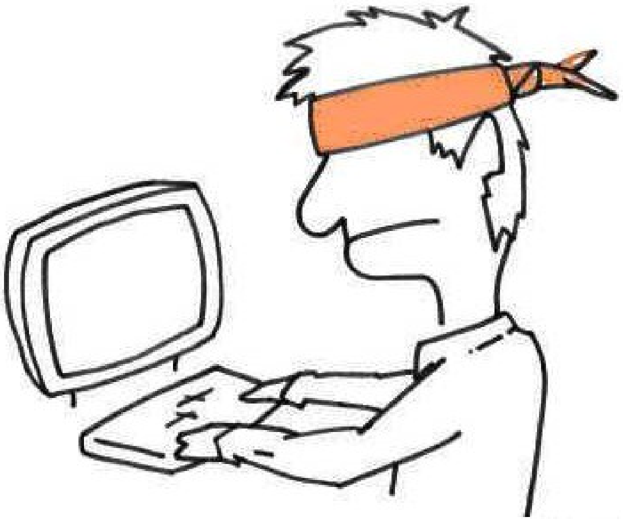
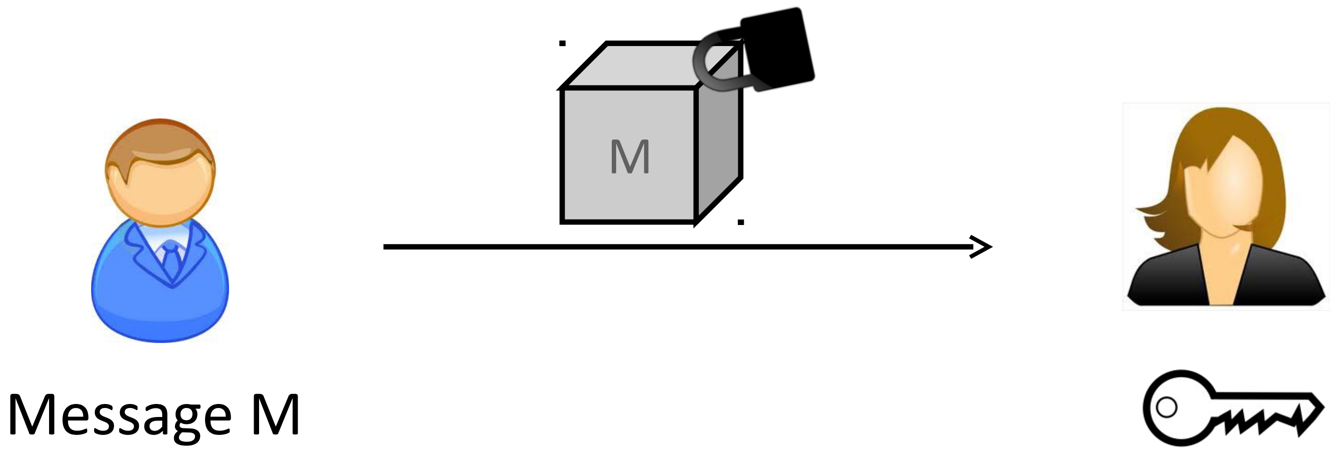
Computing with Encrypted Data

6.857 Lecture 26



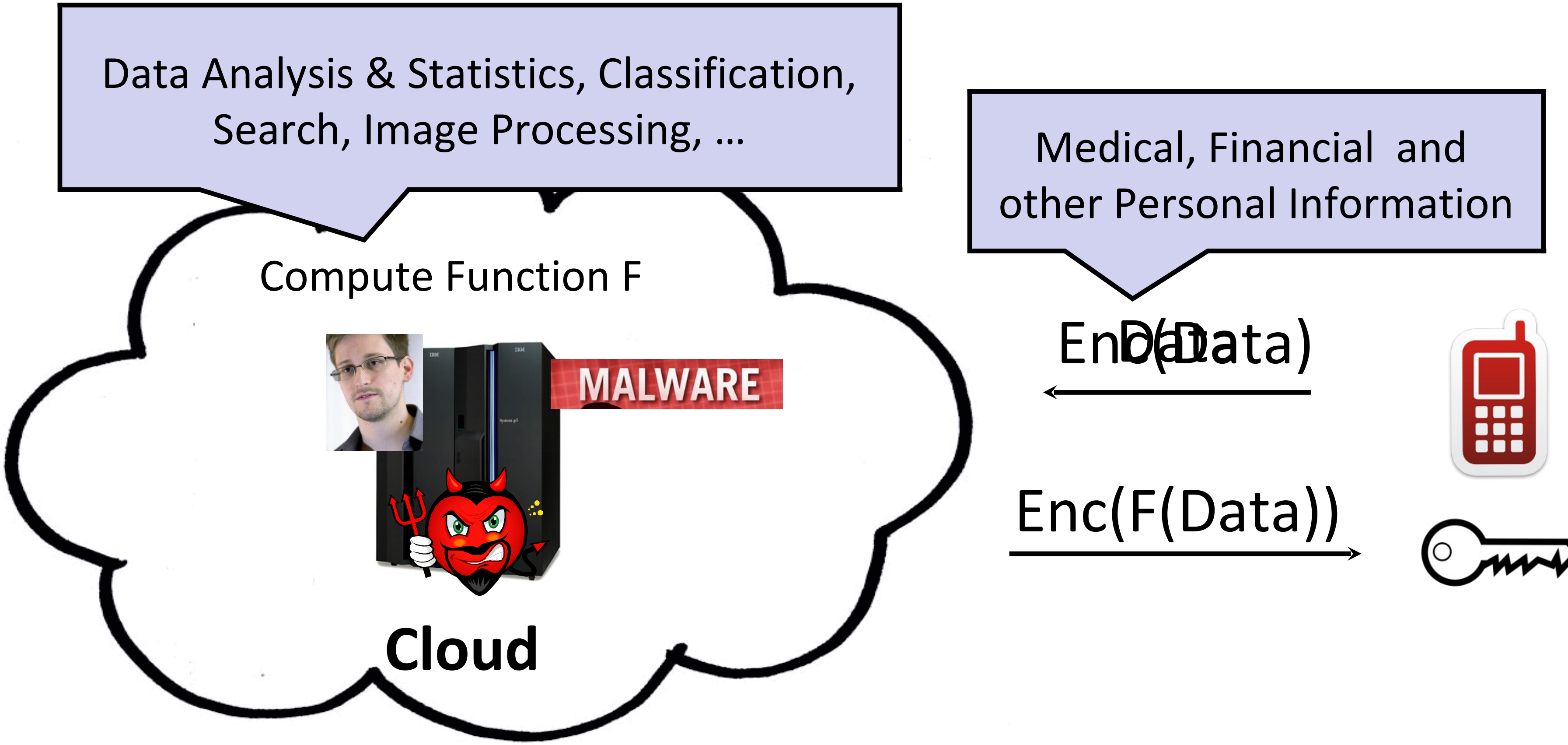
Encryption
for Secure Communication



All-or-nothing

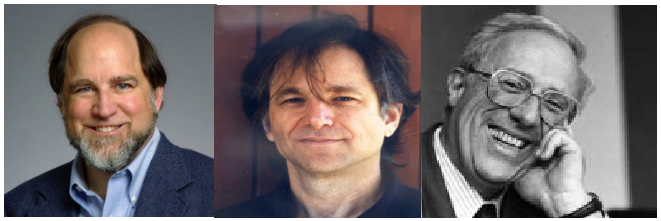
- **Have Private Key, Can Decrypt**
- **No Private Key, No Go**
  - cf. Non-malleable Encryption

Encryption
for Cloud Computing



Need: **Privacy** + **Functionality**

Fully Homomorphic Encryption



[Rivest, Adleman and Dertouzos'78]

Compute arbitrary functions on encrypted data?

ON DATA BANKS AND PRIVACY HOMOMORPHISMS

Ronald L. Rivest
Len Adleman
Michael L. Dertouzos

Massachusetts Institute of Technology
Cambridge, Massachusetts

Enc(data), F → Enc(F(data))

(fully = any function F)

(additive = only additions)

(multiplicative = only mult)

(somewhat = circuits of small depth)

[Gentry'09, BV'11, LTV'12]: Constructions of FHE

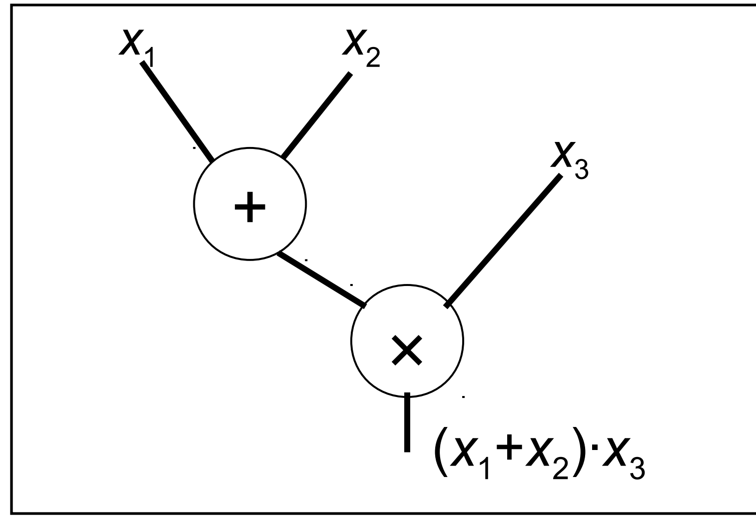
Outline

- Homomorphic Encryption
  - Multiplicative Homomorphism: El Gamal
  - Additive Homomorphism: Goldwasser-Micali
  - Fully Homomorphic Encryption: based on **NTRU**
- What I didn't tell you (and how to learn more)

FHE: The Big Picture

Function

Arithmetic Circuit



If we had:

- $\mathrm{Enc}(x_1)$, $\mathrm{Enc}(x_2) \Rightarrow \mathrm{Enc}(x_1+x_2)$
- $\mathrm{Enc}(x_1)$, $\mathrm{Enc}(x_2) \Rightarrow \mathrm{Enc}(x_1 \cdot x_2)$

then we are done.

Multiplicative Homomorphism

El Gamal Encryption

Setup: Group G of prime order p

(e.g., set of quadratic residues mod q where q = 2p+1)

Private key: $x \in Z_p$

Public key: generator g, $y = g^x \in G$

Enc(m_1): $(g^{r_1}, y^{r_1} \bullet m_1)$

Dec(m): Observe that $(g^{r_1})^x = y^{r_1}$

Multiplicative Homomorphism

El Gamal Encryption

Setup: Group G of prime order p

(e.g., set of quadratic residues mod q where q = 2p+1)

Private key: $x \in Z_p$

Public key: generator g, $y = g^x \in G$

Enc(m_1): $(g^{r_1}, y^{r_1} \bullet m_1)$ X

Enc(m_2): $(g^{r_2}, y^{r_2} \bullet m_2)$

$(g^{r_1+r_2}, y^{r_1+r_2} \bullet m_1 m_2)$

is an encryption of the product $m_1 m_2$

Additive Homomorphism

Goldwasser Micali Encryption

Public key: N, y: non-square mod N

Secret key: factorization of N

Enc(0): r^2 mod N,   Enc(1): y * r^2 mod N

square (0) * square (0) = square (0)

non-square (1) * square (0) = non-square (1)

square (0) * non-square (1) = square (1)

non-square (1) * non-square (1) = non-square (0)

XOR-homomorphic: Just multiply the ciphertexts

Other HE Schemes

- **Additively Homomorphic:**
  - Paillier
  - Damgard-Jurik (both addition of large numbers)

- **Additions + a single Multiplication:**
  - Boneh-Goh-Nissim (based on gap groups)
  - Gentry-Halevi-**V**. (based on lattices)

- **HE with Large ciphertext blowup:**
  - Sander-Young-Yung

How to Construct an FHE Scheme

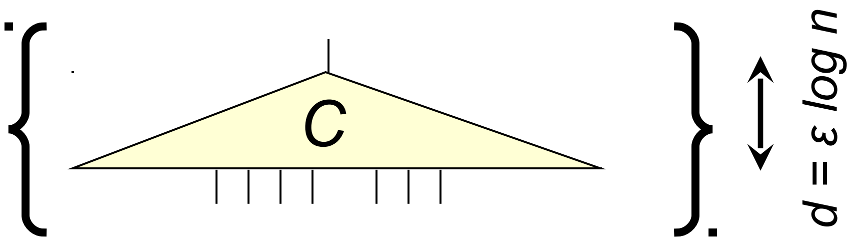
The Big Picture

STEP 1

“Somewhat Homomorphic” (SwHE) Encryption

[Gen09,DGHV10,SV10,BV11a,BV11b,BGV12,LTV12,GHS'12]

Evaluate arithmetic circuits of depth ***d = ε log n*** *



EVAL

* (0 < *ε* < 1 *is a constant, and n is the security parameter)*

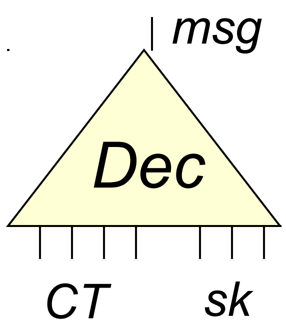
The Big Picture

STEP 2

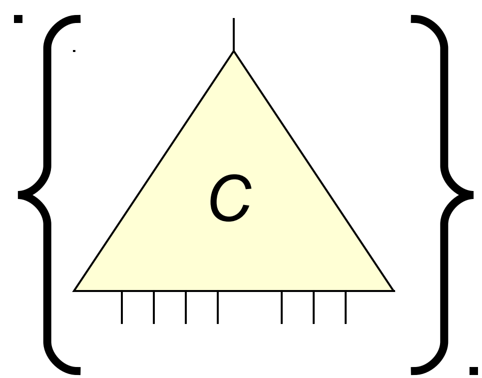
"Bootstrapping" Theorem [Gen09] (Qualitative)

"Homomorphic enough" Encryption $\Rightarrow^{*}$ FHE

Homomorphic enough =
Can evaluate its own Dec Circuit (plus some)



$\in^{(?)}$



Decryption Circuit

EVAL

The Big Picture

STEP 1

“Somewhat Homomorphic” (SwHE) Encryption

[Gen09,DGHV10,SV10,BV11a,BV11b,BGV12,LTV12,GHS’12]

Evaluate arithmetic circuits of depth $d = \varepsilon \log n$

STEP 3

Depth Boosting / Modulus Reduction [BV11b]

Boost the SwHE to depth $d = n^{\varepsilon}$

STEP 2

“Bootstrapping” Method

“Homomorphic enough” Encryption $\Rightarrow^*$ **FHE**

Homomorphic enough =

Can evaluate its own Dec Circuit (plus some)

The NTRU Encryption Scheme
[Hofstein-Pipher-Silverman'97]

Central characters: Polynomials mod q

- Polynomials of degree less than n (think n = 256)
- Coefficients over Z_q (think q = small prime)

- Addition: coefficient-wise
  $(6x^2+5x+10) + (5x^2+x+2) = 6$ (mod 11)

- Multiplication: polynomial mult, modulo an irreducible
  $(6x^2+5x+10) \times (5x^2+x+2) = 9x^3+x^2+9x+$
  (mod 11, x^4+1)

Ring $R_q := Z_q[x] / (x^n+1)$ (x^n+1 cyclotomic, q = 1 mod 2n prime)

The NTRU Encryption Scheme

- **KeyGen:**
  - Sample "small" polynomials $\boldsymbol{f}, \boldsymbol{g} \in \boldsymbol{R}_q$ (s.t. f=1 mod 2)

    coefficients ≤ B
  - Secret key SK=**f** and Public key PK=**h=2g/f**

Multiplying by f "kills" h

- **Encryption** $\mathrm{Enc}_{pk}(m)$,
  - Sample "small" polynomials **s, e** $\in \mathbf{R}_q$,
  - output $\mathbf{C = hs + 2e + m \quad (mod\ q,\ x^n+1)}$
- **Decryption** $\mathrm{Dec}_{sk}(C)$: Output (fC (mod q,x^n+1)) mod 2.
  - **Correctness:** $fC = f(hs+2e+m) = 2(gs+fe) + fm \quad (mod\ q,\ x^n+1)$

    If $|2(gs+fe) + fm| < q/2$, *taking mod 2 gives m.*

The NTRU Encryption Scheme

The "Small Polynomial Ratios" (SPR) Assumption:

Choose two polynomials f and g with "small" coefficients (of magnitude at most B). Then,

$$g/f \quad \cong_c \quad \text{uniform over } R_q$$

The key security parameter: The signal-to-noise ratio q/B

If q/B is too large (> 2^n), we can break NTRU in poly time.

Therefore, typical setting: q/B = $2^{n^{\varepsilon}}$ (for some ε << 1)

Theorem: The encryption scheme is secure under the SPR assumption

Additive Homomorphism

$c_1 = hs_1+2e_1+m_1$

$\mathbf{f.c_1 = 2E_1 + fm_1}$

$c_2 = hs_2+2e_2+m_2$

$\mathbf{f.c_2 = 2E_2 + fm_2}$

Add the ciphertexts: $\boldsymbol{c}_{\text{add}} = \boldsymbol{c_1} + \boldsymbol{c_2}$ ***(over*** $\boldsymbol{R_q}$***)***

$$
\begin{array}{rl}
 & f.c_1 = 2E_1 + fm_1 \\
+ & \\
 & f.c_2 = 2E_2 + fm_2 \\
\hline
\end{array}
$$

$$f.(c_1+c_2) = 2(\underbrace{E_1+E_2}_{E'}) + f.(m_1+m_2)$$

$$\Rightarrow \mathrm{Dec}_f(\boldsymbol{c}_{\text{add}}) = 2E' + f.(m_1+m_2) \text{ (mod 2)} = f.\ (m_1+m_2) \text{ (mod 2)}$$

$$= m_1+m_2 \quad (mod\ 2)$$

Multiplicative Homomorphism

$c_1 = hs_1+2e_1+m_1$ $\mathbf{f.c_1 = 2E_1 + fm_1}$	$c_2 = hs_2+2e_2+m_2$ $\mathbf{f.c_2 = 2E_2 + fm_2}$

Multiply the ciphertexts: $\boldsymbol{c}_{\text{mlt}} = \boldsymbol{c_1.c_2}$ ***(over R_q)***

$$\begin{array}{ll} & f.c_1 = 2E_1 + fm_1 \\ \mathbf{X} & f.c_2 = 2E_2 + fm_2 \\ \hline \end{array}$$

$$f^2.(c_1c_2) = 2\underbrace{(E_1m_2+E_2m_1+2E_1E_2)}_{E'} + f^2(m_1m_2)$$

$\Rightarrow \mathrm{Dec}_{f\wedge 2}(\boldsymbol{c}_{\text{mlt}}) = 2E' + f^2 m_1m_2 \text{ (mod 2)} = f^2 m_1m_2 \text{ (mod 2)}$

$= m_1m_2$ (mod 2)

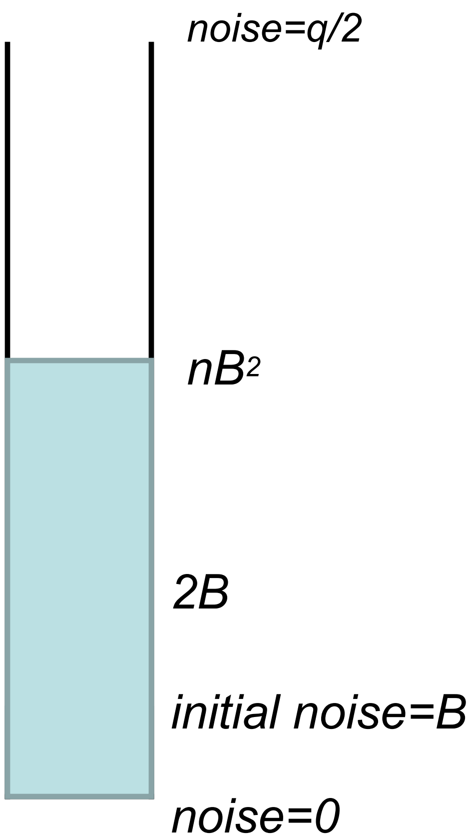
Noise Growth

- Assume the input ciphertext noise is at most B.

- Addition: norm of E_1+E_2 is at most 2B
- Multiplication: noise ≈ E_1E_2
  - Norm of E_1E_2 is at most nB^2

How Homomorphic is this:

The Reservoir Analogy



Additive Homomorphism: B → 2B

Multiplicative Homomorphism: B → nB^2

AFTER d LEVELS:

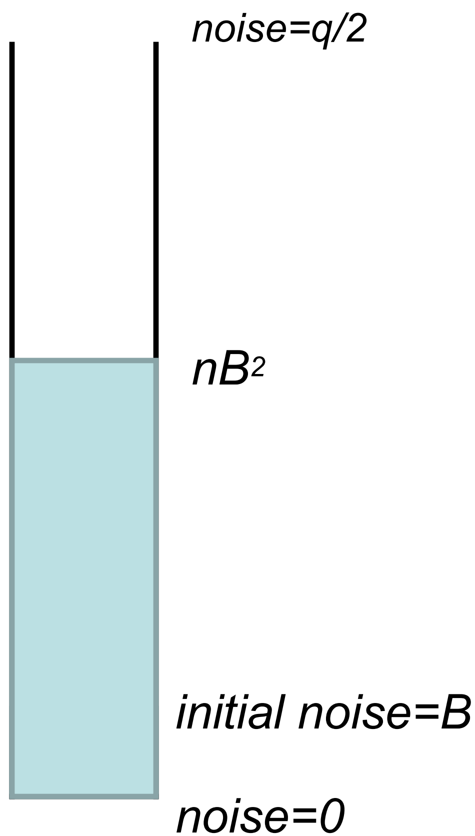
noise B → $(nB)^{2^d}$ (worst case)

$$(nB)^{2^d} \leq \frac{q}{2} \leq B.2^{n^{\varepsilon}}$$

SPR with q/B ratio $2^{n\text{^}\varepsilon}$

How Homomorphic is this:

The Reservoir Analogy



Additive Homomorphism: $B \rightarrow 2B$

Multiplicative Homomorphism: $B \rightarrow nB^2$

<u>AFTER d LEVELS:</u>

noise $B \rightarrow (nB)^{2^d}$ (worst case)

$$(nB)^{2^d} \leq \frac{q}{2} \leq B.2^{n^{\varepsilon}}$$

$$d \lesssim \log(\log q) - \log(\log nB)$$

$$\lesssim \varepsilon \log n - \log\log n$$

The Big Picture

STEP 1

“Somewhat Homomorphic” (SwHE) Encryption

[Gen09,DGHV10,SV10,BV11a,BV11b,BGV12,LTV12,GHS’12]

Evaluate arithmetic circuits of depth *$d = \varepsilon \log n$*

STEP 3

Depth Boosting / Modulus Reduction [BV11b]

Boost the SwHE to depth *$d = n^{\varepsilon}$*

STEP 2

“Bootstrapping” Method

“Homomorphic enough” Encryption $\Rightarrow^{*}$ FHE

Homomorphic enough =
Can evaluate its own Dec Circuit (plus some)

Bootstrapping

Bootstrapping Theorem [Gen09]

- If you can homomorphically evaluate depth d circuits *and*
- the depth of your decryption cicuit $< d$

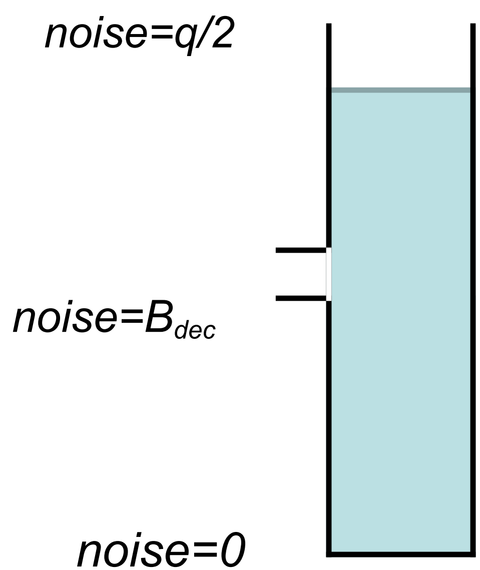
$\Rightarrow^*$ **FHE**

Bootstrapping

Bootstrapping Theorem [Gen09]

d-HE with decryption depth < *d* $\Rightarrow^*$ **FHE**

Bootstrapping = "Valve" at a fixed height
(that depends on decryption depth)



Say $n(B_{dec})^2 < q/2$

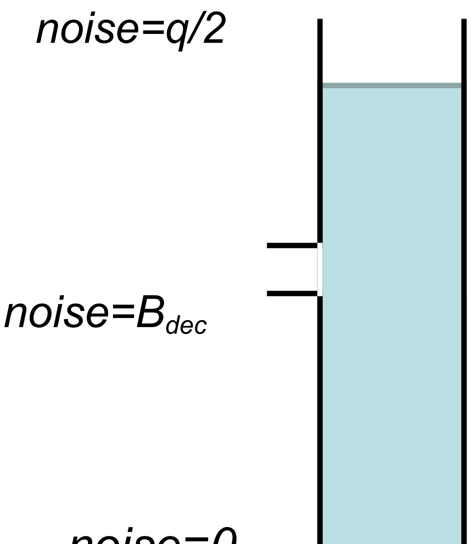
Bootstrapping

Bootstrapping Theorem [Gen09]

d-HE with decryption depth $< d$ $\Rightarrow^{*}$ **FHE**

Bootstrapping = "Valve" at a fixed height

(that depends on decryption depth)



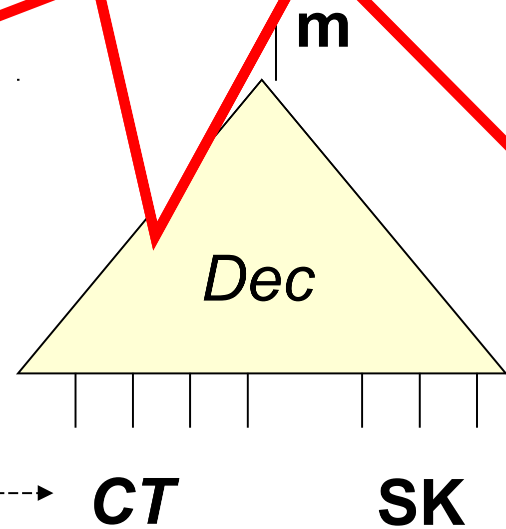
Say $n(B_{dec})^2 < q/2$

Bo

But the evaluator (cloud) does not have SK!

“Best Poss ... n!

“Noiseles ciphertext”



“Very Noisy” ciphertext

Decryption Circuit

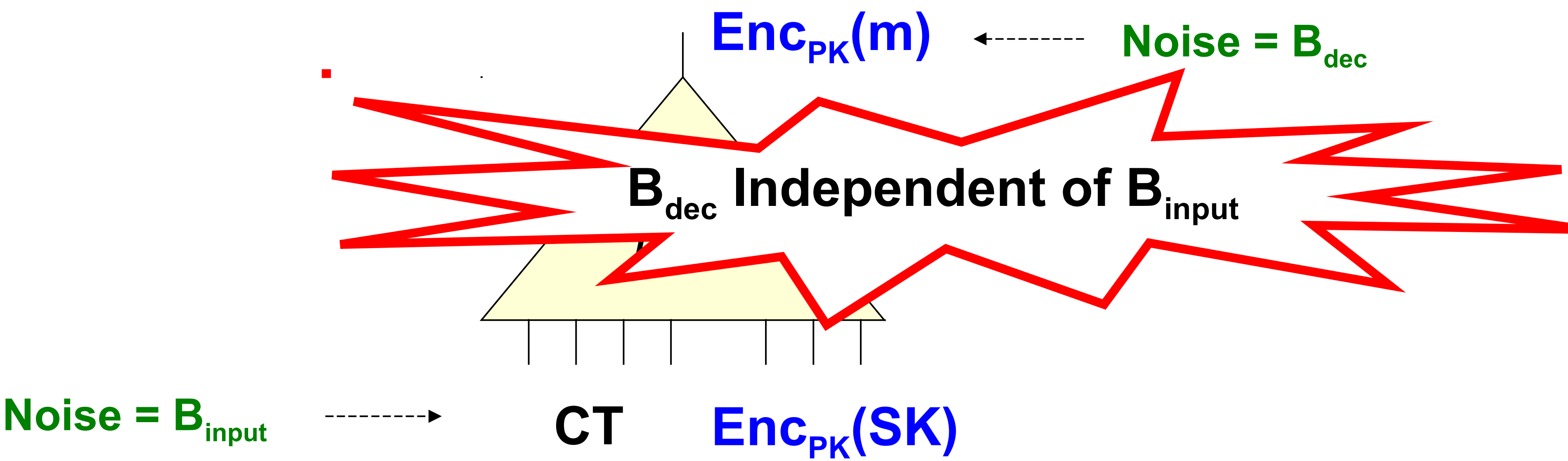
Bootstrapping, Concretely

Next Best = Homomorphic Decryption!

* 

Assume **Enc(SK)** is public.

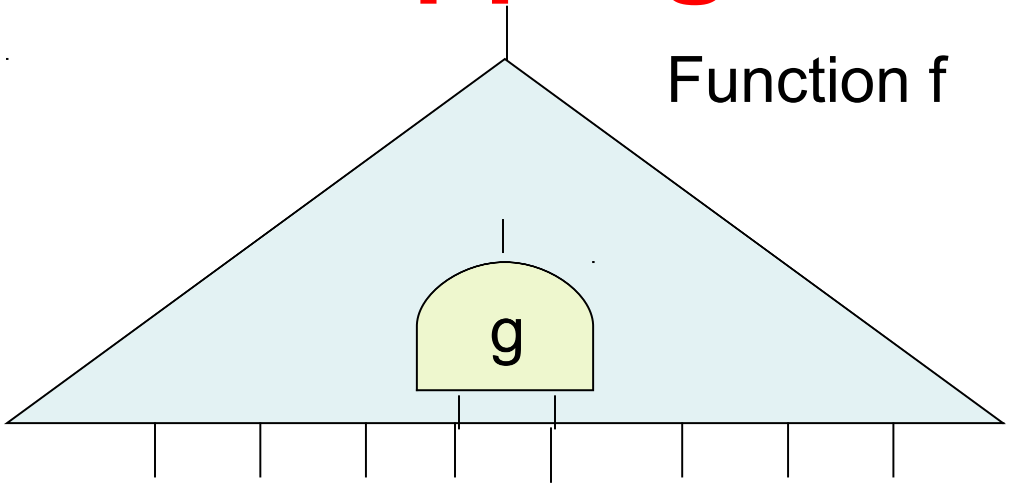
(OK assuming the scheme is "circular secure")



Wrap Up: Bootstrapping

Assume Circular Security:

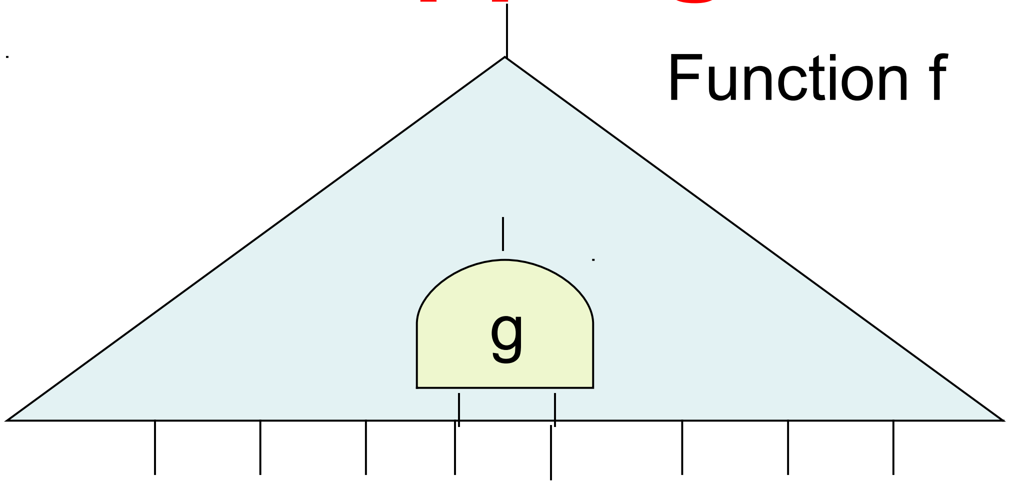
Public key contains $\mathbf{Enc_{SK}(SK)}$



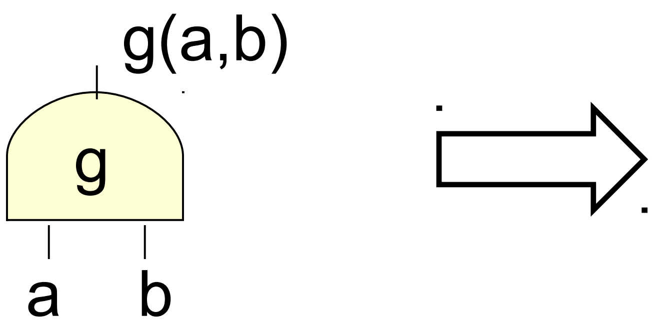
Wrap Up: Bootstrapping

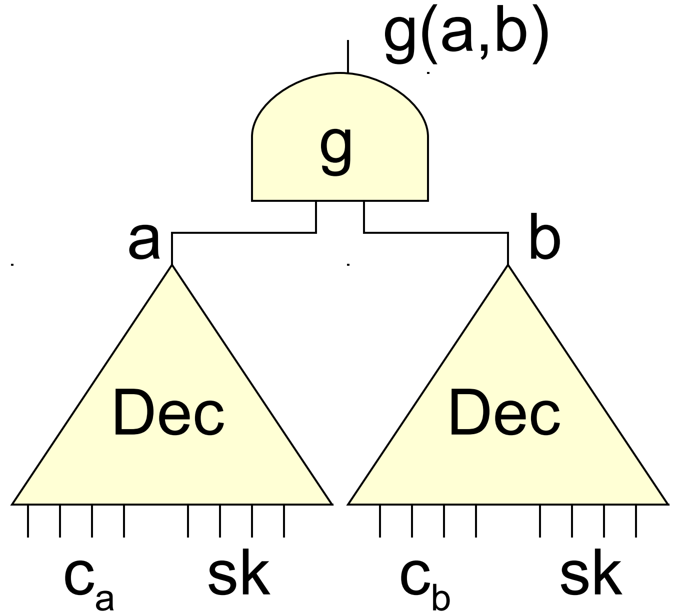
Assume Circular Security:

Public key contains **$Enc_{SK}(SK)$**



Each Gate g → Gadget G:

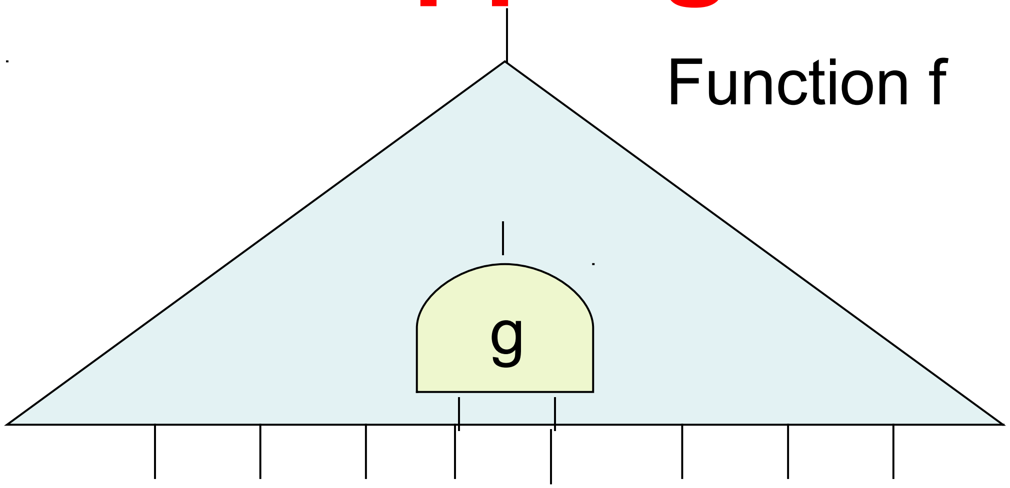




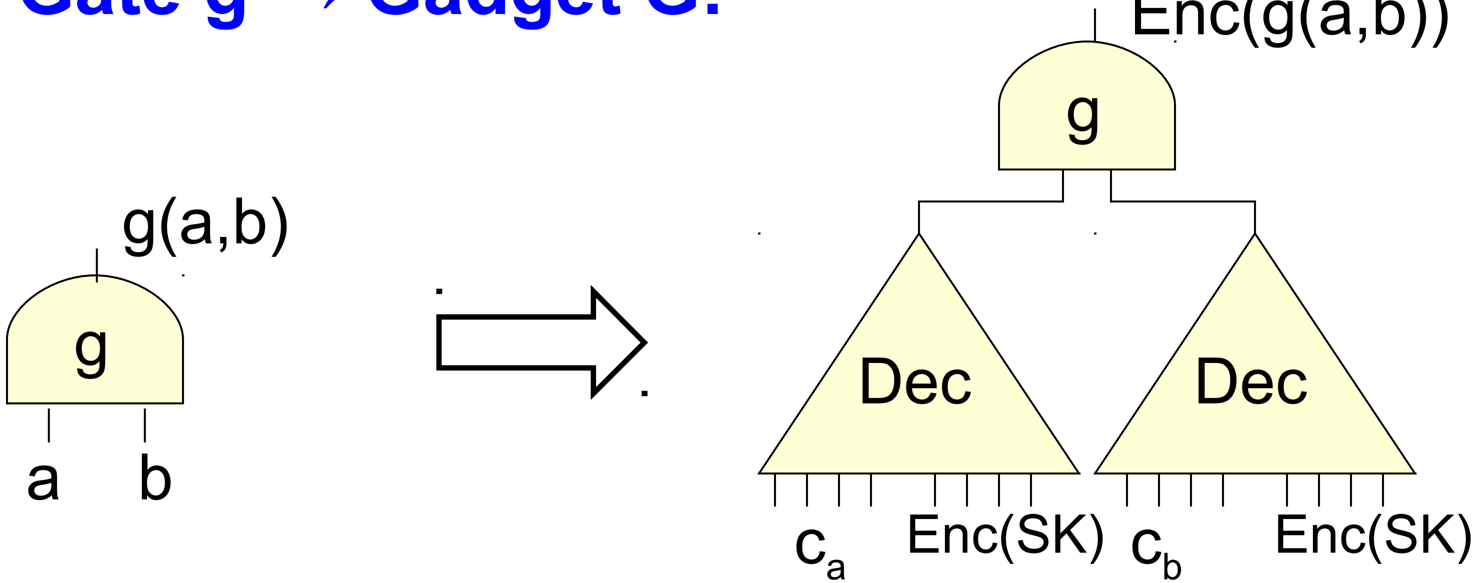
Wrap Up: Bootstrapping

Assume Circular Security:

Public key contains **$Enc_{SK}(SK)$**



Each Gate g → Gadget G:



Wrap-up: FHE

STEP 1

"Somewhat Homomorphic" (SwHE) Encryption

[Gen09,DGHV10,SV10,BV11a,BV11b,BGV12,LTV12,GHS'12]

Evaluate arithmetic circuits of depth ***$d = \varepsilon \log n$***

STEP 3

Depth Boosting / Modulus Reduction [BV11b]

Boost the SwHE to depth ***$d = n^{\varepsilon}$***

STEP 2

"Bootstrapping" Method

"Homomorphic enough" Encryption $\Rightarrow^*$ FHE

Homomorphic enough =
Can evaluate its own Dec Circuit (plus some)

Boosting Depth from log n to n^{ε}

(in one slide)

- The culprit: Multiplication
- Increases noise from B to $nB^2 \gg B$

- Let us pause for a moment. Is $nB^2 > B$?

- … Not if $B < 1$

- Why not scale everything by q, and work over (0,1)?
- Quite amazingly, this works out and gives us an error growth of nB (no squaring)

Wrap-up: FHE

STEP 1

"Somewhat Homomorphic" (SwHE) Encryption

[Gen09,DGHV10,SV10,BV11a,BV11b,BGV12,LTV12,GHS'12]

Evaluate arithmetic circuits of depth $d = \varepsilon \log n$

STEP 3

Depth Boosting / Modulus Reduction [BV11b]

Boost the SwHE to depth $d = n^{\varepsilon}$

STEP 2

"Bootstrapping" Method

"Homomorphic enough" Encryption $\Rightarrow^{*}$ FHE

Homomorphic enough =
Can evaluate its own Dec Circuit (plus some)

What We Didn't Do

A Lot!

– Functional Encryption

– Software Obfuscation: how to encrypt programs

– Practical techniques for computing on encrypted data: searchable encryption, deterministic encryption,…

– Secure Multiparty protocols, …

Come to 6.892!

Thanks!

Good luck with the project write-ups!