CS 528 (Fall 2021)
Data Privacy & Security

Yuan Hong
Department of Computer Science
Illinois Institute of Technology

Chapter 7
Homomorphic Encryption

OUTLINE

- Introduction & History
- Partially HE
- Fully HE

A BRIEF HISTORY OF CRYPTO

If you had the key, you could **encrypt…**

Julius Ceasar (100-44 BC)

DWWDFN DW GDZQ

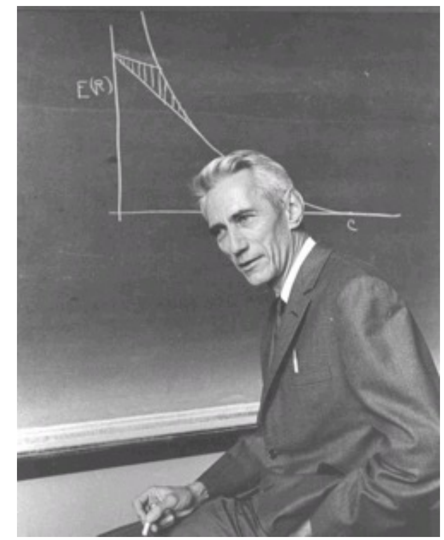
Message:	**ATTACK AT DAWN**
Key: +3	↓↓↓↓↓↓ ↓↓ ↓↓↓↓
Ciphertext:	**DWWDFN DW GDZQ**

A BRIEF HISTORY OF CRYPTO

1900-1950

Vigenere

Enigma



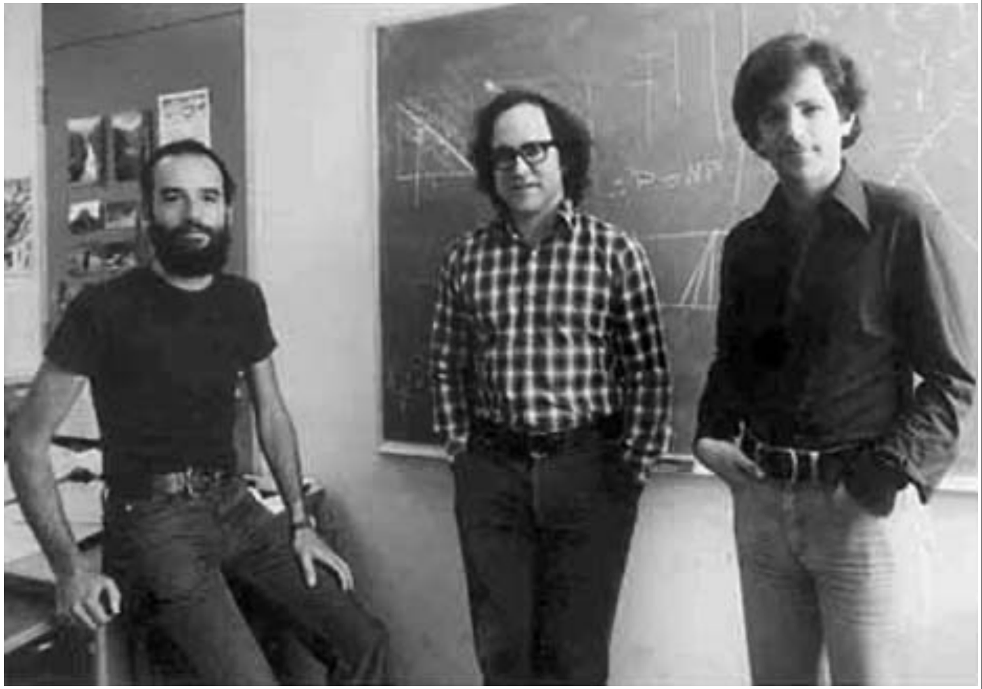
Claude Shannon and Information Theory

Symmetric Encryption:

Encryption and Decryption use the same key

A BRIEF HISTORY OF CRYPTO
(1970s)

Merkle, Hellman and Diffie
(1976)

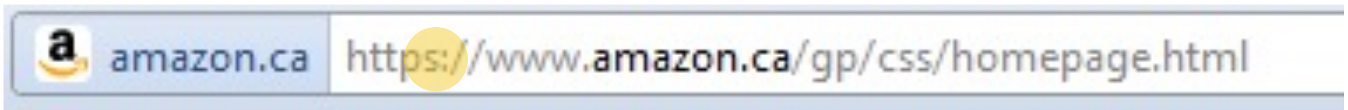


Shamir, Rivest and Adleman
(1978)

Asymmetric Encryption

Encryption uses a public key, Decryption uses the secret key

A BRIEF HISTORY OF CRYPTO



Asymmetric Encryption: The Foundation of E-Commerce

ILLINOIS INSTITUTE OF TECHNOLOGY

A BRIEF HISTORY OF CRYPTO

RSA: The first and most popular asymmetric encryption

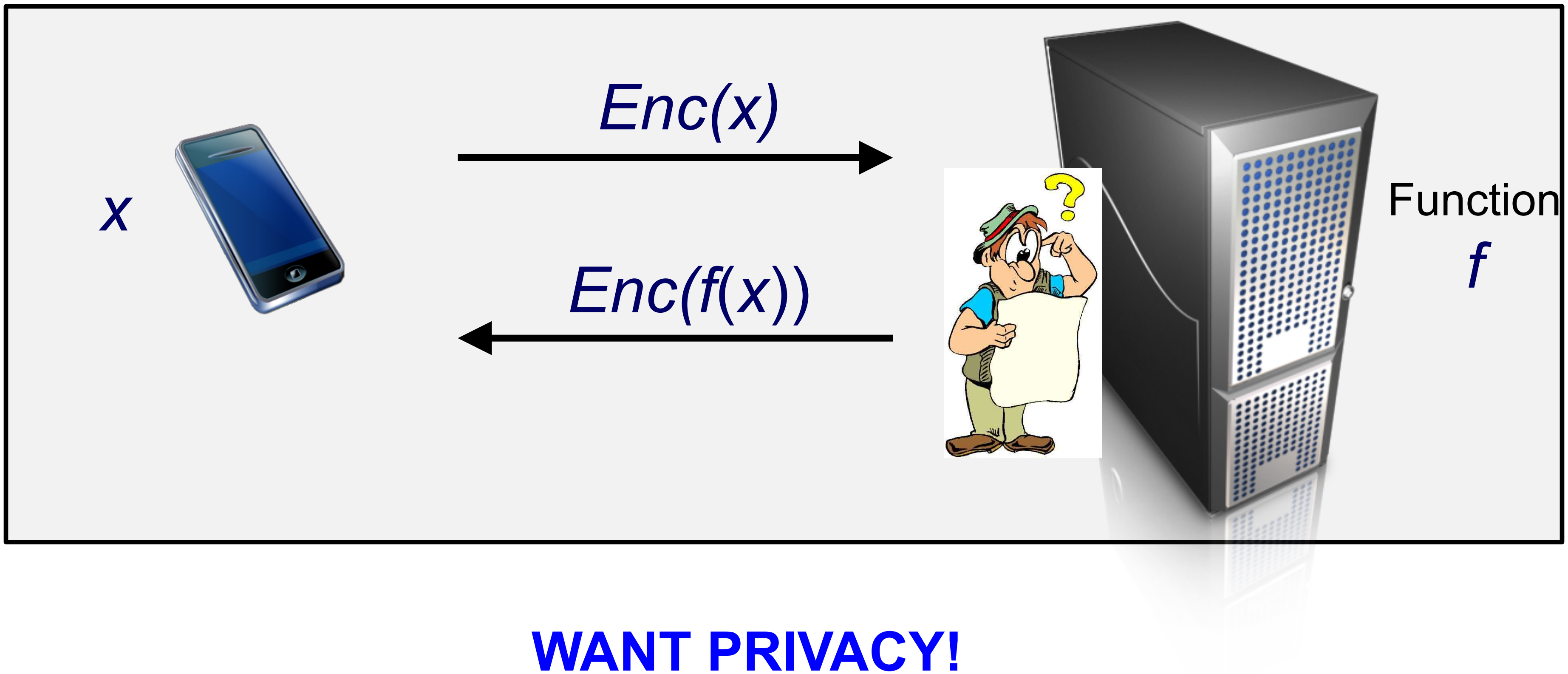
$$E(m) = m^e \pmod{n}$$

$$\mathrm{D}(c) = c^d \pmod{n}$$

COMPUTING ON ENCRYPTED DATA

ILLINOIS INSTITUTE OF TECHNOLOGY

What else can we do with encrypted data, anyway?



WANT PRIVACY!

COMPUTING ON ENCRYPTED DATA

ILLINOIS INSTITUTE OF TECHNOLOGY

Some people noted the algebraic structure in RSA…

$$E(m_1) = m_1{}^e \qquad E(m_2) = m_2{}^e$$

Ergo … $E(m_1) \times E(m_2)$

$$= m_1{}^e \times m_2{}^e$$

$$= (m_1 \times m_2)^e$$

$$= E(m_1 \times m_2)$$

Multiplicative Homomorphism

$$\boxed{E(m_1) \times E(m_2) = E(m_1 \times m_2)}$$

COMPUTING ON ENCRYPTED DATA

RSA is multiplicatively homomorphic

(but not additively homomorphic)

$$E(m_1) = {m_1}^e \qquad E(m_2) = {m_2}^e$$

Ergo … $E(m_1) \times E(m_2)$

$$= {m_1}^e \times {m_2}^e$$

$$= (m_1 \times m_2)^e$$

$$= E(m_1 \times m_2)$$

Multiplicative Homomorphism

$$\boxed{E(m_1) \times E(m_2) = E(m_1 \times m_2)}$$

HOMOMORPHIC ENCRYPTION

Common notations:

pk – public key

sk – secret key

m – message

c – ciphertext

$c = Encrypt_{pk}(m)$

$m = Decrypt_{sk}(c)$

$E_m(pk)$ - encryption algorithm as a circuit

$D_m(sk)$ - decryption algorithm as a circuit

f – is the function or circuit that we want to evaluate on plaintext

F – is the function or circuit that corresponds to f and operates on ciphertext in the cryptosystem

$$c_1 = Encrypt_{pk}(m_1)$$

...

$$c_n = Encrypt_{pk}(m_n)$$

$$c = F(c_1, \ldots, c_n)$$

$$m = Decrypt_{sk}(c)$$

OUTLINE

- Introduction & History
- Partially HE
- Fully HE

PARTIALLY HE

Multiplicative Partially HE
Unpadded RSA
pk=(n,e)
$c=E_{pk}(m)=m^e \bmod n$

$c_1 * c_2 = m_1^e m_2^e \bmod n = E_{pk}(m_1 * m_2)$

Additive Partially HE
Paillier scheme
pk=(n,g)
$c=E_{pk}(m)=g^m r^n \bmod n^2$
r in {0,…,n-1} – some random value

$c_1 * c_2 = (g^{m1} r1^n) * (g^{m2} r_2^n) \bmod n =$
$g^{m1+m2} (r_1 r_2)^n \bmod n = E_{pk}(m_1 + m_2)$

HOMOMORPHIC ENCRYPTION

Homomorphic encryption is a form of encryption that allows computation on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext. [from Wikipedia]

from E[A], E[B], can compute E[f(A,B)]

- e.g. f can be +, ×, xor, …

Ideally, want f = {+,×}

- Can do universal computation on ciphertext!

SO, WHAT SCHEME CAN I USE?

Sorry!

- Doesn't exist yet (NOTE: in practical form!)
- Latest result by Gentry gives such a scheme

Long standing open problem [RAD78]

Existing schemes homomorphic to 1 function

E.g. ElGamal (×), Paillier (+), GM (xor)

But some progress …

- Homomorphic encryption scheme that supports one × and arbitrary +.

Even standard homormorphic schemes very useful!

System	Plaintext operation	Cipher operation
RSA	×	×
Paillier	+, − m×k, m+k	×, ÷ c^k, $c \times g^k$
ElGamal	× m×k, m^k	× c×k, c^k
Goldwasser-Micali	⊕	×
Benaloh	+, −	×, ÷
Naccache-Stern	+, − m×k	×, ÷ c^k
Sander-Young-Yung	×	+
Okamoto-Uchiyama	+, − m×k, m+k	×, ÷ c^k, c+e(k)
Boneh-Goh-Nissim	Paillier (+, −, m×k, m+k) × (once)	Paillier bilinear pairing
US 7'995'750 / ROT13	+	+

HE USAGE SCENARIOS

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Cloud Computing:** storage, computation, search query
- **Spam filtering:** Blacklisting encrypted mails.
- **Medical Applications (Private data, Public functions):** search, cloud computation of certain functions (patient's condition, etc) on behalf of the patient. Analyze disease/ treatment without disclosing them. Search for DNA markers without revealing DNA
- **Financial Applications (Private data, Private functions):** computations on encrypted data such as data about companies, their stock price, or their performance or inventory. The customer may upload encrypted program/function to compute on encrypted data.
- **Advertising and Pricing:** Assume a customer has a mobile phone and uploads his data, such as location, email, time of the day, browsing activity, stream from the camera, etc. The advertising company computes some function to decide which ad is to be sent back to the client.
- **Electronic voting:** need to calculate the result of the voting without decrypting any ballots. More properties are there…
- **Data Mining:** HE solution is both fully private and fully accurate. Allows data miner to compute frequencies of values on the customer data, without revealing the data itself.
- **Biometric Authentication:** relation between biometric trait and personal identity must be hidden, i.e., comparison of the trait and the database fields must be encrypted.

ILLINOIS INSTITUTE OF TECHNOLOGY

REAL-LIFE USAGE

Helios: Web-Based Open-Audit Voting
Ben Adida, Harvard University

- Real-life HE application.
- Anyone can create and run an election
- Any willing observer can audit the entire process
- Uses HE property of the ElGamal cryptosystem to calculate the tally without decrypting of any ballots.

SECURE SCALAR PRODUCT

Alice has a vector A = {a1,a2,…,an}

Bob has a vector B = {b1,b2,…,bn}

Would together like to compute A·B

Can use homomorphic encryption to do it?

OUTLINE

- Introduction & History
- Partially HE
- Fully HE

DOES FHE EVER EXIST

Fully Homomorphic Encryption (FHE). Some Properties.

FHE property (simplified):

- $\text{Decrypt}_{sk}(c_1 * c_2) = m_1 * m_2$
- $\text{Decrypt}_{sk}(c_1 + c_2) = m_1 + m_2$

I.e.:

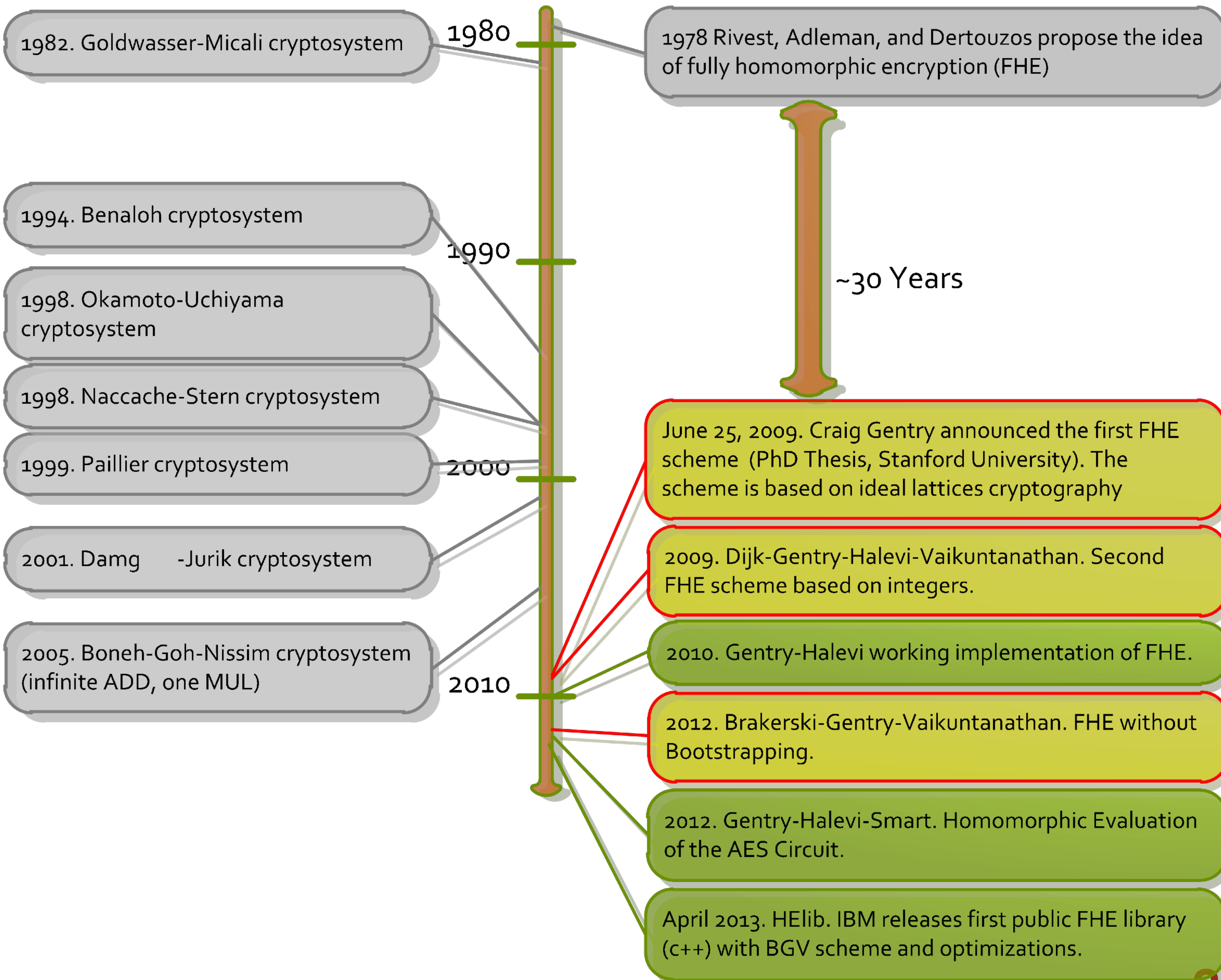
$$\text{Decrypt}_{sk}(F(c_1, \ldots, c_n)) = F(m_1, \ldots, m_n)$$

FHE may support another set of operations to support a ring of plaintexts. Examples: AND, XOR

FHE can be:

- Public key schemes
- Symmetric key schemes

“HOLY GRAIL” FOR 30 YEARS



TYPES OF HE SCHEMES

Homomorphic Encryption (HE) = type of computation for a set of functions $f(m_1,\ldots,m_n)$ carried on ciphertexts $Enc(m_1)\ldots Enc(m_n)$ with a corresponding function F such that

$$f(m_1,\ldots,m_n) = Dec(F(Enc(m_1),\ldots,Enc(m_n)))$$

Partially HE (PHE) = HE scheme where only one type of operations is possible (multiplication or addition)

Somewhat HE (SHE) = HE scheme that can do a **limited** number of additions and multiplications

Fully HE (FHE) = HE scheme that can perform an **infinite** number of additions and multiplications

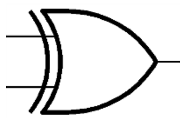
… until, in October 2008 …

… ***Craig Gentry*** came up with the first fully homomorphic encryption scheme …

COMPUTING ON ENCRYPTED DATA

Why SUMs and PRODUCTs?

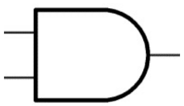
SUM
=



XOR (add mod 2)

0 XOR 0	0
1 XOR 0	1
0 XOR 1	1
1 XOR 1	0

PRODUCT
=



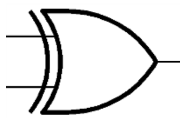
AND (multi mod 2)

0 AND 0	0
1 AND 0	0
0 AND 1	0
1 AND 1	1

COMPUTING ON ENCRYPTED DATA

Because {XOR,AND} is Turing-complete …

… any function is a combination of XOR and AND gates



XOR (add mod 2)

0 XOR 0	0
1 XOR 0	1
0 XOR 1	1
1 XOR 1	0

AND (multi mod 2)

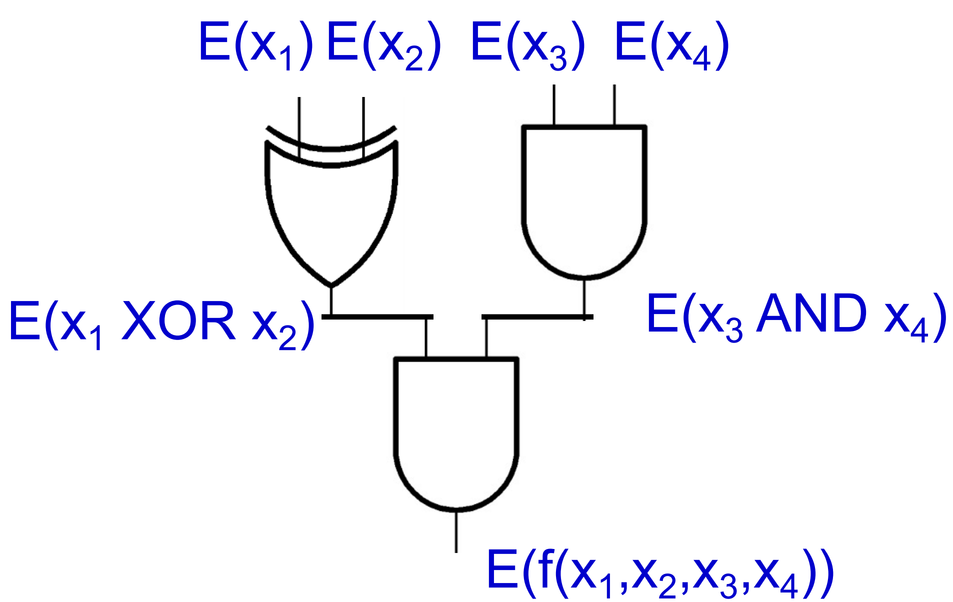
0 AND 0	0
1 AND 0	0
0 AND 1	0
1 AND 1	1

COMPUTING ON ENCRYPTED DATA

Because {XOR,AND} is Turing-complete …

… if you can compute sums and products on encrypted bits

… you can compute ***ANY*** function on encrypted inputs



What sort of objects can we add and multiply?

Polynomials?

$$(x^2 + 6x + 1) + (x^2 - 6x) = (2x^2 + 1)$$

$$(x^2 + 6x + 1) \text{ X } (x^2 - 6x) = (x^4 - 35x^2 - 6x)$$

Matrices?

$$\begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix} + \begin{pmatrix} -1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix} X \begin{pmatrix} -1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ -1 & 3 \end{pmatrix}$$

How about integers?!?

[Gentry, Halevi, van Dijk, **Vaikuntanathan'09**]

$$2 + 3 = 5$$

$$2 \text{ X } 3 = 6$$

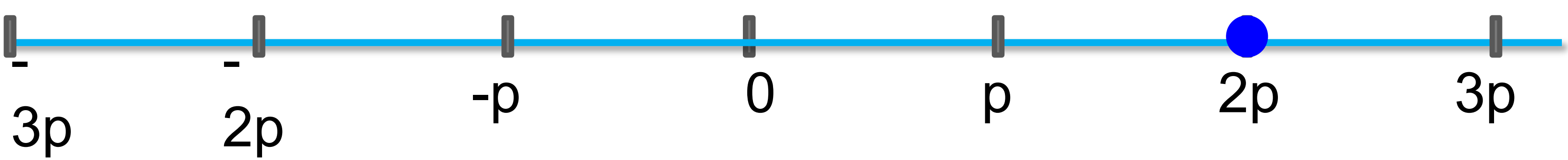
SYMMETRIC ENCRYPTION

ILLINOIS INSTITUTE OF TECHNOLOGY

Secret key: large *odd* number **p**

*To Encrypt a bit **b**:*

– pick a (random) “large” multiple of p, say **q·p**



SYMMETRIC ENCRYPTION

Secret key: large *odd* number **p**

*To Encrypt a bit **b**:*

– pick a (random) “large” multiple of p, say **q·p**

– pick a (random) “small” number **2·r+b**

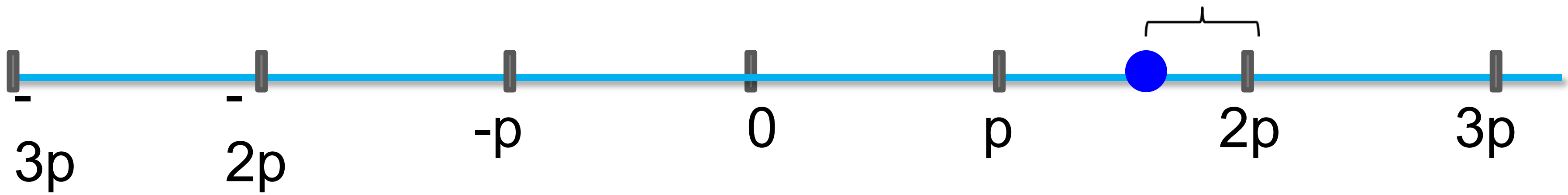
(this is even if b=0, and odd if b=1)

– Ciphertext **c = q·p+2·r+b**

*To Decrypt a ciphertext **c**:*

Take ***c** mod p* recovers the noise
Read off the least significant bit (lsb)

the “noise” = **2·r+b**



HOW SECURE IS THIS?

ILLINOIS INSTITUTE OF TECHNOLOGY

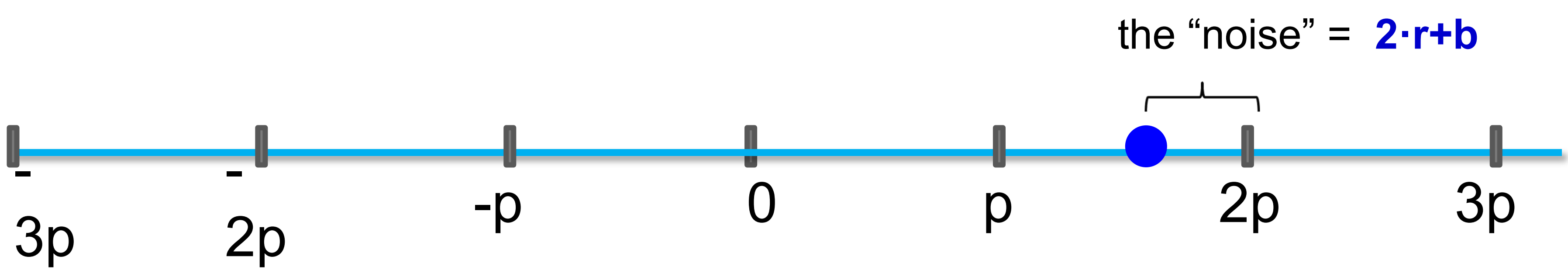
How secure is this?

… if there were no noise (think $r=0$)

… and I give you two encryptions of 0 ($q_1 p$ & $q_2 p$)

… then you can recover the secret key p

$$= \mathrm{GCD}(q_1 p,\ q_2 p)$$



HOW SECURE IS THIS?

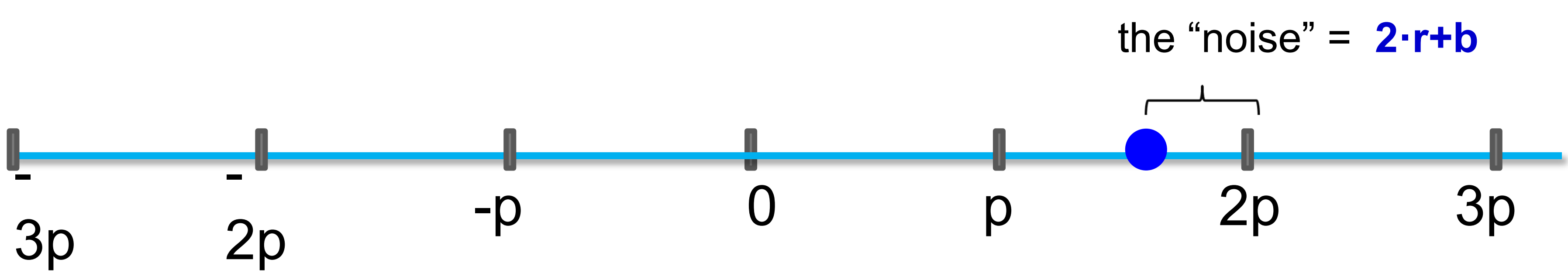
How secure is this?

… but if there is noise

… the GCD attack doesn't work

… and neither does any attack (we believe)

… this is called the *approximate GCD assumption*



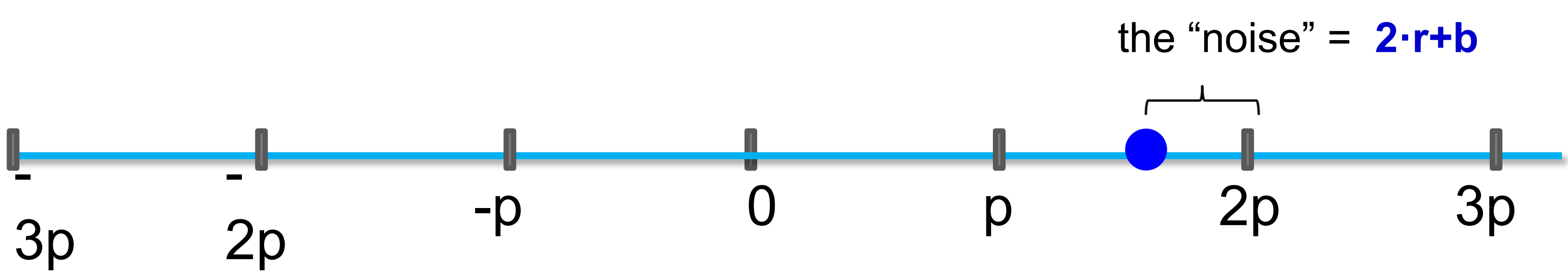
XORing two encrypted bits:

- $\mathbf{c_1} = q_1 \cdot p + (2 \cdot r_1 + b_1)$
- $\mathbf{c_2} = q_2 \cdot p + (2 \cdot r_2 + b_2)$
- $\mathbf{c_1 + c_2} = \mathbf{p} \cdot (q_1 + q_2) + \underbrace{\mathbf{2} \cdot (r_1 + r_2) + (b_1 + b_2)}$

Odd if $b_1=0, b_2=1$ (or) $b_1=1, b_2=0$

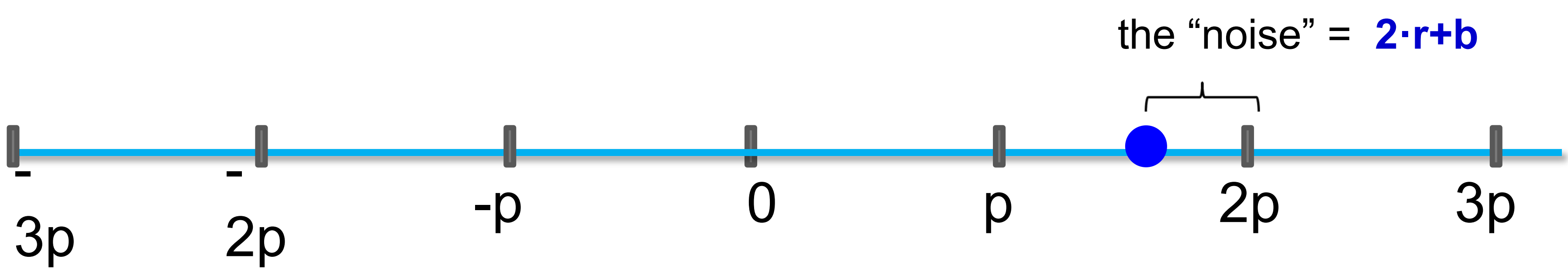
Even if $b_1=0, b_2=0$ (or) $b_1=1, b_2=1$

***lsb*= b_1 XOR b_2**

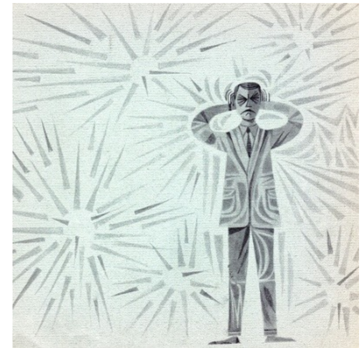


ANDing two encrypted bits:

- $\mathbf{c_1} = q_1 \cdot p + (2 \cdot r_1 + b_1)$
- $\mathbf{c_2} = q_2 \cdot p + (2 \cdot r_2 + b_2)$
- $\mathbf{c_1 c_2} = \mathbf{p} \cdot (c_2 \cdot q_1 + c_1 \cdot q_2 - q_1 \cdot q_2) + \underbrace{\mathbf{2} \cdot (r_1 r_2 + r_1 b_2 + r_2 b_1) + b_1 b_2}_{\textit{lsb} = b_1 \text{ AND } b_2}$



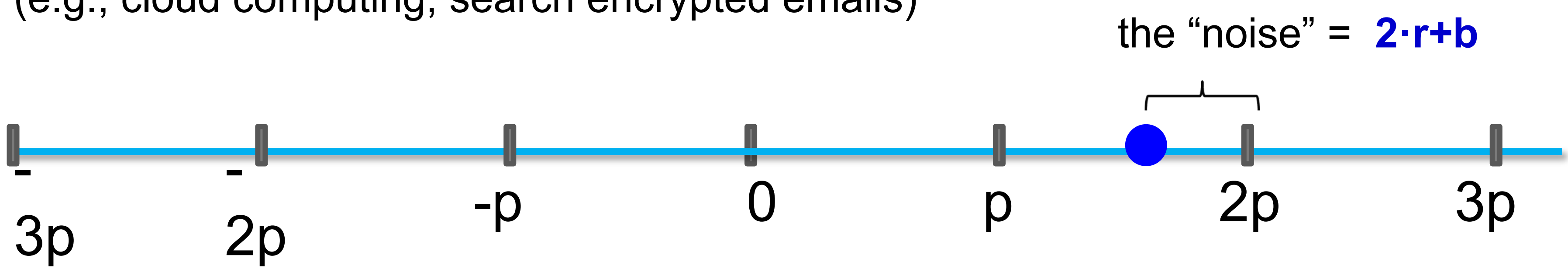
the noise grows!



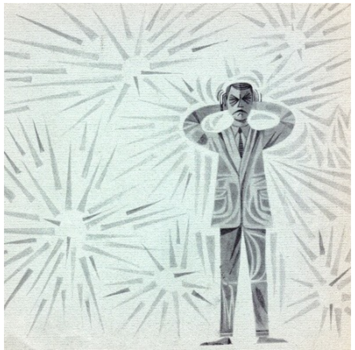
– $\mathbf{c_1+c_2} = \mathbf{p}\cdot(q_1 + q_2) + \underbrace{\mathbf{2}\cdot(r_1+r_2) + (b_1+b_2)}_{\textit{noise}= 2 * (\textit{initial noise})}$

– $\mathbf{c_1c_2} = \mathbf{p}\cdot(c_2\cdot q_1+c_1\cdot q_2-q_1\cdot q_2) + \underbrace{\mathbf{2}\cdot(r_1r_2+r_1b_2+r_2b_1) + b_1b_2}_{\textit{noise} = (\textit{initial noise})^2}$

Useless for many applications
(e.g., cloud computing, search encrypted emails)



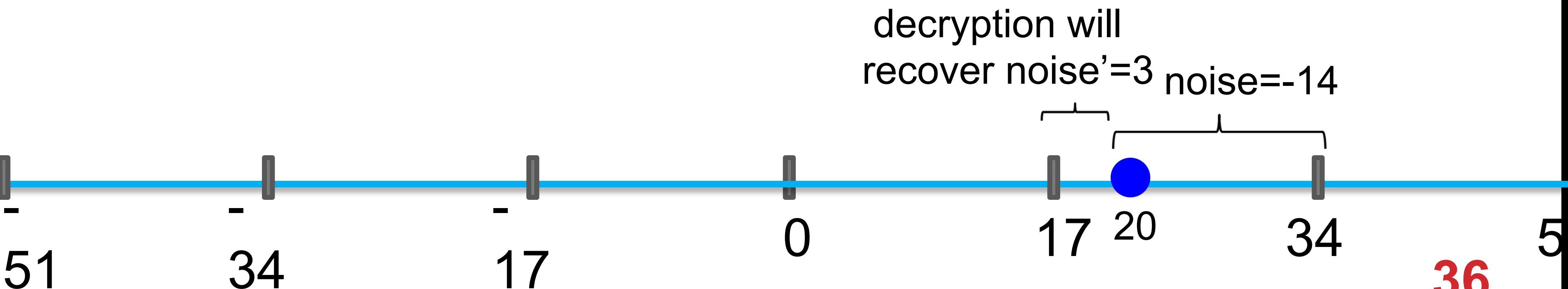
the noise grows!



… so what's the problem?

If the |noise| > p/2, then …

decryption will output an incorrect bit



ILLINOIS INSTITUTE OF TECHNOLOGY

So, what did we accomplish?

… we can do lots of additions and

… some multiplications

(= a "somewhat homomorphic" encryption)

… enough to do many useful tasks, e.g., database search, spam filtering etc.

much more …

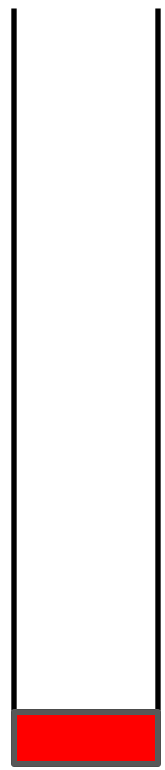
*The **“bootstrapping method”***

… If you can go a (large) part of the way,

then you can go all the way.

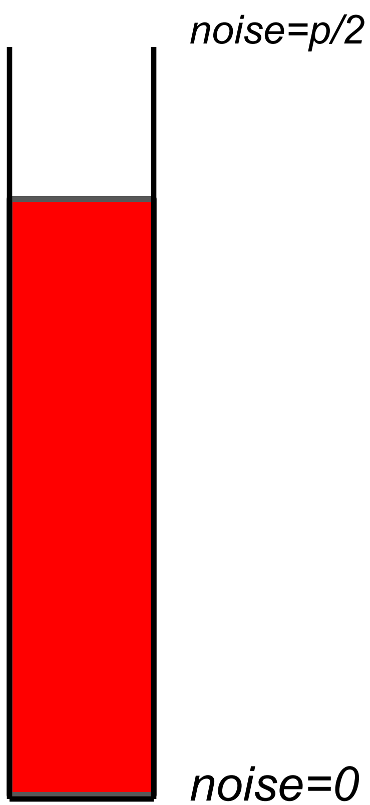
… but how?

The *“bootstrapping method”*



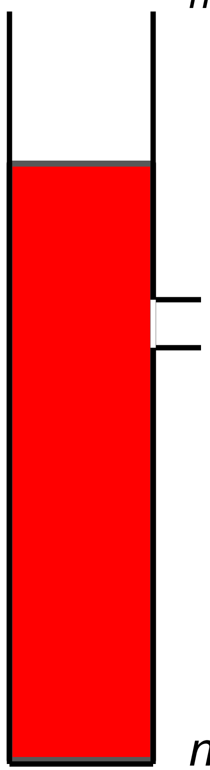
Initial noise

The “bootstrapping method”



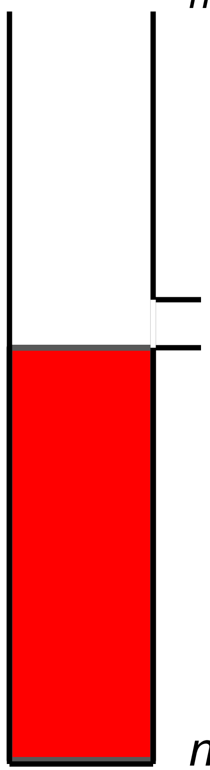
Noise after some sums and products

The *"bootstrapping method"*



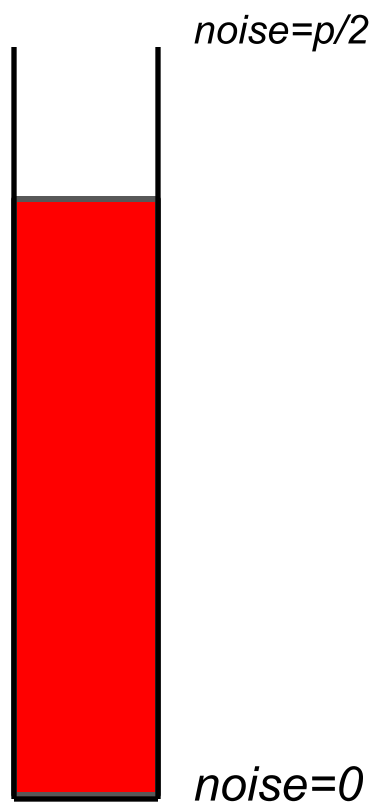
Bootstrapping =
"Valve" at a fixed height

The *"bootstrapping method"*



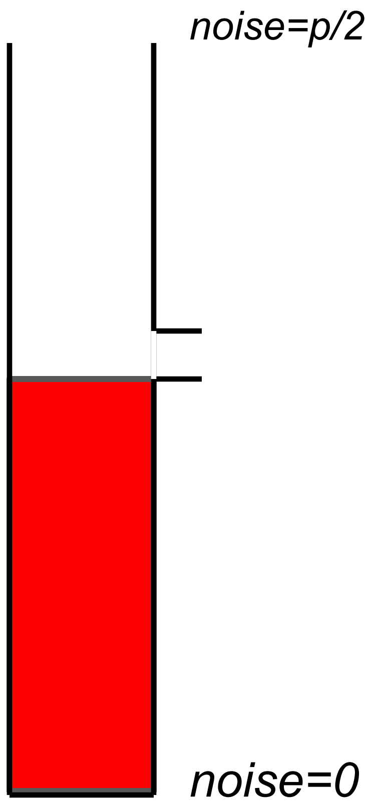
Bootstrapping =
"Valve" at a fixed height

The *“bootstrapping method”*



… repeat until done

The ***“bootstrapping method”***



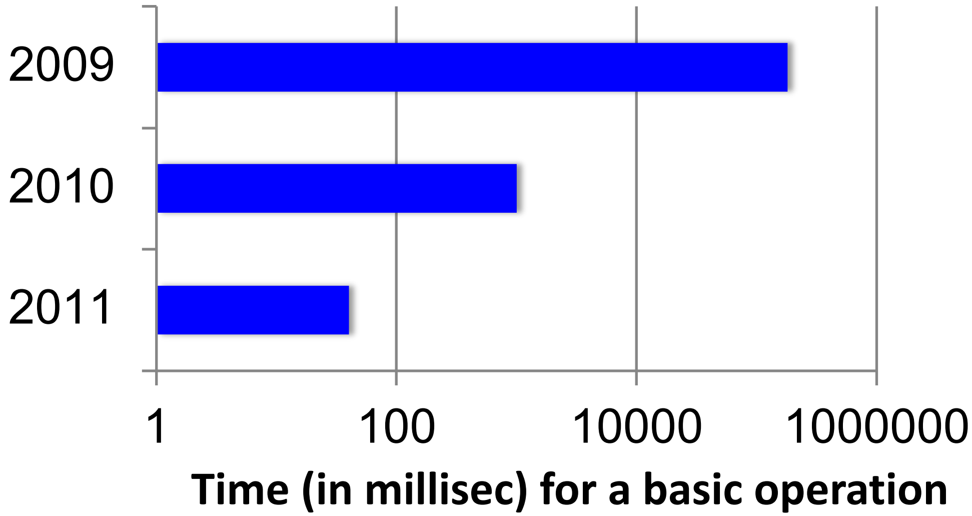
… repeat until done

- *Lots of new Encryption Schemes*

  … simpler, more secure, more efficient

  e.g., [Brakerski, Vaikuntanathan 2012]

- *Dramatic Efficiency Improvements*



References:

[1] "Computing arbitrary functions of Encrypted Data",
Craig Gentry, Communications of the ACM 53(3), 2010.

[2] "Computing Blindfolded: New Developments in Fully Homomorphic Encryption",
Vinod Vaikuntanathan, IEEE Foundations of Computer Science Invited Talk, 2012.

[3] "Fully Homomorphic Encryption from the Integers",
Marten van Dijk, Craig Gentry, Shai Halevi, Vinod Vaikuntanathan
http://eprint.iacr.org/2009/616, Eurocrypt 2010.

ACKNOWLEDGMENTS

Note: Some of the slides in this lecture are based on material created by

- Dr. Vinod Vaikuntanathan at University of Toronto
- Dr. Alexander Maximov at Ericsson