



Machine Learning on Encrypted Data

Prof. Jaewook Lee
Statistical Learning and Computational Finance Lab.
[*jaewook@snu.ac.kr*](mailto:jaewook@snu.ac.kr)
<http://slcf.snu.ac.kr>

This document is confidential and is intended solely for the use

* Many slides taken from Craig Gentry and Shai Halevi

HISTORY OF CRYPTOGRAPHY

A Brief History of Cryptography

In the beginning, if you had the key, **symmetric encryption**, you could **decrypt**.

Julius Ceasar (100-44 BC)



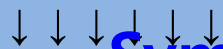
DWWDFN DW GDZQ



Message: **ATTACK**

Ciphertext: **DWWDFN DW GDZQ**

Key: +3



Key: -3



Symmetric Encryption:

Ciphertext: **DWWDFN**

Message: **ATTACK AT DAWN**

Encryption and Decryption use the same key

Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

A Brief History of Cryptography

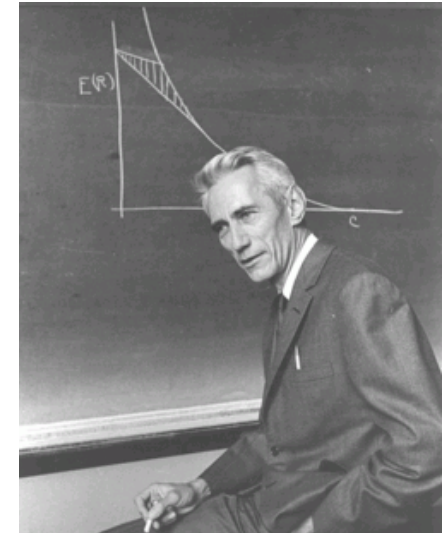
1900-1950



Vigenere



Enigma



Claude Shannon and
Information Theory

Symmetric Encryption:

Encryption and Decryption use the same key

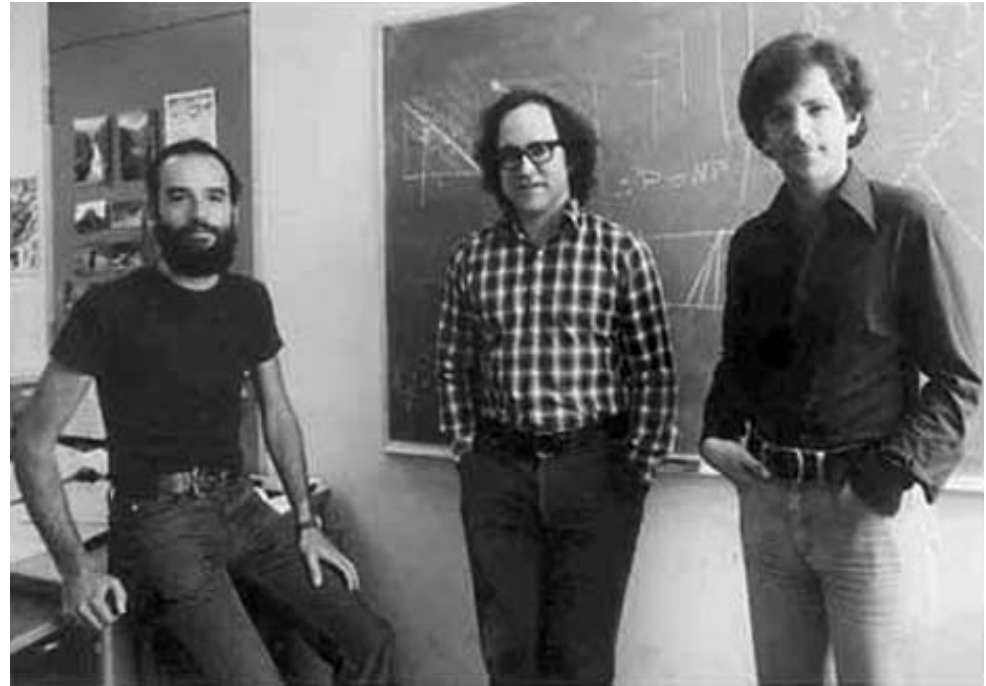
Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

A Brief History of Cryptography

(1970s)



Merkle, Hellman and Diffie
(1976)



Shamir, Rivest and Adleman
(1978)

Asymmetric Encryption

Encryption uses a public key, Decryption uses the secret key

Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

A Refresher on Public Key Encryption Schemes

■ Algorithms in an Encryption Scheme

- **KeyGen(...security_parameters) → keys**: The Key Generation algorithm takes some security parameter(s) as its input. It then outputs the key(s) of the scheme.
- **Encrypt(plaintext, key, randomness) → ciphertext**: The Encryption algorithm takes a plaintext, a key, and some randomness (encryption must be probabilistic to be "secure") as inputs. It then outputs the corresponding ciphertext (i.e. encrypted data).
- **Decrypt(ciphertext, key) → plaintext**: The Decryption algorithm takes a ciphertext and a key as inputs. It outputs the corresponding plaintext.

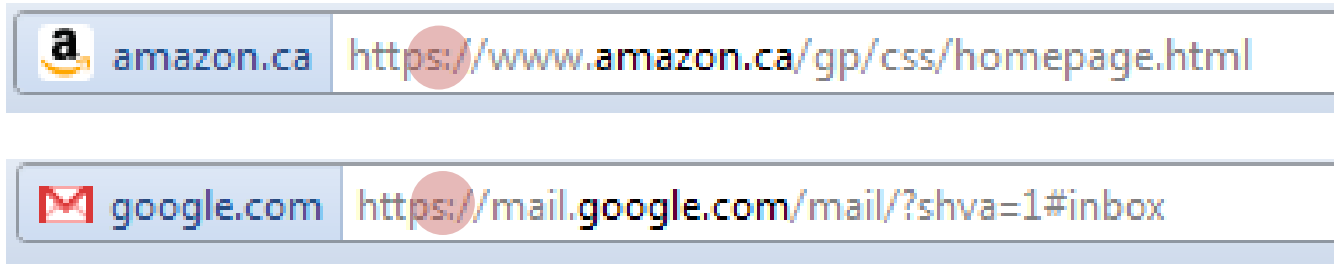
■ Symmetric (*private* key) encryption scheme

- the same key is used to both encrypt and decrypt

■ Asymmetric (*public* key) encryption scheme

- different keys are used to encrypt and decrypt. An individual may be able to encrypt data but have no way of decrypting the corresponding ciphertext

A Brief History of Cryptography



Asymmetric Encryption:

The Foundation of E-Commerce

RSA: The first and most popular asymmetric encryption

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

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

Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

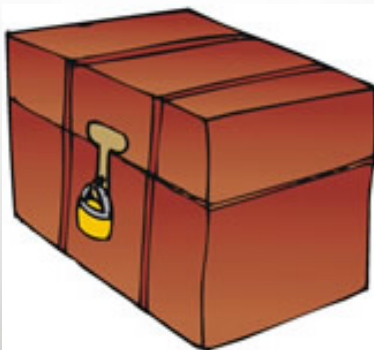
YET...

The world was black and white

The only thing anyone did with encrypted data
was ...

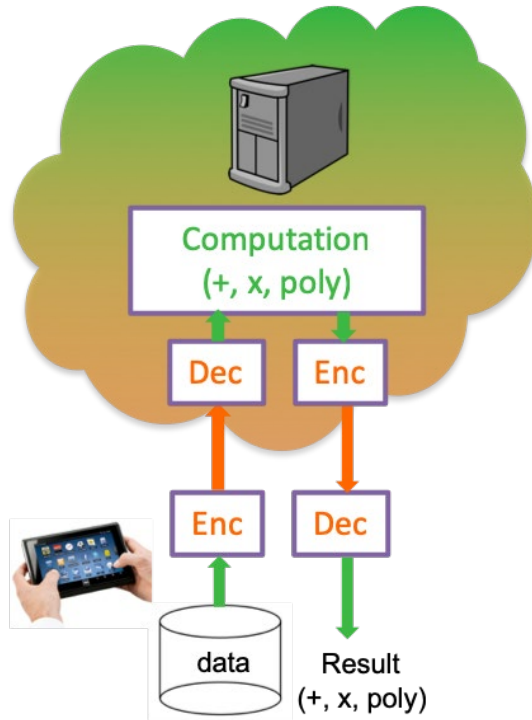
... decrypt it.

Encryption =



COMPUTING ON ENCRYPTED DATA

Classical Encryption : SSL (Internet), Credit Cards. . .

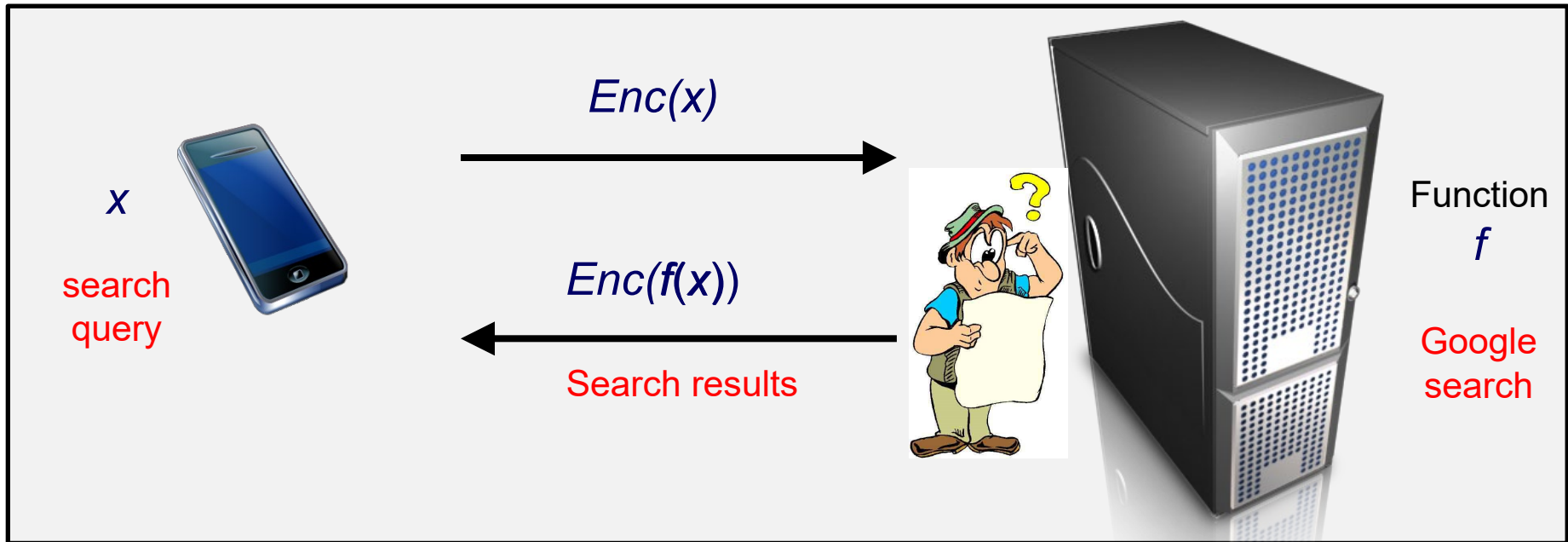


Data is encrypted for transmission and storage but processed in clear :-)

Source: C. Fontaine, "Fully Homomorphic Encryption Implementation Progresses and Challenges"

Computing on Encrypted Data

Can we delegate the processing of data, without giving away access to it.



WANT PRIVACY!

What else can we do with encrypted data, anyway?

Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

Encrypted Cloud Computing

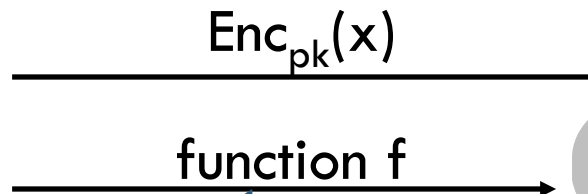
Source: Shai Halevi, "Homomorphic Encryption"

The special sauce! For security parameter λ , Eval's running should be $\text{Time}(f) \cdot \text{poly}(\lambda)$

"I want 1) the cloud to process my data
2) even though it is encrypted."

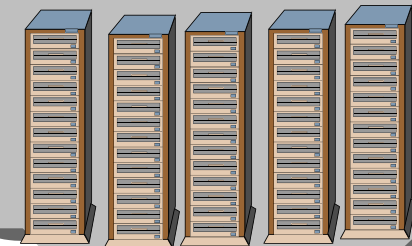


Alice
(Input: data x ,
secret key sk)
 $f(x)$



This could be
encrypted too.

Run
 $\text{Eval}[f, \text{Enc}_{pk}(x)]$
 $= \text{Enc}_{pk}[f(x)]$



Server
(Cloud)

Delegation: Should cost less for Alice to encrypt x and decrypt $f(x)$ than to compute $f(x)$ herself.

$\text{Enc}_{pk}[f(x)]$

An Analogy: Alice's Jewelry Store

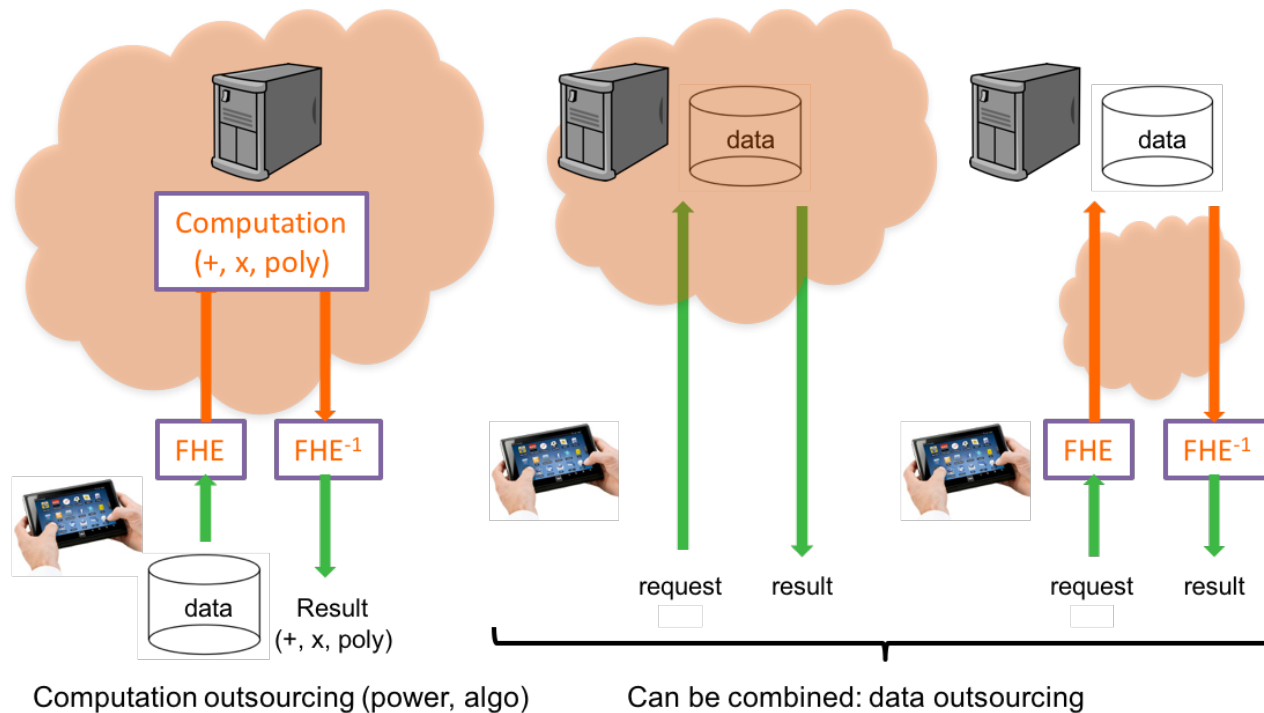
- Alice wants workers to assemble raw materials into jewelry
- But Alice is worried about theft:
 - She wants workers to process raw materials without having access.



- Alice puts raw materials in locked glovebox.
- Workers assemble jewelry inside glovebox, using the gloves.
- Alice unlocks box to get "results".

Source: Shai Halevi, "Homomorphic Encryption"

Homomorphic Encryption : we are dreaming of . . .



A revolution : data and/or services outsourcing without losing confidentiality !

Impact : citizens, administrations, companies, military, . . .

Domains : health care, power plants, multimedia content delivery, . . .

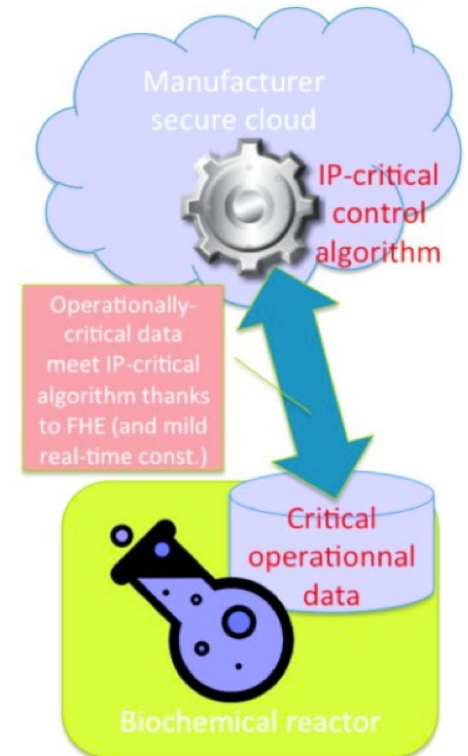
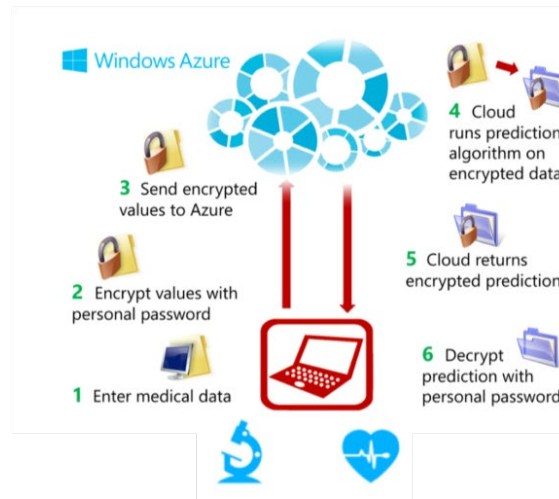
Computations : comparing, sorting/filtering, clustering, compressing, . . .

Source: C. Fontaine, "Fully Homomorphic Encryption Implementation Progresses and Challenges"

Also “Intelligent” and “Evolving” algorithms :-)

- + **privacy** concerns for the **end-user**
- + **IP** concerns and **software update** for the **service provider**

- ❖ Targeted **advertising**
- ❖ **Access Control** with respect to user profile
- ❖ **Biometric** authentication
- ❖ **Medical Diagnosis**
- ❖ **Critical engine** (reactor)
- ❖ **control**
- ❖ **Machine Learning**
- ❖ (deep learning)



Source: C. Fontaine, “Fully Homomorphic Encryption Implementation Progresses and Challenges”

Computing on Encrypted Data

Some people noted the algebraic structure in RSA...

Unpadded RSA

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

$$E(m_1) = m_1^e \quad E(m_2) = m_2^e$$

$$\begin{aligned} \text{Ergo ... } E(m_1) \times E(m_2) \\ &= m_1^e \times m_2^e \\ &= (m_1 \times m_2)^e \\ &= E(m_1 \times m_2) \end{aligned}$$

Multiplicative Homomorphism

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

**RSA is multiplicatively homomorphic
(but not additively homomorphic)**

Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

Computing on Encrypted Data

Other Encryptions were additively homomorphic
(but not multiplicatively homomorphic)

Paillier scheme

$$pk = (n, g), c = E_{pk}(m) = g^m r^n \bmod n^2$$

r in $\{0, \dots, n-1\}$ – some random value

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

Additive Homomorphism

$$E(m_1) + E(m_2) = E(m_1 + m_2)$$

What people really wanted was the ability to do arbitrary computing on encrypted data...

... and this required the ability to compute *both* sums *and* products ...
... on the same encrypted data set!

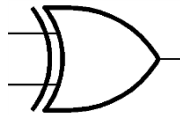
Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

Computing on Encrypted Data

Why SUMs and PRODUCTS?

SUM

=

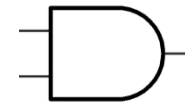


XOR

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

PRODUCT

=



AND

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

Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

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**

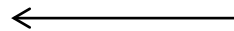
Example: Indexing a database

DB

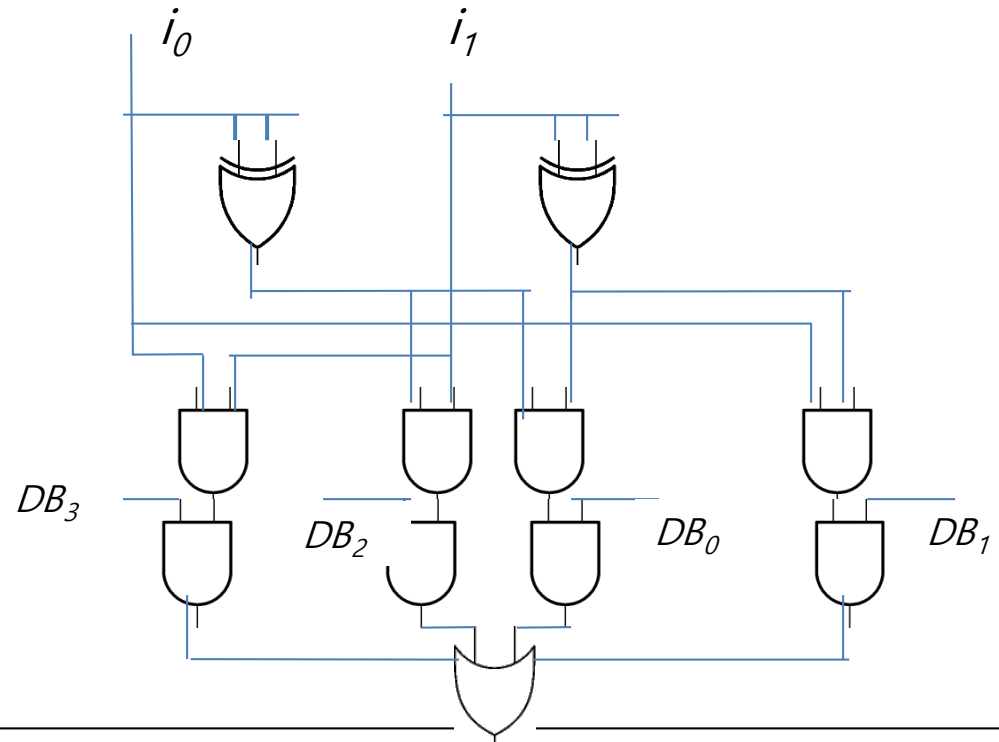
0
1
1
0

index

$$i = i_1 i_0$$



return DB_i



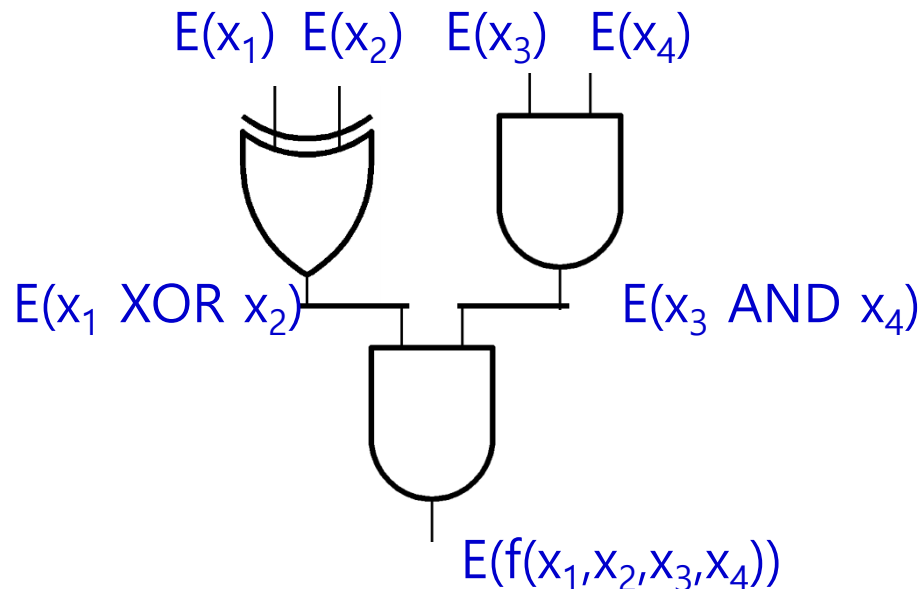
Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

Computing on Encrypted Data

Because **{XOR,AND}** is Turing-complete ...

... if you can compute sums and products on **encrypted bits**

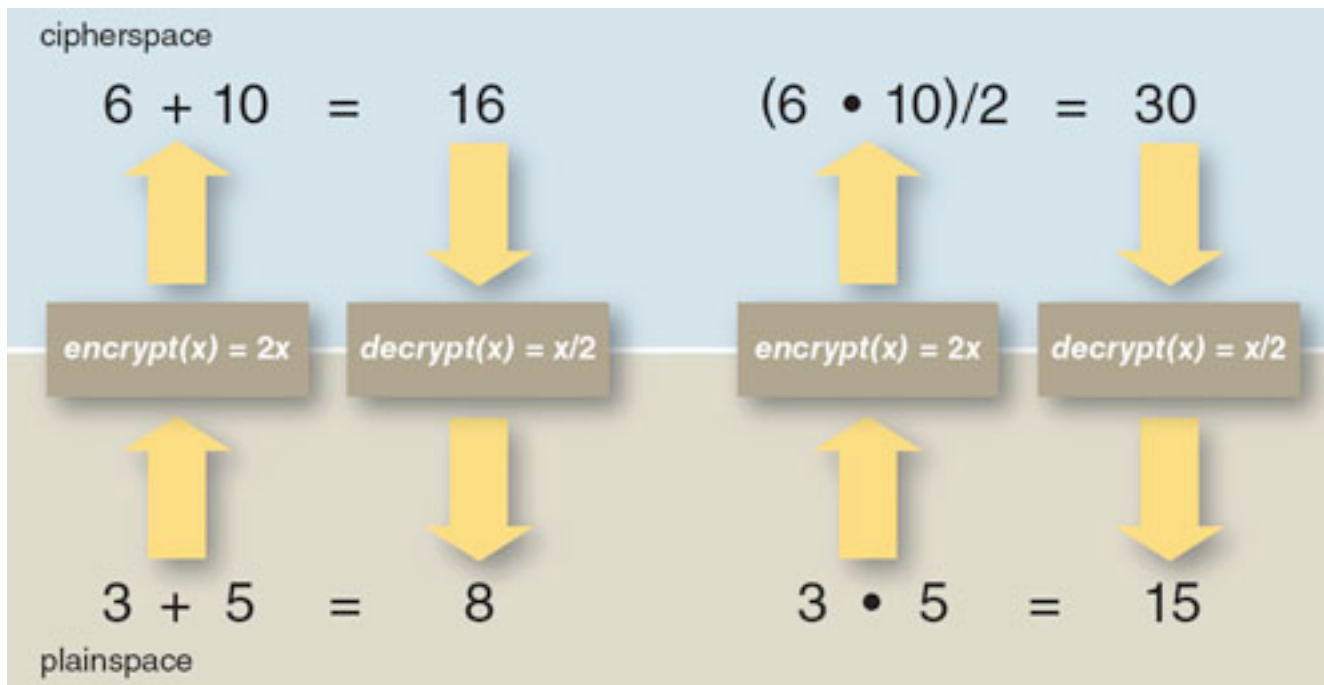
... you can compute **ANY** function on **encrypted inputs**



Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

A Toy HE Scheme (from American Scientist magazine)

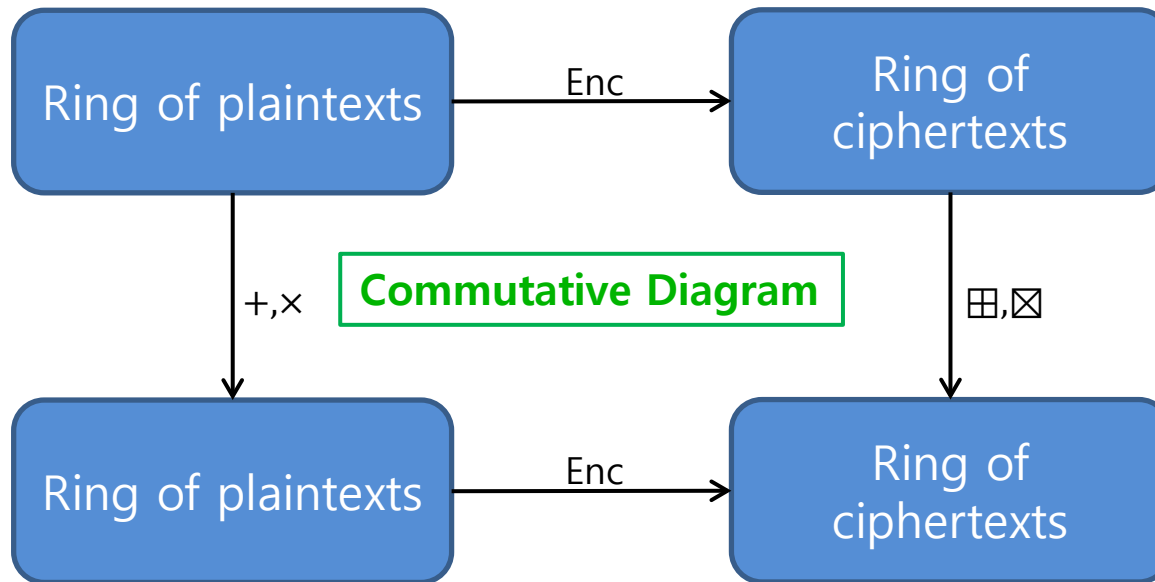
- Encryption: Double the plaintext. $x \rightarrow 2x$
- Decryption: Halve the ciphertext. $x \rightarrow x/2$



Source: Shai Halevi, "Homomorphic Encryption"

About “Homomorphism”

- Name inspired by ring-homomorphism



- Homomorphism should not be taken too literally
 - Else zero-encryptions form a linear subspace (ideal)
- Is it possible to hide such an ideal?
 - Some attempts (e.g. Polly Cracker), but broken

Source: C. Fontaine, “Fully Homomorphic Encryption Implementation Progresses and Challenges”

Notations

Common notations:

pk – public key

sk – secret key

m – message

c – ciphertext

$$c = \text{Enc}_{pk}(m)$$

$$m = \text{Dec}_{sk}(c)$$

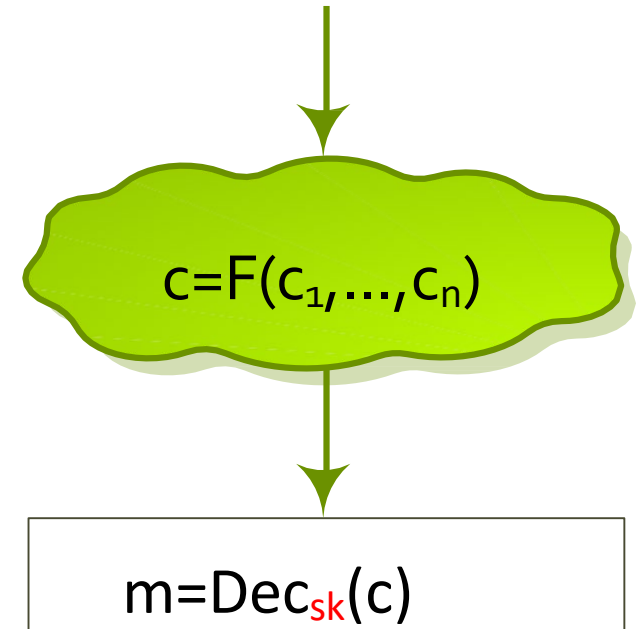
$E_m(pk)$ - encryption algorithm as a circuit

$D_m(sk)$ - 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

$$c_1 = \text{Enc}_{pk}(m_1) \\ \dots c_n = \text{Enc}_{pk}(m_n)$$



Source: A.r Maximov, "Homomorphic Encryption"

Homomorphic Encryption

■ Homomorphic Encryption (HE)

- **Procedures:** KeyGen, Encrypt, Decrypt, **Eval**
- **Semantic Security:** same as for basic encryption
- **Correctness:** For any function f in "supported" family F :

$$c_1 \leftarrow \text{Enc}_{pk}(m_1) \quad \dots \quad c_t \leftarrow \text{Enc}_{pk}(m_t)$$

$$c^* \leftarrow \text{Eval}_{pk}(f, c_1, \dots, c_t) \quad \text{Dec}_{sk}(c^*) = f(m_1, \dots, m_t)$$

- **Compactness:** complexity of decrypting c^* does not depend on complexity of f

A way to delegate processing of your data,
without giving away access to it.

HE Security: A Paradox?

■ A Paradox and its resolution

- Cloud stores my encrypted files: $pk, \text{Enc}_{pk}(f_1), \dots, \text{Enc}_{pk}(f_n)$.
- Later, I want f_3 , but want to hide "3" from cloud.
- I send $\text{Enc}_{pk}(3)$ to the cloud.
- Cloud runs $\text{Eval}_{pk}(F, \text{Enc}_{pk}(3), \text{Enc}_{pk}(f_1), \dots, \text{Enc}_{pk}(f_n))$,
where $F(n, \{\text{files}\})$ is the function that outputs the n th file.
- It sends me the (encrypted) file f_3 .
- Paradox?: Can't the cloud "see" it is sending the 3rd encrypted file? By comparing the stored value $\text{Enc}_{pk}(f_3)$ to the ciphertext it sends?

Resolution of paradox:

Semantic security implies:

- Many encryptions of f_3 ,
- Hard to tell when two ciphertexts encrypt the same thing.

Source: Shai Halevi, "Homomorphic Encryption"

Does Fully Homomorphic Encryption Ever Exist?

Some Properties of Fully Homomorphic Encryption (FHE).

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

Source: Shai Halevi, "Homomorphic Encryption"

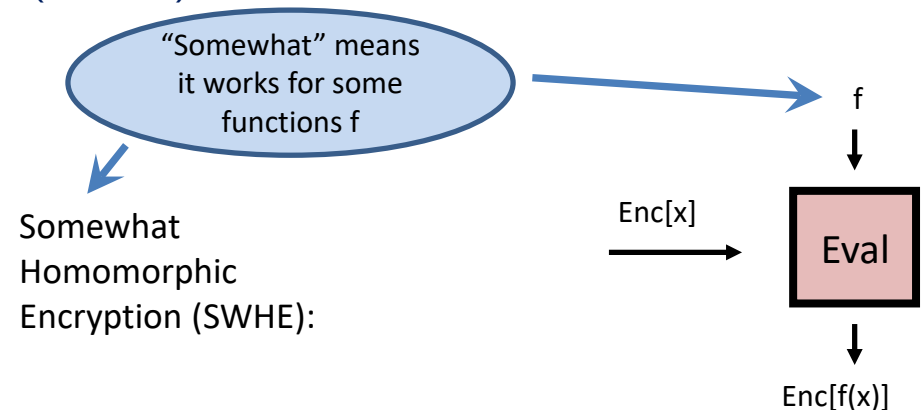
Homomorphic Encryption Basics

■ Partially HE (PHE)

- HE scheme where only one type of operations is possible (multiplication or addition)

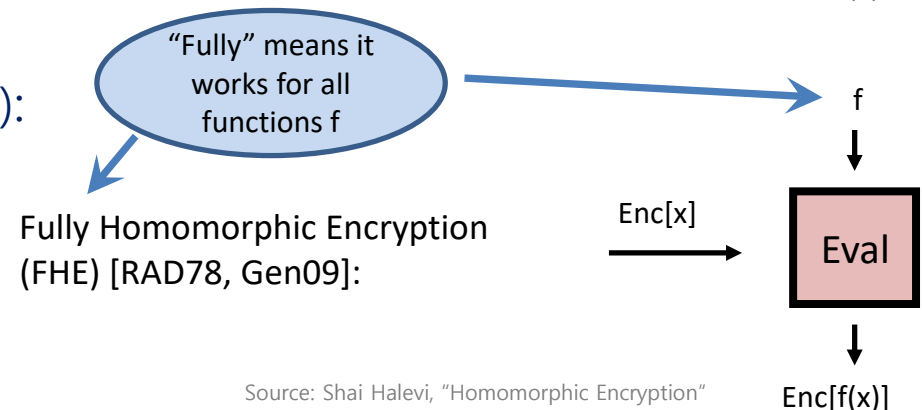
■ Somewhat Homomorphic Encryption (SWHE):

- Limited processing
- Could be cheaper computationally.
- Pre-2009 schemes were *somewhat homomorphic*.



■ Fully Homomorphic Encryption (FHE):

- Arbitrary processing
- But computationally expensive.

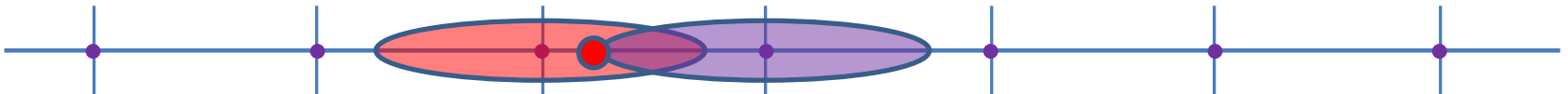


Source: Shai Halevi, "Homomorphic Encryption"

Constructing SWHE

- How to compute on encrypted data?
 - Express functions to compute as a binary circuit with $+$, \times
 - That's always possible
 - Design a bit-encryption scheme that supports addition, multiplication of encrypted bits
 - That's the hard part

- Noisy Ciphertexts



- Each ciphertext has some noise that hides the message.
 - Think: "hidden" error correcting codes...
- If error is small, Alice can use knowledge of "hidden" code to remove the noise.
- If noise is large, decryption is hopeless even for Alice.

Source: Shai Halevi, "Homomorphic Encryption"

COMPUTING ON ENCRYPTED DATA

Fully-Homomorphic Encryption!



Cryptography's Holy Grail

Computing on Encrypted Data

Fully-Homomorphic Encryption!

Amazing Applications:

People tried do this ...

... for years ...

... and years ...

... with no success.



Private Cloud Computing





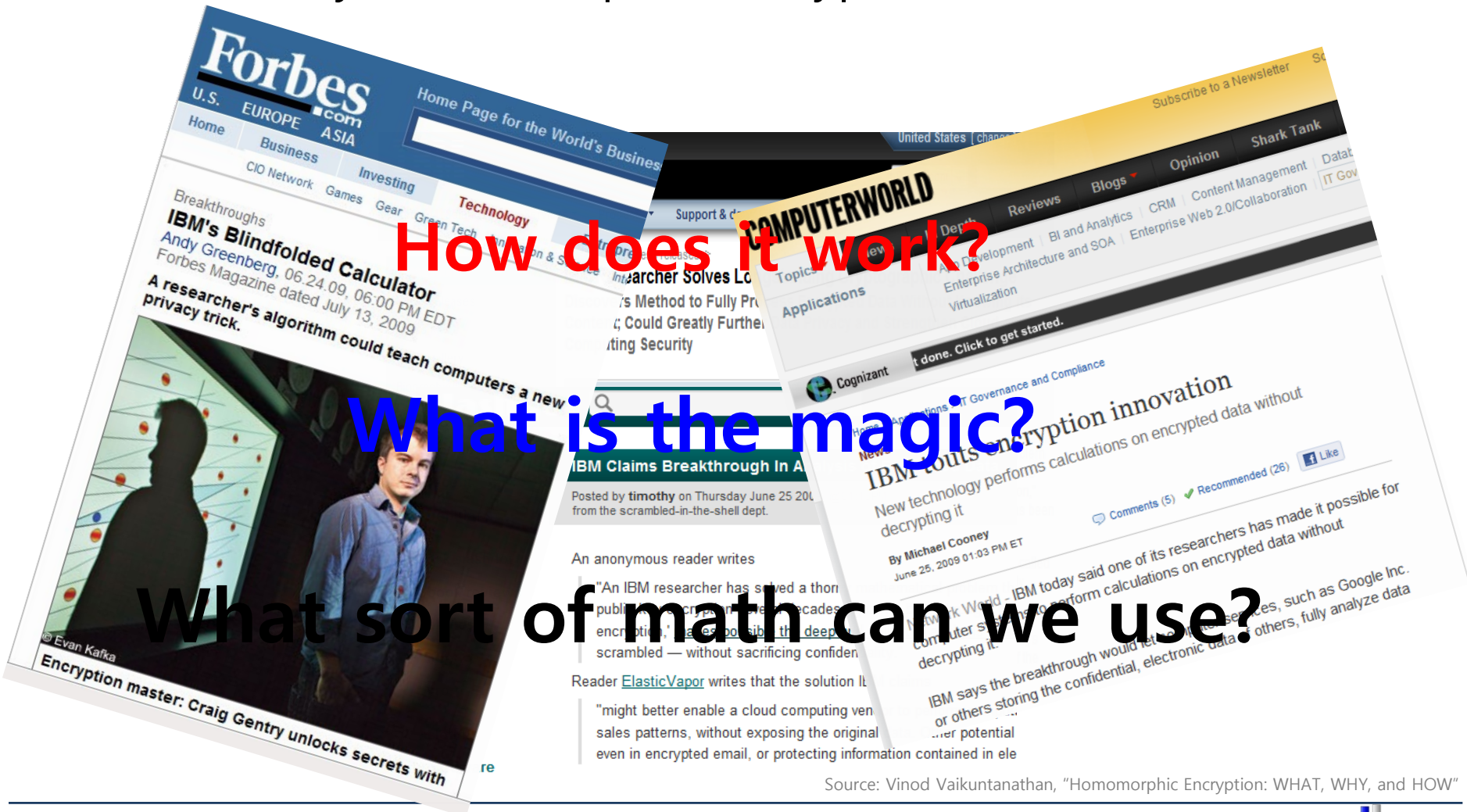

oogle.com <https://mail.google.com/mail/?shva=1#inbox>

Delegate *arbitrary processing* of data
without giving away *access* to it

Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

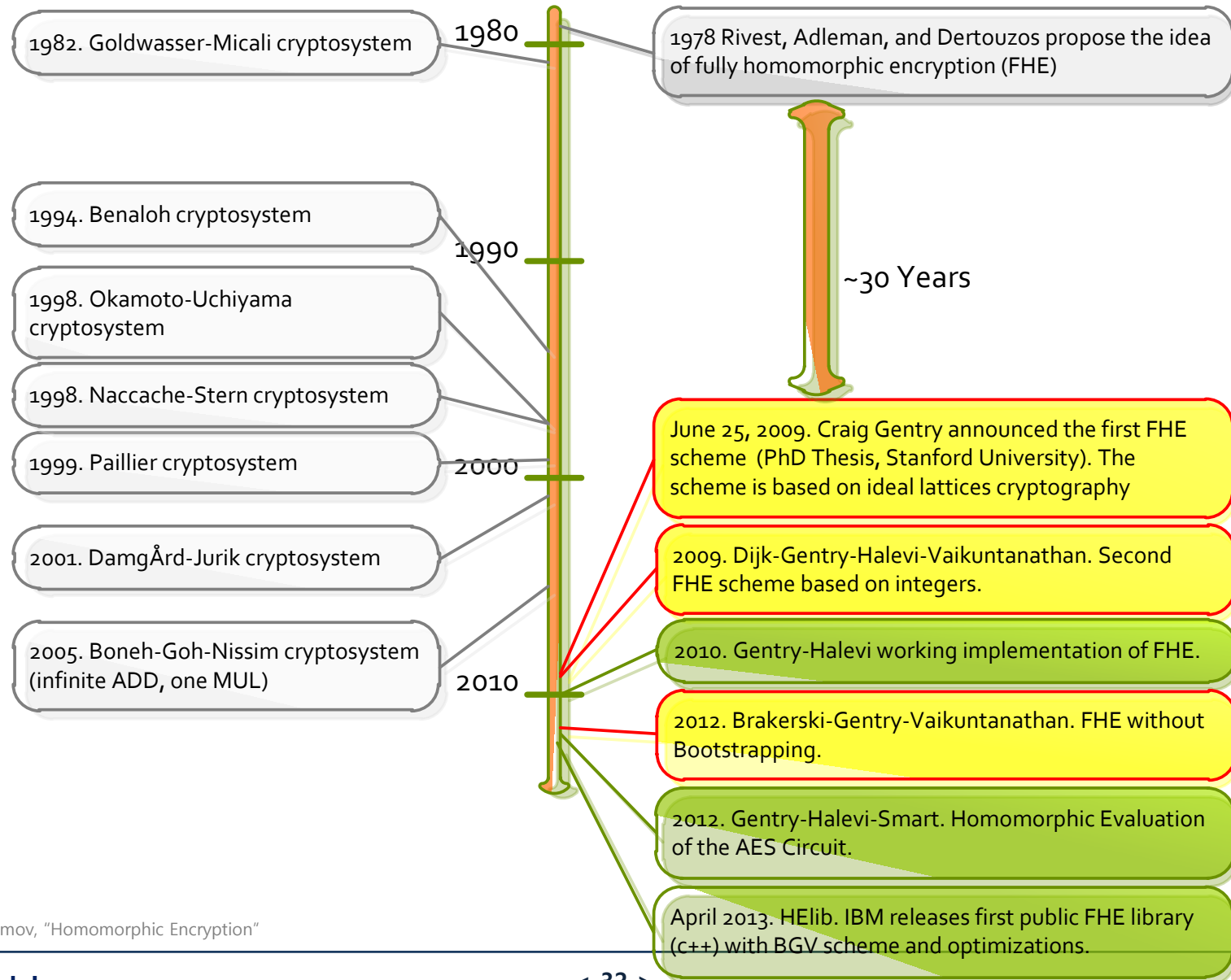
... until, in October 2008 ...

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



Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

“Holy Grail” for 30 years



Source: A.r Maximov, “Homomorphic Encryption”

Why do we care about FHE?

■ Private Transactions on a Public Ledger (e.g. blockchain)

- A private transaction scheme allows a user to securely send digital currency to some recipient without revealing the amount being sent to anyone (apart from the intended recipient) on a public distributed ledger. This is in contrast to cryptocurrencies like Bitcoin where transactions and balances are all public information.
- We assume a fully homomorphic (public key) encryption scheme. All encryptions are done with respect to the Receiver's keys.
 - Sender wants to send t tokens to the Receiver. Sender encrypts t under the Receiver's public key.
 - Sender publishes $\text{Encrypt}(t)$ on the public ledger.
 - Notice that $\text{Encrypt}(t) + \text{Encrypt}(\text{Receiver's balance}) = \text{Encrypt}(\text{Receiver's balance after performing the transaction})$.
- This can be extended to more interesting financial applications such as auctions, voting, and derivatives. An FHE scheme that models computation as *modular arithmetic* may be the most suitable for this type of application.
 - Note: For this to work in practice, some other cryptographic tools are also needed (such as [zero-knowledge proofs](#)).

Why do we care about FHE?

■ Private Machine Learning

- "learn" from data (via machine learning) while still keeping the data itself private.
- One well-known work in this space is [Crypto-Nets](#) which combines (as the name might suggest) FHE and [neural networks](#) to allow for outsourced machine learning.
 - When people talk about "outsourced machine learning," the setting usually being described is one in which a data *owner* wants to learn about his data but does not feel comfortable openly sharing his data with a company providing machine learning models. Thus, the data owner encrypts his data using FHE and allows the machine learning company to run their models on the encrypted data set.
- Homomorphic encryption is not the sole solution to this problem; [differential privacy](#) is a closely related subject that has garnered a lot of attention in the past few years. Differential privacy is more statistics than cryptography as it does not directly involve encryption/decryption. It works by adding noise to data in the clear while still guaranteeing that meaningful insights can be drawn from the noisy data sets. Apple is one major company that [purportedly uses](#) differential privacy to learn about your behavior while maintaining some level of privacy.

Why FHE Hasn't Taken Off Yet: Challenges in the Space

- Many Different Schemes (aka everyone is speaking a different language)
 - FHE schemes model computation in one of three ways—as TFHE (boolean), BFV (modular arithmetic), and HEAAN (floating point arithmetic).
 - you don't know the tradeoffs between the different FHE schemes or how the schemes relate to one another
 - you have to invest a lot of energy into understanding a single scheme
 - a lot of the knowledge gained from understanding one scheme is not really transferable to learning a new scheme (under a different computational model)
 - Potential Solutions: (Not yet!!)
 - Updated and improved benchmarking efforts
 - Better resources/guides to explain tradeoffs between schemes
 - Ability to move between different FHE schemes (i.e. interoperability)

Why FHE Hasn't Taken Off Yet: Challenges in the Space

- Efficiency (more complex of an issue than it seems)
 - How many computations do you want to perform?
 - Many schemes, however, only allow for a certain number of homomorphic operations to be performed. After that point, there's no guarantee decryption will be successful.
 - The schemes that allow for a truly *arbitrary* number of homomorphic computations suffer from very poor performance.
 - How big do you need your plaintext space to be?
 - The larger the plaintext space, the slower it will be to perform operations. Looking again at [TFHE](#), the plaintext-to-ciphertext expansion is 10,000:1 for an acceptable level of security (100 bits)
 - Are you going to perform the same operation on many different plaintexts or ciphertexts (i.e. "SIMD" style computations)?
 - SIMD style operations, if used correctly, can really improve efficiency
 - What key sizes are acceptable to you?
 - With [TFHE](#), some keys you'll need to access in the scheme can be 1 GB large!
 - Hardware acceleration
 - accelerate FHE schemes using [GPUs](#) ([nuFHE](#) and [cuFHE](#)) or using [FPGAs](#) (e.g. [HEAX](#))

<https://blog.nucypher.com/an-engineers-guide-to-fully-homomorphic-encryption/>

Show me the code!

- Usability (there is none).
 - FHE is *not* beginner-friendly, *not* user-friendly, and *certainly not* non-cryptographer-friendly.
 - Standardization efforts
 - Standardization efforts are currently led by the open industry/government/academic consortium "[Homomorphic Encryption Standardization](#)." They've already released [draft proposals](#) around security and API standards for homomorphic encryption.
 - More user-friendly libraries and examples
 - [Microsoft's SEAL](#) has an incredibly instructive [examples section](#)
 - Potential compilers: [Cingulata](#) is one such attempt.
 - In the news:
 - February 21, 2019: [Microsoft SEAL open source homomorphic encryption library gets even better for .NET developers!](#)
 - June 4, 2020: [IBM releases FHE toolkit for MacOS and iOS; Linux and Android Coming Soon](#)

Software libraries

■ Software libraries

- Some libraries (such as SEAL & PALISADE) offer multiple types of FHE schemes

COMPUTATIONAL MODEL	AVAILABLE LIBRARIES
Boolean	TFHE , nuFHE , PALISADE (tFHE)
Modular Arithmetic	HElib , SEAL (BFV), Lattigo (BFV), PALISADE (BFV, BGV)
Floating Point Arithmetic	SEAL (CKKS), Lattigo (CKKS), PALISADE (CKKS)

- IBM [HElib](#) (Halevi & Shoup)
- Microsoft [SEAL](#)
- NJIT/Duality [PALISADE](#) (Rohloff, Cousins & Polyakov)
- Functional Lattice Cryptography [LoL](#) (Crockett & Peikert)
- Fastest FHE of the West [FHEW](#) (Ducas & Micciancio)
- FHE over the Torus [TFHE](#) (Chillotti, Gama, Georgieva & Izabachene)
- Approximate FHE [HEAAN](#) (Cheon, Kim, Kim & Song)

HOMOMORPHIC ENCRYPTION

What sort of objects can we add and multiply?

Polynomials?

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

$$(x^2 + 6x + 1) \times (x^2 - 6x) = (x^4 - 35x^2 - 6x)$$

Matrices?

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

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

How about integers?!?



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

$$2 + 3 = 5$$

$$2 \times 3 = 6$$



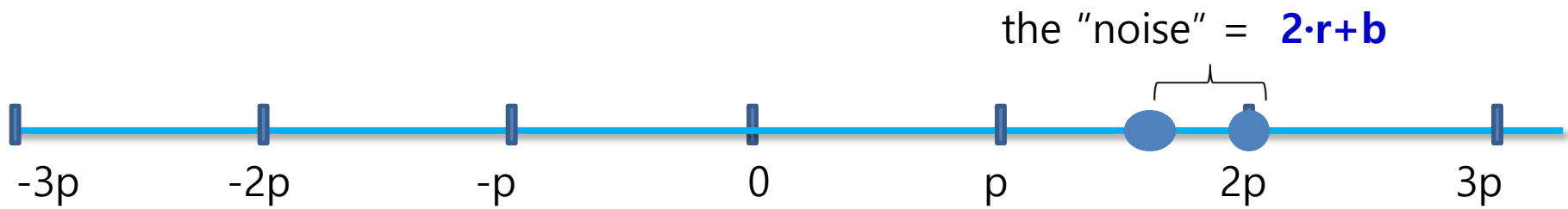
Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

SHE over the Integers – SYMMETRIC KEY SCHEME

- Second FHE scheme based on integers. [2009. Dijk-Gentry-Halevi-Vaikuntanathan.]
- **KeyGen**: key is an odd integer $p \in [2^{\eta-1}, 2^{\eta})$.
- **Encrypt**(p, m): to encrypt one bit $m \in \{0,1\}$

$$c = pq + 2r + m$$
 - q, r – are chosen random, such that $|2r| < p/2$.
- **Decrypt**(p, c):

$$m = (c \bmod p) \bmod 2$$
- Proposed constraint: $p \sim 2^{\eta}, q \sim 2^{\eta^3}, r \sim 2^{\text{sqrt}(\eta)}$.



Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

Source: A.r Maximov, "Homomorphic Encryption"

How secure is this?

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

... and I give you two encryptions of 0 (q_1p & q_2p)

... then you can recover the secret key p

$$= \text{GCD}(q_1p, q_2p)$$

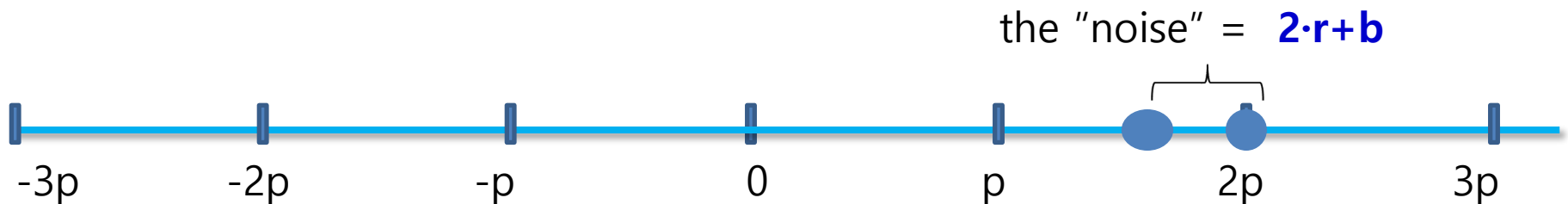
최대공약수 (Greatest Common Divisor)

... 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*



Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

Security of SWHE with Integers

- Reduction:
 - If “approximate gcd” problem is hard, then the scheme is **semantically secure**.
- Approximate GCD Problem:
 - Given many $x_i = q_i p + r_i$ (approx multiples of p), find p .
- Compactness Property
 - This means that the bit-length of the resulting ciphertext does not depend on the evaluation function F .
 - For this reason we do:
 - add more “encryptions of zero” to the public key y_j such that they use larger q 's. y_0 must be of bit-length from $|x_0| \dots 2|x_0|$.
 - after each operation (ADD or MUL) we select y_j such that a subtraction from the result of the operation will lead again to a ciphertext of size $|x_0|$.
 - *Multiplication doubles the number of bits of the ciphertext, but bringing it back to the initial size would cost just an addition of a few trivial noises from y_j .*
 - **Consequence:**
 - Evaluation of $c_1 * c_2 + c_1 * c_3$ and $c_1 * (c_2 + c_3)$ will result in different ciphertexts, although the same result in plaintext.

XORing two encrypted bits:

$$- c_1 = q_1 \cdot p + (2 \cdot r_1 + b_1)$$

$$- c_2 = q_2 \cdot p + (2 \cdot r_2 + b_2)$$

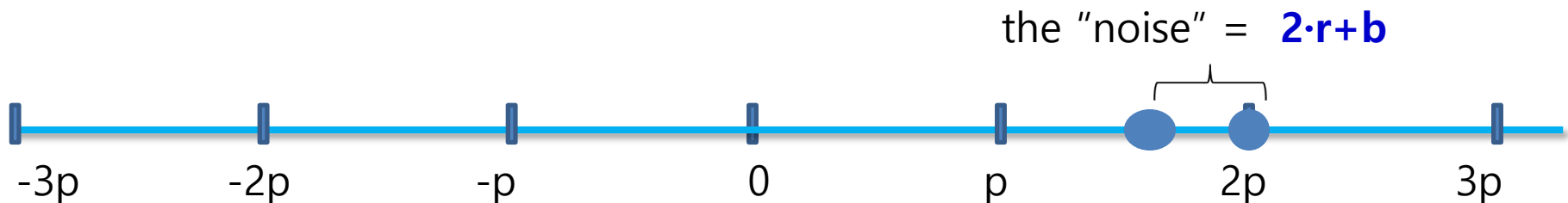
$$- c_1 + c_2 = p \cdot (q_1 + q_2) + 2 \cdot (r_1 + r_2) + (b_1 + b_2)$$

$$lsb = b_1 \text{ XOR } b_2$$

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 = least significant bit



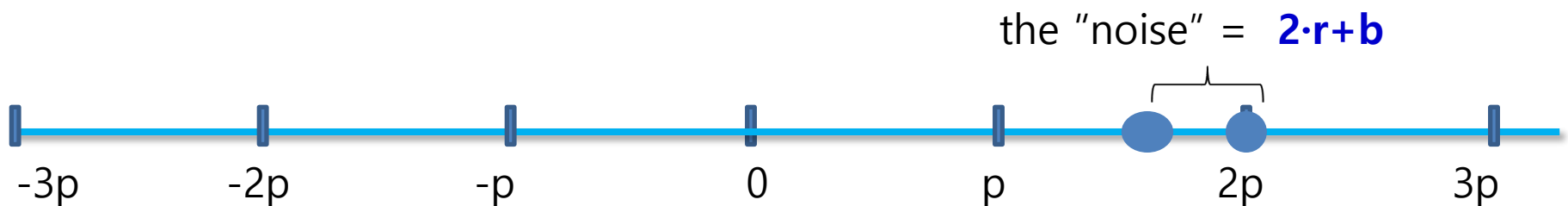
Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

ANDing two encrypted bits:

$$- c_1 = q_1 \cdot p + (2 \cdot r_1 + b_1)$$

$$- c_2 = q_2 \cdot p + (2 \cdot r_2 + b_2)$$

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



Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

HE over the Integers – SYMMETRIC KEY SCHEME

■ ADDITION:

$$\begin{aligned}
 \text{Decrypt}(c_1 + c_2, p) &= ((pq_1 + 2r_1 + m_1) + (pq_2 + 2r_2 + m_2) \bmod p) \bmod 2 \\
 &= (p(q_1 + q_2) + 2(r_1 + r_2) + (m_1 + m_2) \bmod p) \bmod 2 \\
 &= 2(r_1 + r_2) + (m_1 + m_2) \bmod 2 \\
 &= m_1 \oplus m_2
 \end{aligned}$$

■ MULTIPLICATION:

$$\begin{aligned}
 \text{Decrypt}(c_1 * c_2, p) &= ((pq_1 + 2r_1 + m_1) * (pq_2 + 2r_2 + m_2) \bmod p) \bmod 2 \\
 &= (p(pq_1q_2 + 2q_1r_2 + q_1m_2 + 2q_2r_1 + q_2m_1) + 2(2r_1r_2 \\
 &\quad + r_1m_2 + m_2r_1) + m_1m_2) \bmod p) \bmod 2 \\
 &= 2(2r_1r_2 + r_1m_2 + m_2r_1) + m_1m_2 \bmod 2 \\
 &= m_1 \otimes m_2
 \end{aligned}$$

- The scheme is both additively and multiplicatively homomorphic for shallow arithmetic circuits.
- The number of ADD and MUL is limited since the noise grows.
- The noise r must be sufficiently smaller than p to allow more ADDs and MULs.

Source: A.r Maximov, "Homomorphic Encryption"

Algorithm: SWHE over the Integers

Main Idea

Encryptions of 0 are something small and even modulo a secret integer.

- **Secret key:** large odd integer $sk = p$
- **Public key:** integers $pk = \{x_i = q_i p + 2r_i\}_i$, $|r_i| \ll p$
 - These are encryptions of 0
- **Encrypt($pk, m \in \{0,1\}$):** S is a random subset of pk (x_i 's), then add $m \in \{0,1\}$
 - $c = \sum_{i \in S} x_i = (\sum_S q_i) \cdot p + 2 \sum_S r_i + m$
- **Decrypt($sk = p, c$):** $(c \bmod p) = 2r + m$, LBS is message m
- **ADD, MULT:** sum/product over the integers

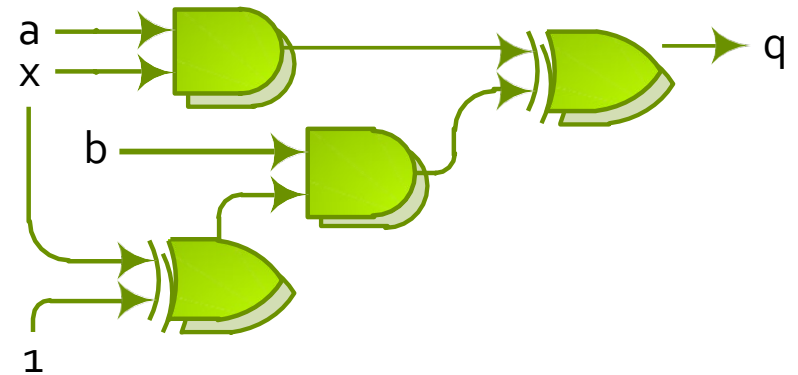
Encryption can now be viewed as:
adding m to a random subset sum of "encryptions of zero"

Source: Shai Halevi, "Homomorphic Encryption"

HE over the Integers – SYMMETRIC KEY SCHEME

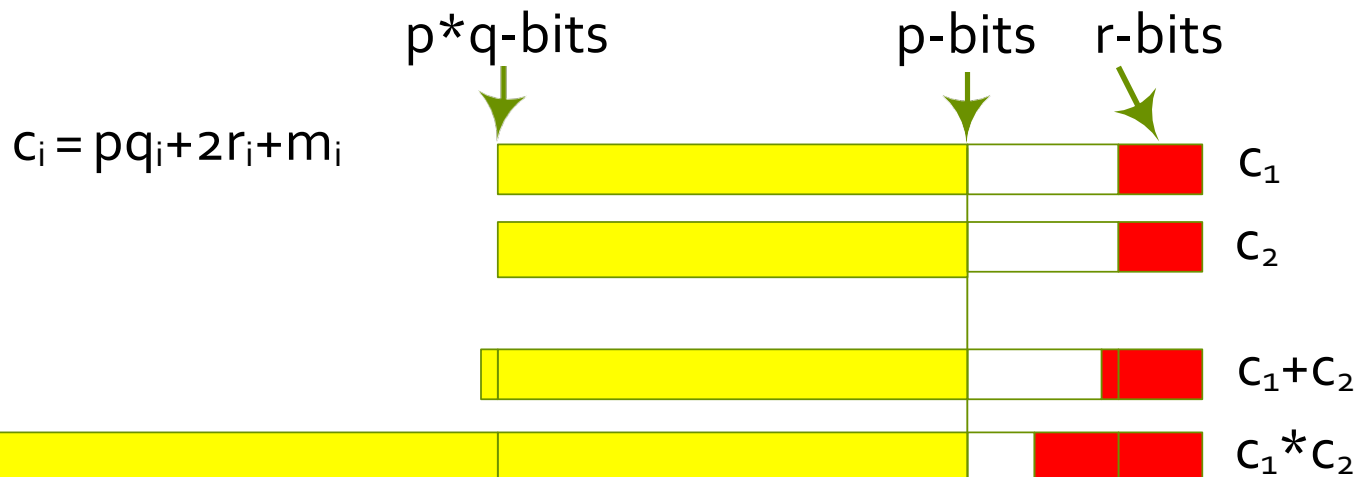
Any computer program can be represented in terms of AND-XOR gates

$f(x, a, b)$: if($x==1$) $q=a$; else $q=b$



$F(c_a, c_x, c_b, c_1)$: $c_q = (c_a * c_x) + (c_x + c_1) * c_b$

Ciphertext and noise size expansion

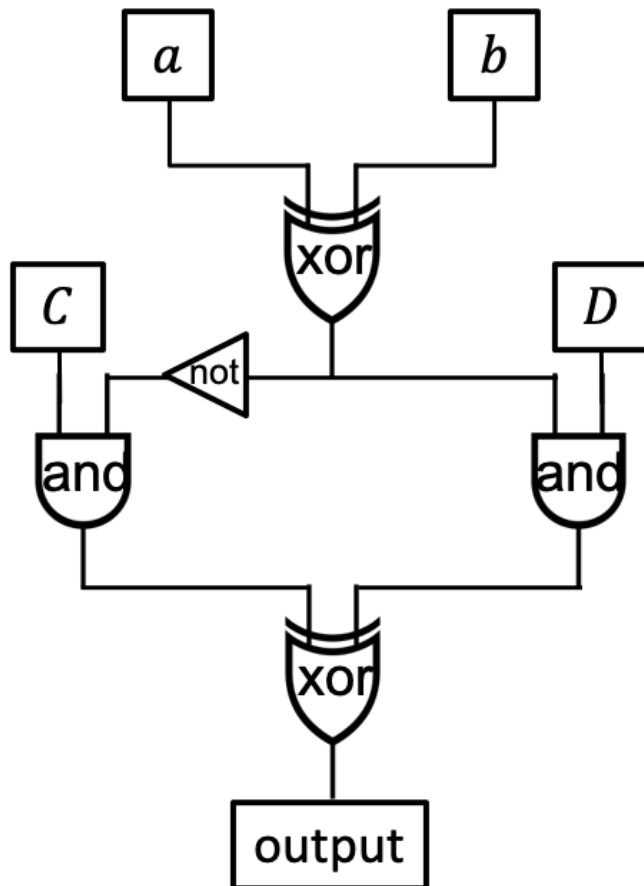


Source: A.r Maximov, "Homomorphic Encryption"

If noise grows too much. . .

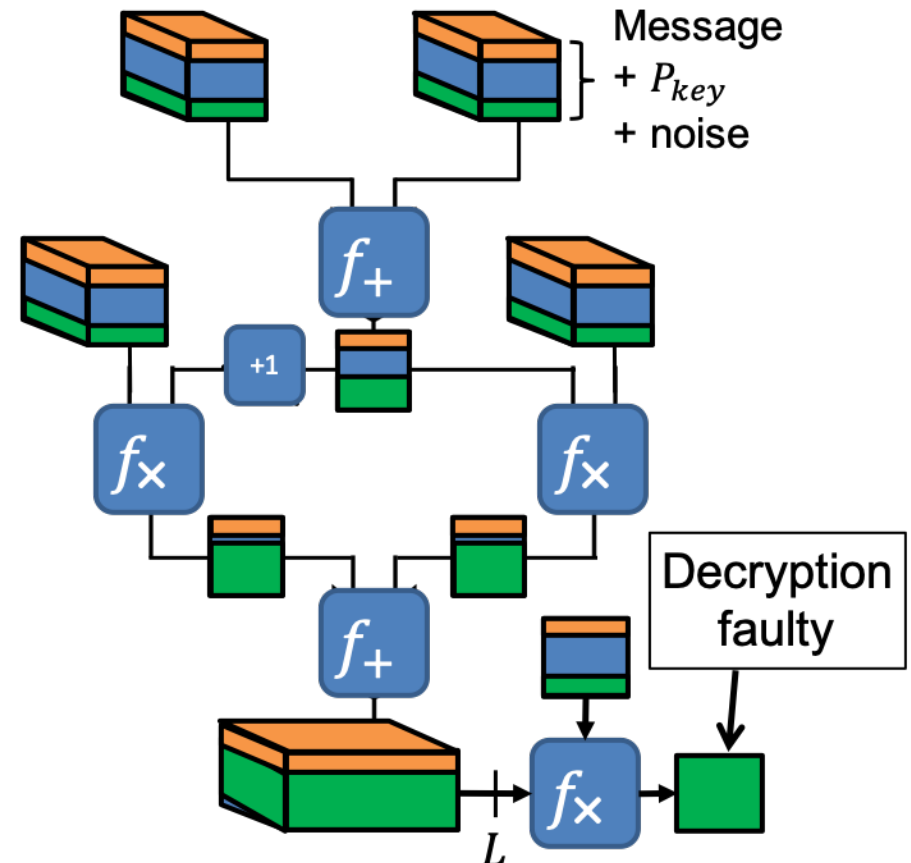
F computed on **plaintexts**

$$F(x_1, \dots, x_n)$$



F computed on **ciphertexts**

$$F(Enc(x_1), \dots, Enc(x_n))$$

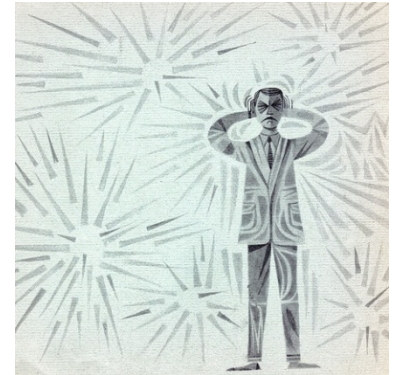


Source: C. Fontaine, "Fully Homomorphic Encryption Implementation Progresses and Challenges"

the noise grows!

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

*noise = 2 * (initial noise)*

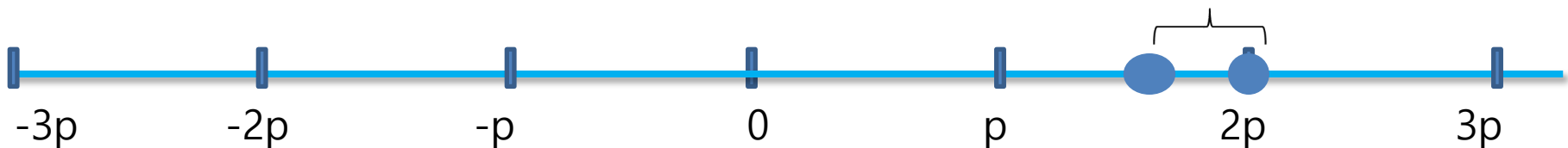


$$-c_1 c_2 = p \cdot (c_2 \cdot q_1 + c_1 \cdot q_2 - q_1 \cdot q_2) + \underbrace{2 \cdot (r_1 r_2 + r_1 b_2 + r_2 b_1) + b_1 b_2}_{\text{noise} = (\text{initial noise})^2}$$

noise = (initial noise)²



the "noise" = $2 \cdot r + b$



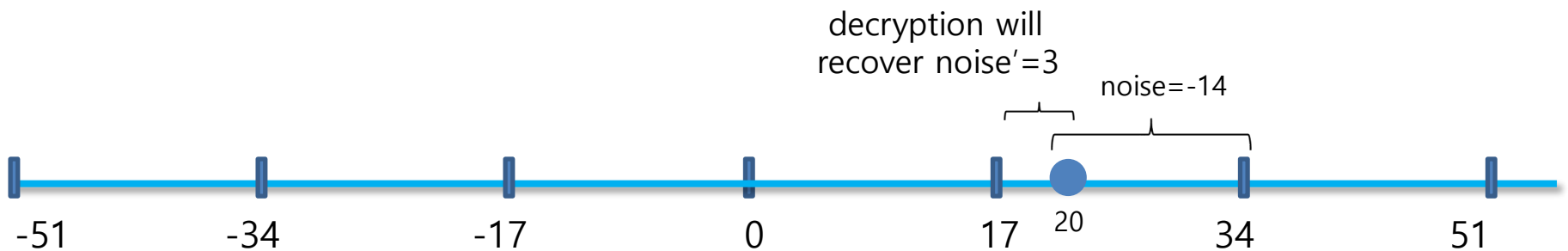
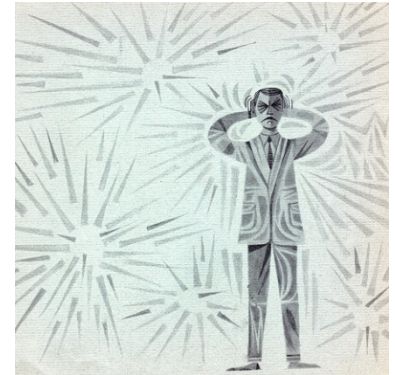
Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

the noise grows!

... so what's the problem?

If the $|noise| > p/2$, then ...

decryption will output an incorrect bit



Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

Noise Count

- The degree of noise can be calculated by the evaluator internally.
- Assume two arguments to an operation
 - c_1 – has the noise of degree n_1 bits
 - c_2 – has the noise of degree n_2 bits
- The result of:
 - $\text{ADD}(c_1, c_2) \Rightarrow$ noise of degree $\log_2(2^{n_1} + 2^{n_2})$ bits
 - $\text{MUL}(c_1, c_2) \Rightarrow$ noise of degree $(n_1 + n_2)$ bits
- Only when the noise would be larger than p -bits then the evaluator must do the “refreshment” step.

Source: A.r Maximov, “Homomorphic Encryption”

The Noise Problem

- **ADD:** $c = c_1 + c_2$.
 - Noise of c is $[c \bmod n] = [c_1 \bmod n] + [c_2 \bmod n]$
 - Unless this sum is bigger than n (decryption error).
- **MULT:** $c = c_1 \cdot c_2$.
 - Noise $[c \bmod n]$ is product of noises, unless product $> n$.
 - $(q_1n + e_1)(q_2n + e_2) = (q_1q_2n + q_1e_2 + q_2e_1)n + e_1e_2$
- **Function** $f: c = f(c_1, \dots, c_t)$.
 - Noise $[c \bmod n] = f([c_i \bmod n]_i)$ – i.e., f applied to noises.
 - Rough approximation:
 - Noise magnitude increases exponentially with degree of f .
- **The Noise Problem Hurts Efficiency. Why?**
 - Ciphertexts must be large to let noise “room to grow”.
 - Noise grows exponentially with degree.
 - Bit-length of noise grows linearly with degree.
 - Ciphertext size grows linearly with degree.

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.*

Gentry's "bootstrapping method"

... If you can go a (large) part of the way, then you can go all the way.

... but how?

[bootstrapping]

RSA&friends

Fully homomorphic

MANY mult
ZERO add

WE ARE HERE!

MANY add
MANY mult

Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

The "bootstrapping method"

Problem: Add and Mult increase noise
(Add doubles, Mult squares the noise)



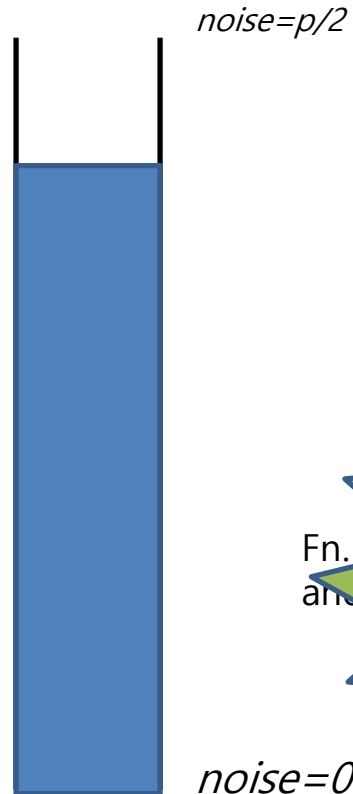
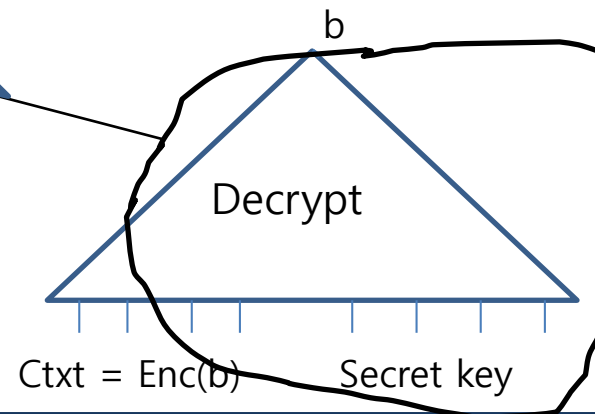
So, we want to do *noise-reduction*

Let's think...

... What is the best noise-reduction procedure?
... something that kills all noise
and recovers the message



Decryption!



Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

KEY IDEA:

- ... I want to reduce noise *without letting you decrypt*

... I cannot release the secret key (lest everyone sees my data)

... but I can release $\text{Enc}(\text{secret key})$

... called "Circular Encryption"

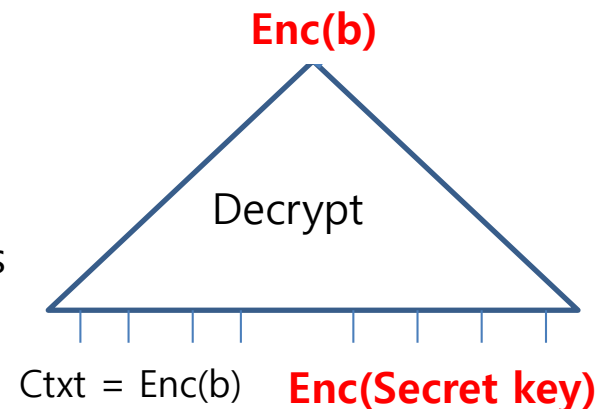


... Now, to reduce noise ...

... Homomorphically evaluate the decryption ckt!!!

KEY OBSERVATION:

... the input $\text{Enc}(b)$ and output $\text{Enc}(b)$ have different noise levels ...
Regardless of the noise in the input $\text{Enc}(b)$...
the noise level in the output $\text{Enc}(b)$ is **FIXED**



$\text{noise} = p/2$

$\text{noise} = 0$

Source: Vinod Vaikuntanathan, "Homomorphic Encryption: WHAT, WHY, and HOW"

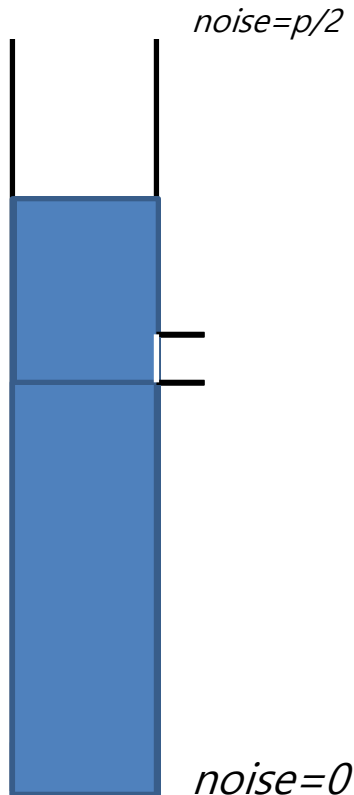
The “bootstrapping method”

Bottomline: whenever noise level increases beyond a limit ...

... use bootstrapping to reset it to a fixed level

Bootstrapping requires homomorphically evaluating the decryption circuit ...

... repeat until done



Thus Gentry’s “*bootstrapping theorem*”:
If an enc scheme can evaluate its own decryption circuit, then it can evaluate everything

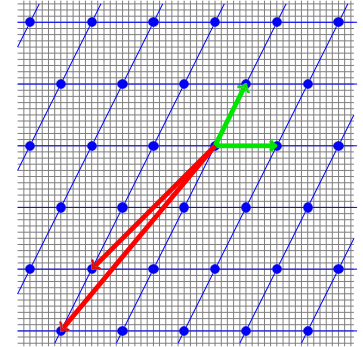
Source: Vinod Vaikuntanathan, “Homomorphic Encryption: WHAT, WHY, and HOW”

SOMEWHAT HOMOMORPHIC ENCRYPTION

Learning With Errors [Regev'05]

- Parameters: dimension n , modulus $q = \text{poly}(n)$, error distribution
- Search:** find secret $\mathbf{s} \in \mathbb{Z}_q^n$ given many 'noisy inner products'

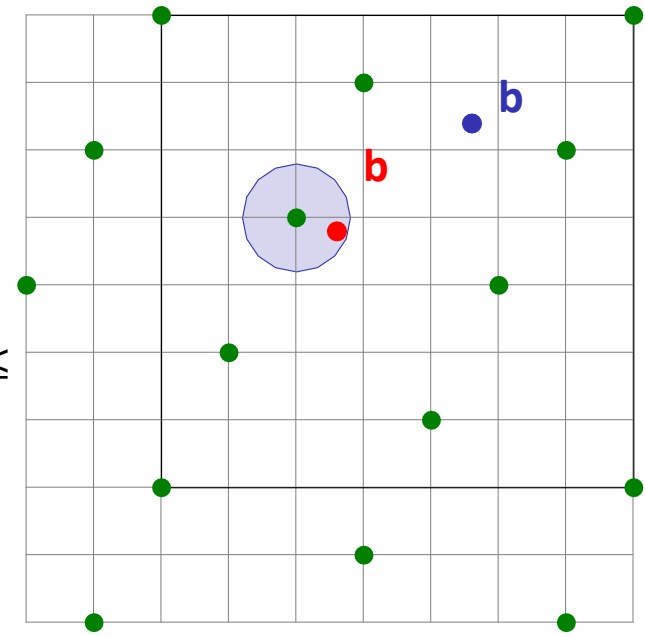
$$\underbrace{\left(\dots \quad \mathbf{A} \quad \dots \right)}_m \in \mathbb{Z}_q^{n \times m}, \quad \mathbf{b}^t = \mathbf{s}^t \mathbf{A} + \mathbf{e}^t \quad \text{OR} \quad \mathbf{b} \leftarrow \mathbb{Z}_q^m$$



- Decision:** distinguish (\mathbf{A}, \mathbf{b}) from uniform (\mathbf{A}, \mathbf{b})

LWE is Hard

$$\begin{array}{ccccc} (n/\alpha)\text{-approx } \textit{worst case} & \leq & \text{search-LWE} & \leq & \text{decision-LWE} \leq \\ \text{lattice problems} & & \text{crypto} & \uparrow & \\ & & \text{(quantum [R'05])} & & [\text{BFKL'93, R'05, ...}] \end{array}$$



Source: C. Peikert, "Lattice-Based Cryptography"

SWHE: Regev's Encryption Scheme

- Somewhat Homomorphic Encryption Based on LWE
 - Focusing on the Gentry-Sahai-Waters scheme.
 - Brakerski and Vaikuntanathan were the first to construct HE based on LWE(Learning With Errors)
- Regev's Encryption Scheme
 - **KeyGen**(1^n): Choose secret key $\mathbf{t} = (1, \mathbf{s})^t \in \mathbb{Z}_q^n$
 - Public key: $B \in \mathbb{Z}_q^{m \times n}$ random except $[B \times \mathbf{t}]_q$ "small"
 - **Encrypt**($B, \mu \in \{0,1\}$): For random $\mathbf{r} \in \{0,1\}^m$ output
 - $\mathbf{c} \leftarrow (\mu, 0, \dots, 0) \cdot \frac{q}{2} + \mathbf{r} \times B$
 - **Decrypt**(\mathbf{c}, \mathbf{t}): Compute
 - $\langle \mathbf{c}, \mathbf{t} \rangle = \mu \cdot \frac{q}{2} + \mathbf{r} \times B \times \mathbf{t} = \mu \cdot \frac{q}{2} + \text{"small"} \pmod{q}$
 - Recover μ as $\text{MSB}([\langle \mathbf{c}, \mathbf{t} \rangle]_q)$

Source: Shai Halevi, "Homomorphic Encryption"

Properties of Regev's Scheme

■ Properties of Regev's Scheme

- **Vectors**: Ciphertext \mathbf{c} and secret-key \mathbf{t} are vectors over Z_q , the 1st entry in \mathbf{t} is 1
- **Small dot-product**: If \mathbf{c} encrypts μ under \mathbf{t} then $[\langle \mathbf{c}, \mathbf{t} \rangle]_q = \mu \cdot \frac{q}{2} + \text{small}$
- **Ciphertexts are pseudorandom**: Under the hardness of decision-LWE, ciphertexts are indistinguishable from uniform vectors over Z_q to an attacker who doesn't know the secret key
- Homomorphic ADD in Regev
 - **Additive Homomorphism**: Add ciphertexts
 - If $[\langle \mathbf{c}_i, \mathbf{t} \rangle]_q = \mu_i \cdot \frac{q}{2} + \text{small}_i$ then $[\langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{t} \rangle]_q = (\mu_1 + \mu_2) \cdot \frac{q}{2} + \text{small}$ (if original *small*'s were small enough)
- Homomorphic MULT in Regev
 - **Multiplicative homomorphism**: multiply ciphertexts? How do you multiply vectors?
 - **Tensor product?** For $\mathbf{a} = (a_1, \dots, a_m), \mathbf{b} = (b_1, \dots, b_n)$, the tensor is $\mathbf{a} \otimes \mathbf{b} = (a_1 b_1, \dots, a_m b_n)$
 - Fact: $\langle \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{t} \otimes \mathbf{t} \rangle = \langle \mathbf{c}_1, \mathbf{t} \rangle \cdot \langle \mathbf{c}_2, \mathbf{t} \rangle$ (mixed prod.)
 - $\mathbf{c}_1 \otimes \mathbf{c}_2$ can be seen as encrypting $\mu_1 \cdot \mu_2$ under $\mathbf{t} \otimes \mathbf{t}$
 - Efficiency problem: MULT squares the dimension [Brakerski and Vaikuntanathan]
 - **Turn ciphertexts into matrices?** Use matrix product

Matrix Version (1st try)

- **KeyGen:** As before, secret key $\mathbf{t} = (1, \mathbf{s}) \in \mathbb{Z}_q^n$
- **Encrypt**($\mu \in \{0,1\}$): Choose $C_0 \in \mathbb{Z}_q^{n \times n}$
 - Rows are Regev-encryption of 0, $C_0 \times \mathbf{t} = \text{small}$
 - Output $C = \mu \cdot I + C_0$ (I is the identity)
 - Security: C_0 is pseudorandom (hence also C)
- **Decrypt_t**(C): Compute

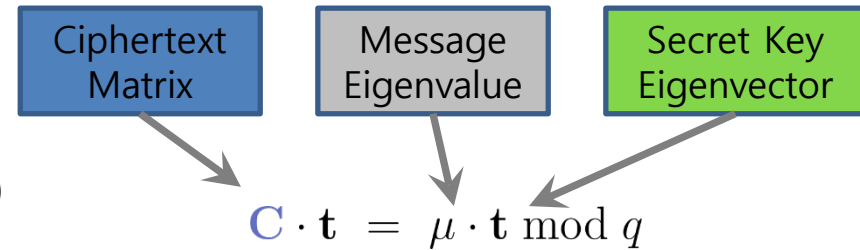
$$\mathbf{z} = [C \times \mathbf{t}]_q = (\mu \cdot I + C_0) \times \mathbf{t} = \mu \cdot \mathbf{t} + \text{small}$$
 - Output 0 if \mathbf{z} is small, 1 if \mathbf{z} is close to \mathbf{t}
- Secret key \mathbf{t} is *approximate eigenvector* of C
 - Message μ is the corresponding eigenvalue

Source: Shai Halevi, "Homomorphic Encryption"

Homomorphism

■ Homomorphism in Error-Free Setting

- If $C_i \times t = \mu_i \cdot t \pmod{q}$
- Then $(C_1 + C_2) \times t = (\mu_1 + \mu_2) \cdot t \pmod{q}$
- And $C_1 \times C_2 \times t = \mu_1 \cdot \mu_2 \cdot t \pmod{q}$




■ Homomorphism with Error (1st try)

- $C_i \times t = \mu_i \cdot t + e_i \pmod{q}$ for $i = 1, 2$
- **Addition:** $(C_1 + C_2) \times t = (\mu_1 + \mu_2) \cdot t + (e_1 + e_2)$
- **Multiplication:** $C^\times = C_1 \times C_2$

$$\begin{aligned}
 C^\times \times t &= C_1 \times (C_2 \times t) \\
 &= C_1 \times (\mu_2 \cdot t + e_2) \\
 &= \mu_2 \cdot (C_1 \times t) + C_1 \times e_2 \\
 &= \mu_2 \cdot (\mu_1 \cdot t + e_1) + C_1 \times e_2 \\
 &= \mu_1 \cdot \mu_2 \cdot t + (\mu_2 \cdot e_1 + C_1 \times e_2) \pmod{q}
 \end{aligned}$$

New Noise



Source: Shai Halevi, "Homomorphic Encryption"

Controlling the Noise

- $C^\times \times t = \mu_1 \cdot \mu_2 \cdot t + (\mu_2 \cdot e_1 + C_1 \times e_2)$ New Noise
- Keep messages μ small:
 - Easy! Restrict messages to $\{0,1\}$ and use NAND gates
- Keep ciphertext entries small:
 - Can "flatten" the product matrix C^\times to make its entries small using gadget G .
- Modified Matrix Encryption
 - Encryption of μ is a matrix $C \in Z_q^{m \times n}$ such that

$$C \times t = \mu \cdot \underbrace{(G \times t)}_{t'} + e$$
 - Easy to modify Enc to get this form
 - Set $C = \mu \cdot G + C_0$ rather than $C = \mu \cdot I + C_0$
 - Security follows from LWE as before
 - Additive homomorphism works just as before

Source: Shai Halevi, "Homomorphic Encryption"

Gadget for Flattening

- Use "Gadget Matrix" $G \in \mathbb{Z}_q^{m \times n}$ with associated "inverse transformation"
 $G^{-1}: \mathbb{Z}_q^{m \times n} \rightarrow \mathbb{Z}_q^{m \times m}$
 - Used for "lattice trapdoors" [A99,...,MP12]
- For any $C \in \mathbb{Z}_q^{m \times n}$:
 - $G^{-1}(C) \in \mathbb{Z}_q^{m \times m}$ is small
 - $G^{-1}(C) \times G = C$
- Example of a Gadget Matrix
 - $G^{-1}(C)$ breaks each entry in C to its bits

$$\begin{pmatrix} 7 & 4 \\ 2 & 5 \\ & \ddots \\ 1 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ & & \ddots & & & \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

- G has powers of two

$$G = \begin{pmatrix} 4 & 0 \\ 2 & 0 \\ 1 & 0 \\ 0 & 4 \\ 0 & 2 \\ 0 & 1 \end{pmatrix}$$

Source: Shai Halevi, "Homomorphic Encryption"

Homomorphic Multiplication

- $C_i \times t = \mu_i \cdot t' + e_i$ for $i = 1, 2$ ($t' = G \times t$)
- Set $C^\times = G^{-1}(C_1) \times C_2$

$$\begin{aligned}
 C^\times \times t &= G^{-1}(C_1) \times C_2 \times t \\
 &= G^{-1}(C_1) \times (\mu_2 \cdot t' + e_2) \\
 &= \mu_2 \cdot \underbrace{G^{-1}(C_1) \times (G \times t)}_{\text{small}} + G^{-1}(C_1) \times e_2 \\
 &= \mu_2 \cdot (C_1 \times t) + G^{-1}(C_1) \times e_2 \\
 &= \mu_2 \cdot (\mu_1 \cdot t' + e_1) + G^{-1}(C_1) \times e_2 \\
 &= \mu_1 \mu_2 \cdot t' + \underbrace{\mu_2 \cdot e_1 + G^{-1}(C_1) \times e_2}_{\text{small}} \pmod{q}
 \end{aligned}$$

New Noise

Source: Shai Halevi, "Homomorphic Encryption"

Summary of GSW

- Encryption of μ is $C \times t = \mu \cdot t' + e$ ($t' = G \times t$)
- Additive homomorphism: $C^+ = C_1 + C_2$
 - New noise is $|e^+| \leq |e_1| + |e_2| \leq 2 \cdot \max(|e_i|)$
- Multiplicative homomorphism: $C^\times = G^{-1}(C_1) \times C_2$
 - New noise is $|e^\times| \leq \mu_2 \cdot |e_1| + m \cdot |e_2|$
 $\leq (m + 1) \cdot \max(|e_i|)$

Somewhat homomorphic:

- can evaluate circuits with $\log_{m+1} q$ levels

Source: Shai Halevi, "Homomorphic Encryption"

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.