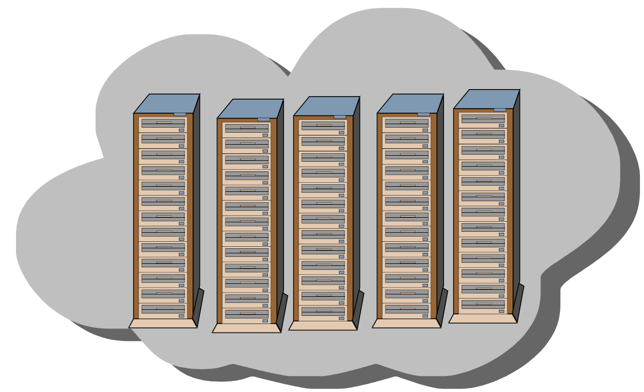
From Leveled to Fully Homomorphic

Input: x

Client

Enc(sk,x)

Function: f



Server (the Cloud)

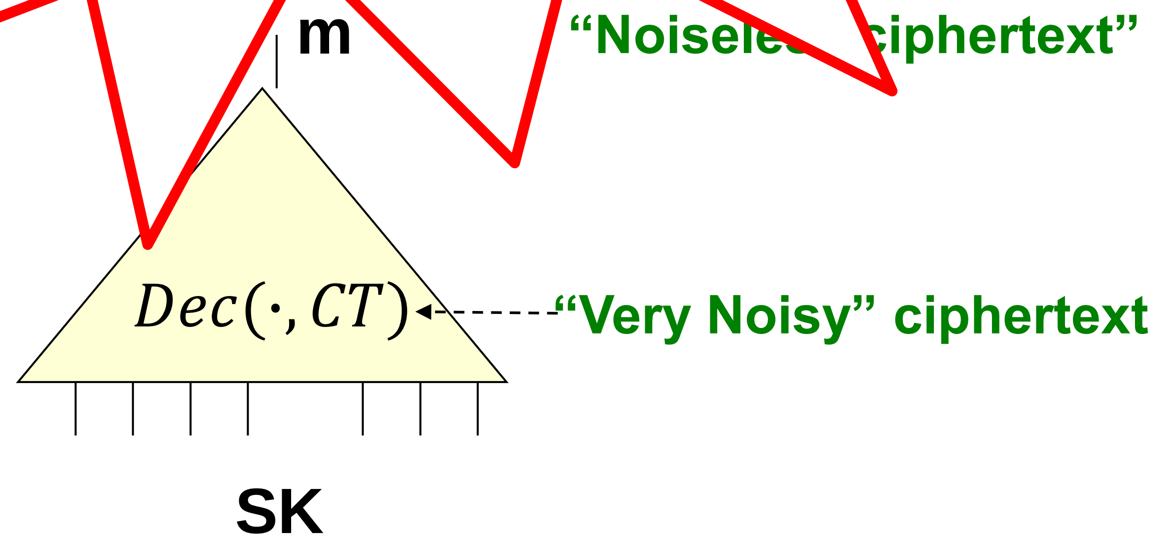
The cloud keeps homomorphically computing, but after a certain depth, the ciphertext is too noisy to be useful. What to do?

Idea: "Bootstrapping"!

Bo y

“Best Pos n!

But the evaluator/cloud does not have SK!



Decryption Circuit

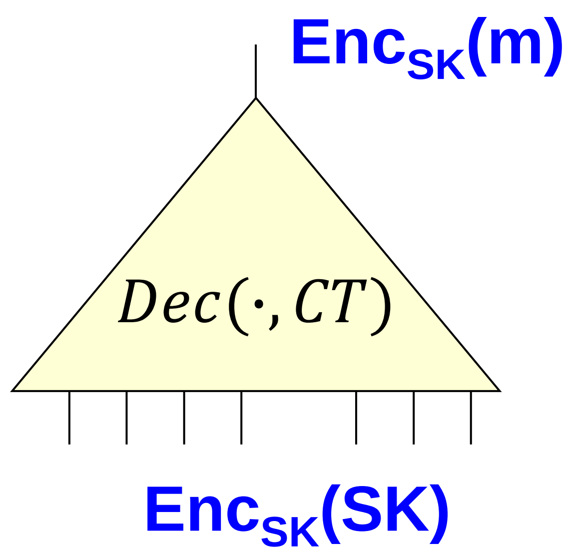
Bootstrapping, Concretely

Next Best = Homomorphic Decryption!

* 

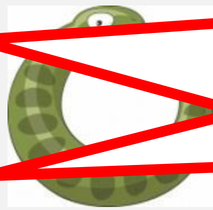
Assume server knows **ek = Enc$_{SK}$(SK)**.

(OK assuming the scheme is "circular secure")

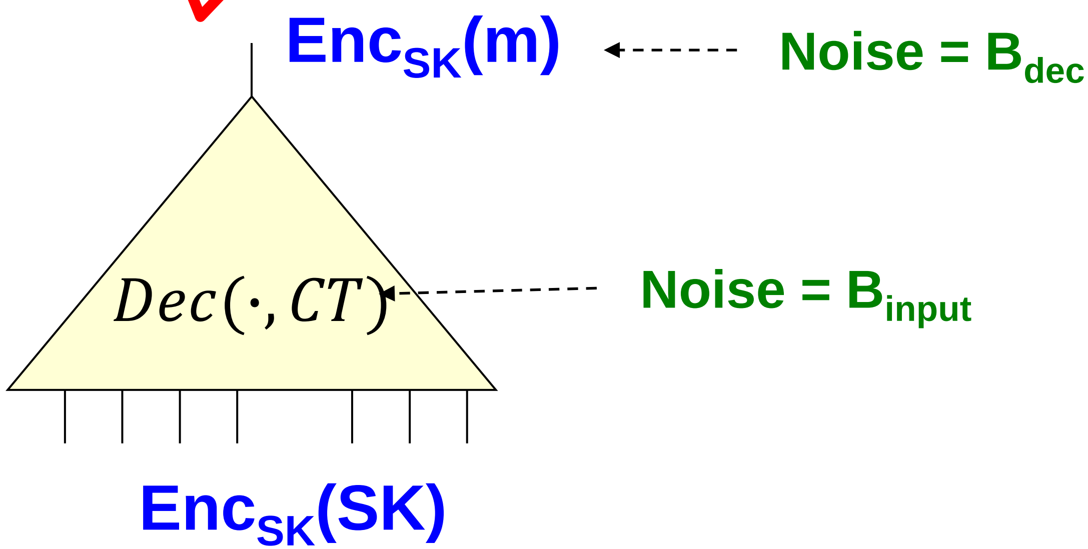


Bootstrapping, Concretely

Next Best = Homomorphic Decryption!

*  …SK). (… ircular secure")

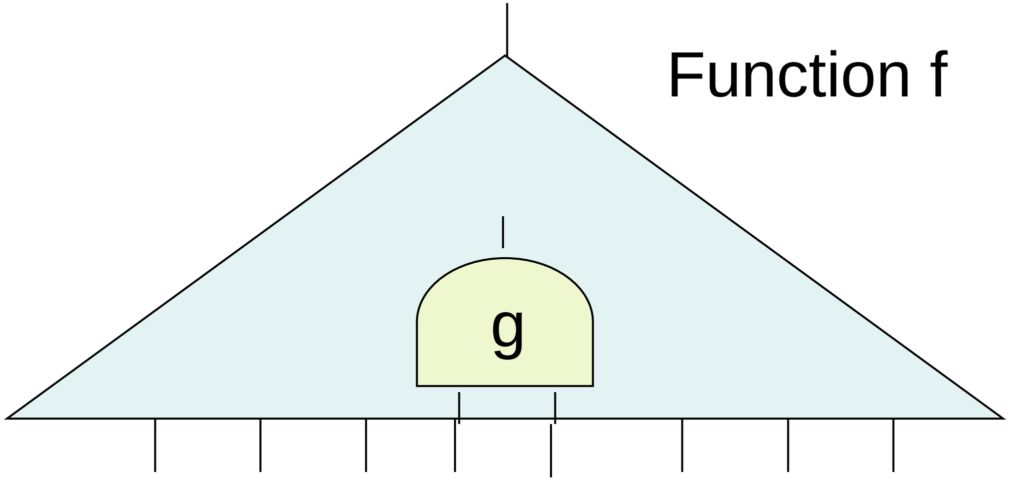
B_{dec} Independent of B_{input}



Wrap Up: Bootstrapping

Assume Circular Security:

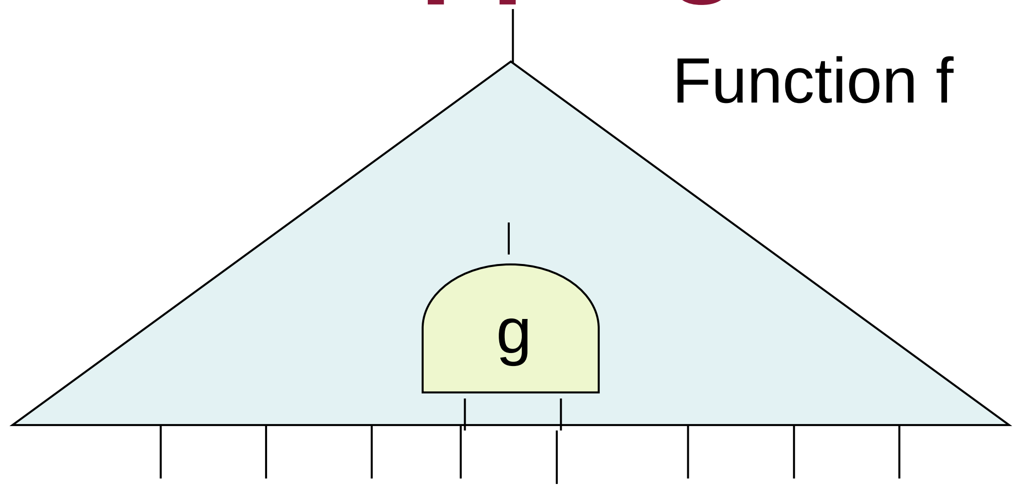
Evaluation key is $\mathbf{Enc_{SK}(SK)}$



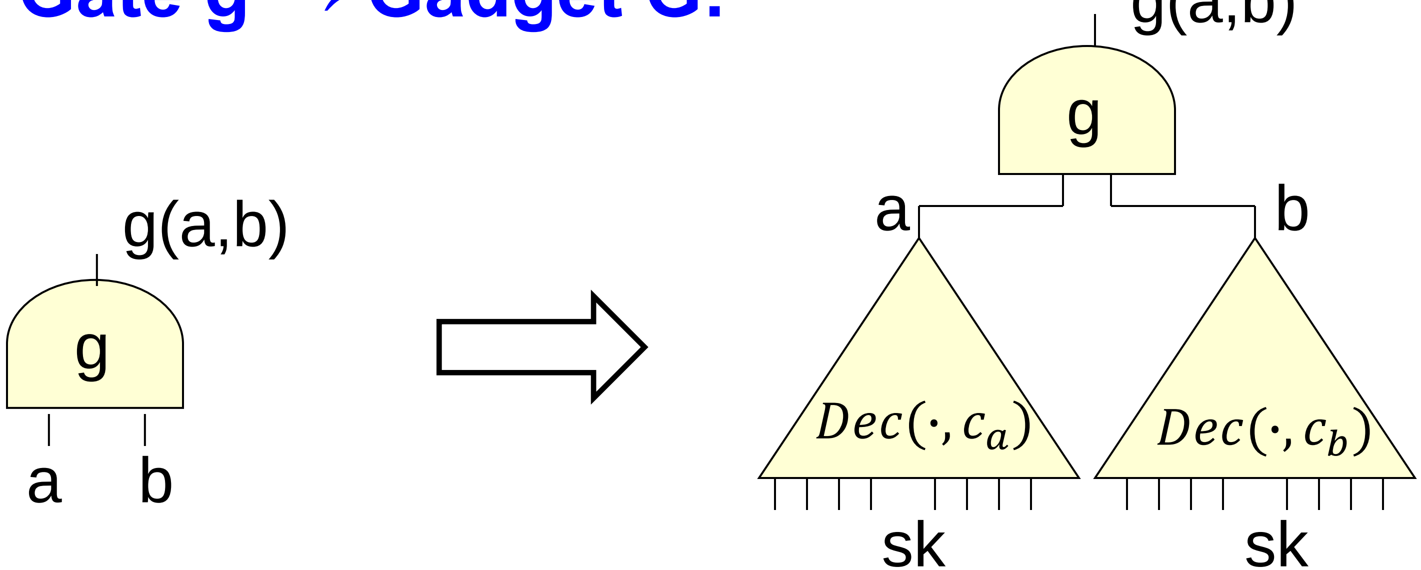
Wrap Up: Bootstrapping

Assume Circular Security:

Evaluation key is **$\text{Enc}_{SK}(SK)$**



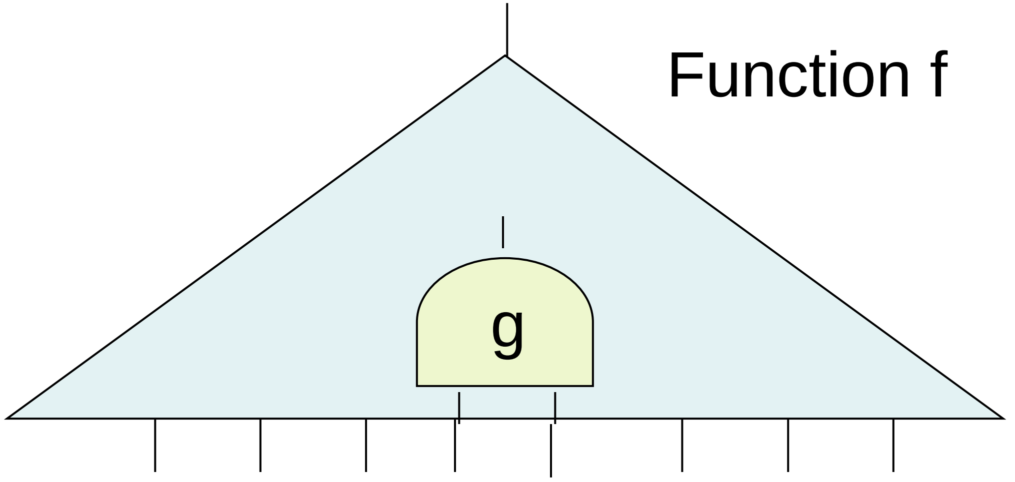
Each Gate g → Gadget G:



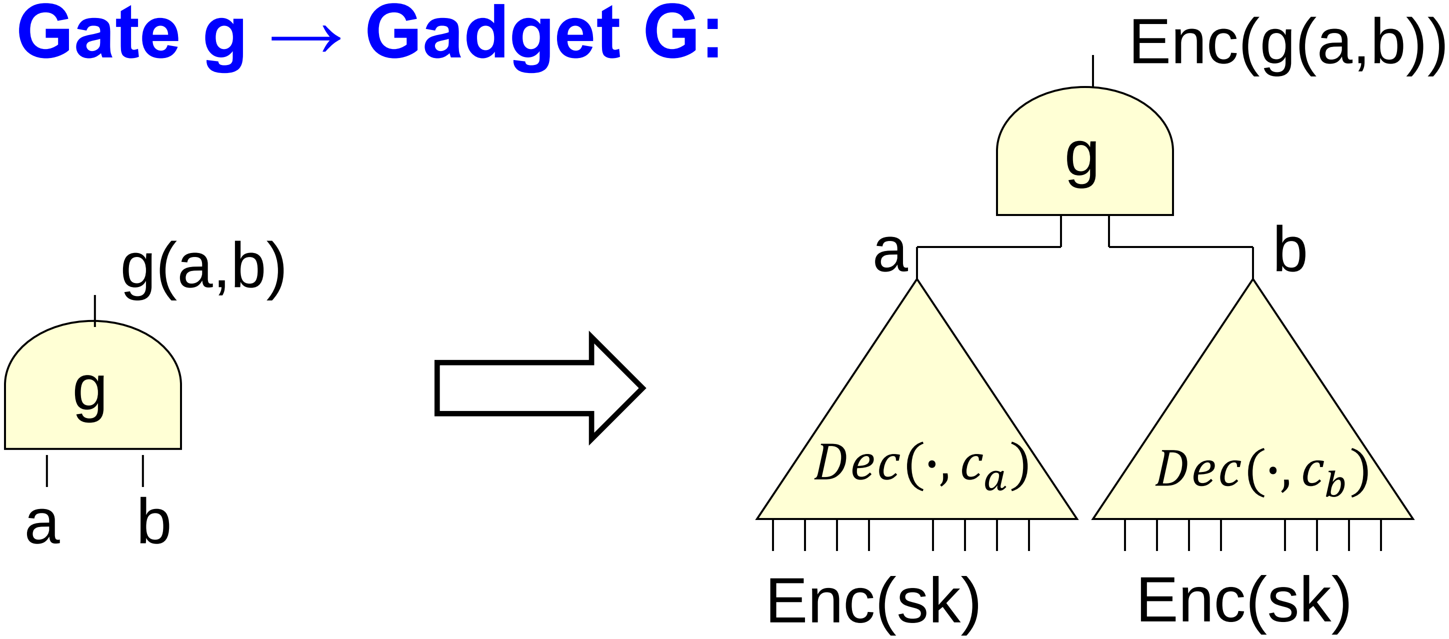
Wrap Up: Bootstrapping

Assume Circular Security:

Evaluation key is **$\mathbf{Enc_{SK}(SK)}$**

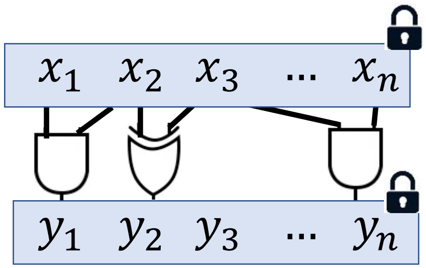


Each Gate g → Gadget G:



Subsequent Work: FHE in Practice

[Gentry-Halevi-Smart’12]: “FHE with Polylog Overhead”



Homomorphic computations “in place”.

SIMD computation + slot permutations (automorphisms)

“HELib”: The first homomorphic encryption library.

SEAL

PALISADE

HEEAN

FHEW

TFHE

Concrete

NFLLib

$\Lambda \circ \lambda$

Lattigo

cuFHE

FHE Bounty #1:

We have "leveled" FHE from the LWE assumption

$$sk_1 \xrightarrow{Enc_{pk_2}(sk_1)} sk_2 \xrightarrow{Enc_{pk_3}(sk_2)} sk_3 \xrightarrow{Enc_{pk_4}(sk_3)} \cdots \xrightarrow{Enc_{pk_L}(sk_{L-1})} sk_L$$

and "unbounded" FHE under a "circular secure" LWE assumption.

$$sk \quad Enc_{pk}(sk)$$

FHE Bounty #1: Why Circular Security?

Partial Answer:

[CLTV’15]: Unbounded FHE from indistinguishability obfuscation (IO).

+ [JLS’22]: Unbounded FHE from LPN + PRG in NC0 + Bilinear maps.

(Unbounded) FHE from LWE.

FHE Bounty #2:

Why Lattices/LWE?

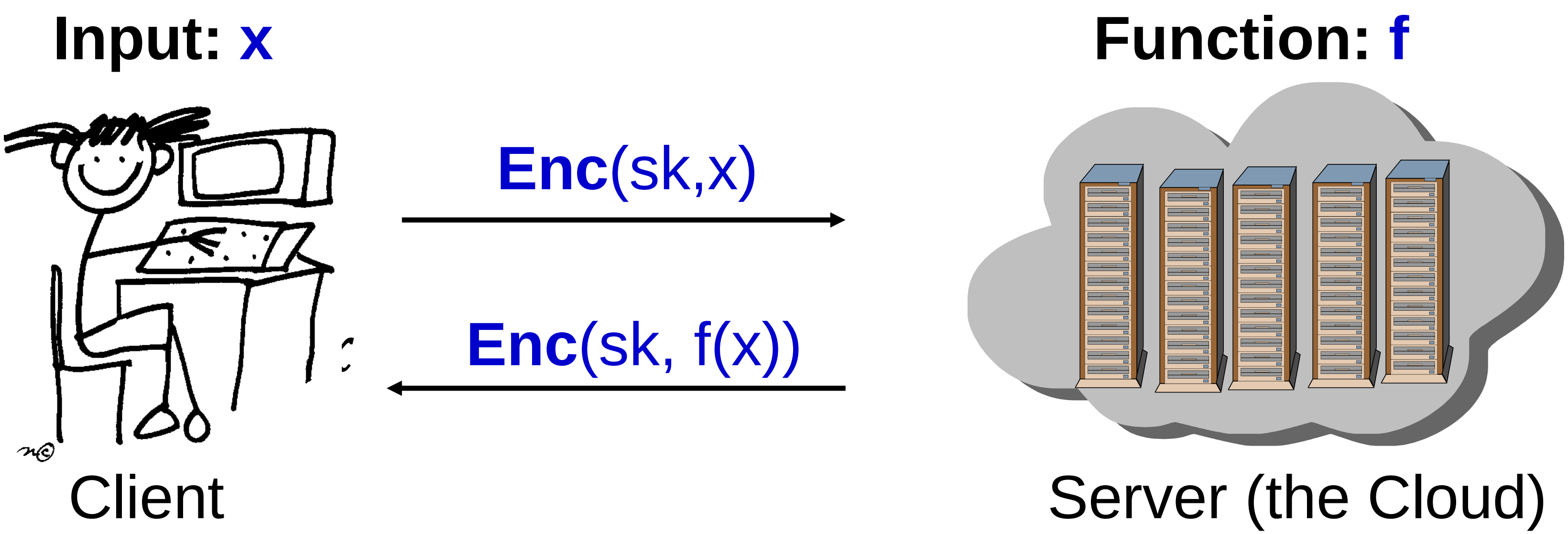
FHE from the Diffie-Hellman assumption.

FHE Bounty #3:
FHE ≈ as efficient as plaintext computation.

- **Advances in Rate-1 FHE**: FHE with ≈ 0 *communication* overhead
  [GH'19, BDGM'19]
- **Advances in Private Information Retrieval**:
  PIR with server computation ≈ 1 add + 1 mult per database byte* [CHHV'22]

If you solve truly practical FHE, you don't need my $100(0). ☺

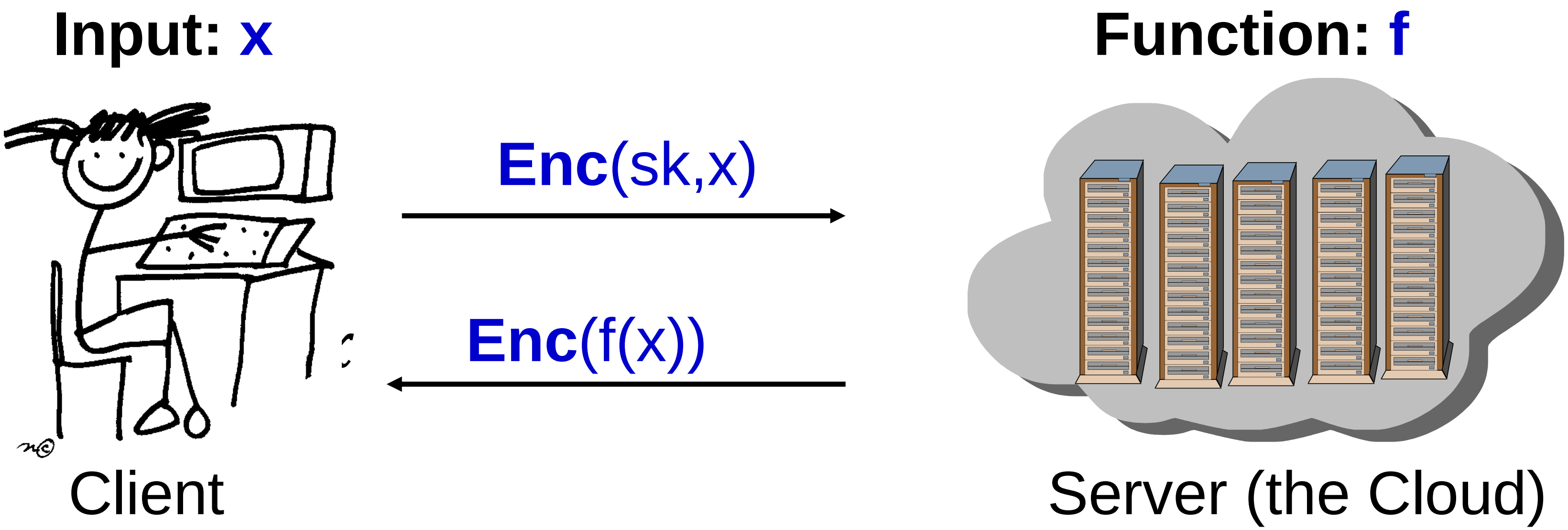
Unresolved Issue 1: Function Privacy



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

Security against a curious user?

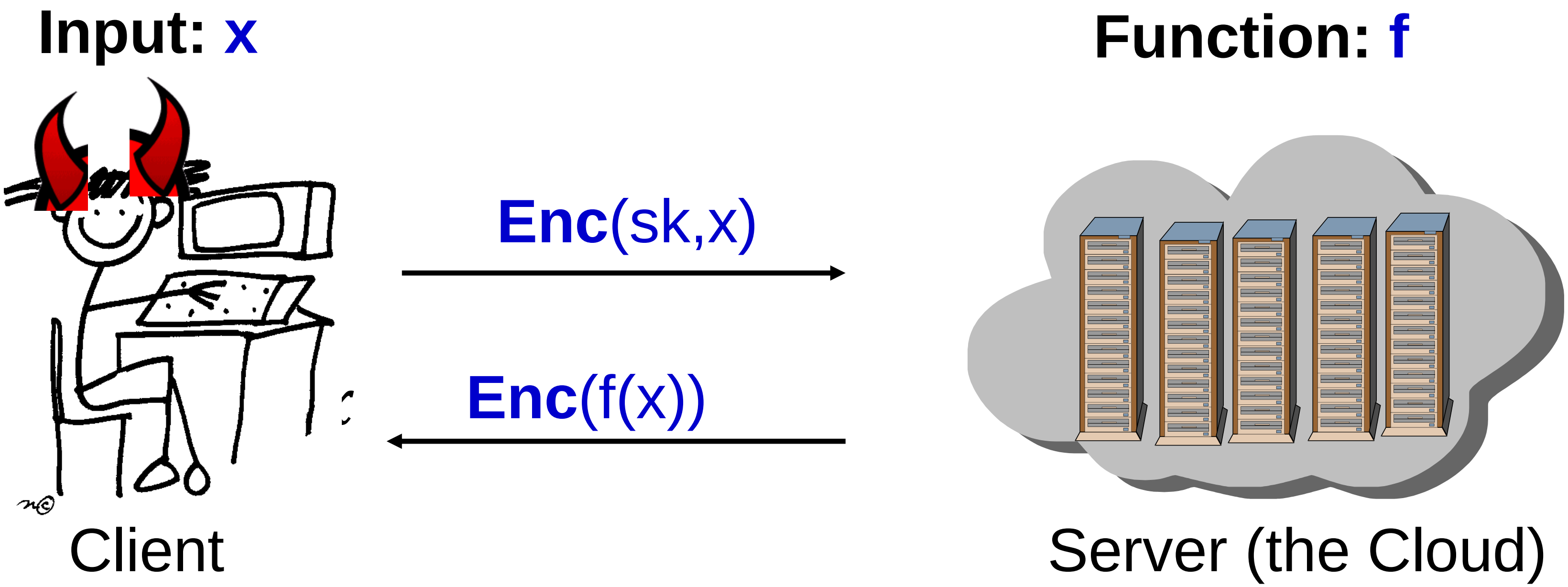
Unresolved Issue 1: Function Privacy



<u>Function Privacy</u>: Enc(f(x)) reveals no more information (about f) than f(x).

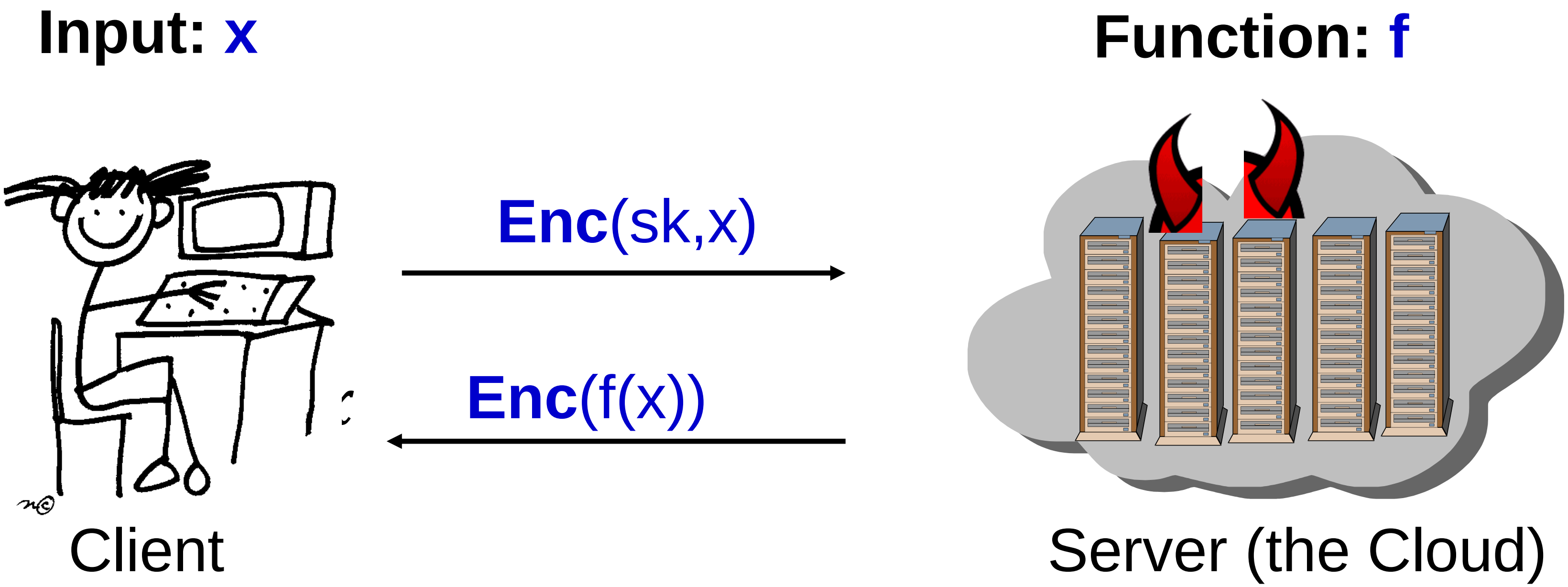
Function privacy via noise-flooding (on the board)

Unresolved Issue 2: Malicious Client



Idea: Use zero knowledge proofs.

Unresolved Issue 3: Malicious Cloud



Idea: “Succinct Interactive Proofs”. [Kilian92]