Homomorphic Encryption

Roman Walch

Privacy Enhancing Technologies – WT 2020/21

SCIENCE
PASSION
TECHNOLOGY

Outline

Introduction

Motivation: Privacy in Cloud Computing

Classical Crypto:

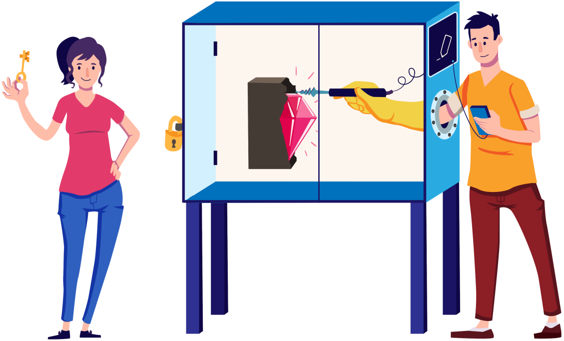
- ✔ Protect data in transmission
- ✔ Protect stored data
- ✘ Cannot manipulate encrypted data
  - ... secret decryption key required

Scenario: Using cloud services while maintaining privacy

- Outsourcing computation
- Access to pre-trained machine learning models
- Can we protect privacy of input items? (eHealth, etc.)
  - ... But cloud still wants to compute on input data

Homomorphic Encryption

- Operate on encrypted, unknown data
- Without knowing secret decryption key



Homomorphic Encryption (cont.)

Preserve plaintext properties in the ciphertext:

$$\mathcal{E}(x \star y) = \mathcal{E}(x) * \mathcal{E}(y)$$

Examples:

- Textbook RSA and ElGamal
  Homomorphic property is rather an unwanted side-effect
- Bilinear pairings
  A versatile building block with some homomorphic properties
- Fully homomorphic encryption
  Perform any computation on encrypted data

Homomorphic Encryption (cont.)

Preserve plaintext properties in the ciphertext:

$$\mathcal{E}(x \star y) = \mathcal{E}(x) * \mathcal{E}(y)$$

Examples:

- Textbook RSA and ElGamal
  Homomorphic property is rather an unwanted side-effect
- Bilinear pairings
  A versatile building block with some homomorphic properties
- Fully homomorphic encryption
  Perform any computation on encrypted data

This lecture

- Introduction into homomorphic encryption
  - ... concepts
  - ... schemes
  - ... optimizations
- Some HE concepts are math-heavy
  - This lecture does not give every detail
  - A lot of things are intentionally omitted
  - Talk to me after the lecture if interested ☺

Partial Homomorphic Encryption (PHE)

Partial Homomorphic Encryption

Allow evaluation of one operation on encrypted data:

- Support just addition or just multiplication, not both
- Multiplicative Homomorphic Encryption
  - RSA
- Additive Homomorphic Encryption
  - Paillier
  - Many practical applications just require addition
- Both schemes support arbitrary number of homomorphic operations

Multiplicative Homomorphic Encryption – RSA

- RSA (1977) is homomorphic with respect to multiplication
- Consider two different plaintext-ciphertext pairs:

$$C_1 = \mathcal{E}(M_1) = (M_1)^e \mod n$$
$$C_2 = \mathcal{E}(M_2) = (M_2)^e \mod n$$

Multiplicative homomorphism

$$\begin{aligned} C_1 \cdot C_2 &= ((M_1)^e \mod n) \cdot ((M_2)^e \mod n) \\ &= (M_1)^e \cdot (M_2)^e \mod n \\ &= (M_1 \cdot M_2)^e \mod n \end{aligned}$$

$$\mathcal{E}(\mathbf{M_1}) \cdot \mathcal{E}(\mathbf{M_2}) = \mathcal{E}(\mathbf{M_1} \cdot \mathbf{M_2})$$

Multiplicative Homomorphic Encryption – RSA

- RSA (1977) is homomorphic with respect to multiplication
- Consider two different plaintext-ciphertext pairs:

$$C_1 = \mathcal{E}(M_1) = (M_1)^e \mod n$$
$$C_2 = \mathcal{E}(M_2) = (M_2)^e \mod n$$

Multiplicative homomorphism

$$\begin{aligned} C_1 \cdot C_2 &= ((M_1)^e \mod n) \cdot ((M_2)^e \mod n) \\ &= (M_1)^e \cdot (M_2)^e \mod n \\ &= (M_1 \cdot M_2)^e \mod n \end{aligned}$$

$$\mathcal{E}(\mathbf{M_1}) \cdot \mathcal{E}(\mathbf{M_2}) = \mathcal{E}(\mathbf{M_1} \cdot \mathbf{M_2})$$

Additive Homomorphic Encryption – Paillier

- Proposed in 1999 by Pascal Paillier
- Consider two different plaintext-ciphertext pairs:

$$C_1 = \mathcal{E}(M_1) = g^{M_1} \cdot r_1^n \mod n^2$$
$$C_2 = \mathcal{E}(M_2) = g^{M_2} \cdot r_2^n \mod n^2$$

... with random r_1 and r_2

Additive homomorphism

$$C_1 \cdot C_2 = (g^{M_1} \cdot r_1^n \mod n^2) \cdot (g^{M_2} \cdot r_2^n \mod n^2)$$
$$= g^{M_1+M_2} \cdot (r_1 \cdot r_2)^n \bmod n^2$$

$$\mathcal{E}(\mathbf{M_1}) \cdot \mathcal{E}(\mathbf{M_2}) = \mathcal{E}(\mathbf{M_1} + \mathbf{M_2})$$

Additive Homomorphic Encryption – Paillier

- Proposed in 1999 by Pascal Paillier
- Consider two different plaintext-ciphertext pairs:

$$C_1 = \mathcal{E}(M_1) = g^{M_1} \cdot r_1^n \mod n^2$$
$$C_2 = \mathcal{E}(M_2) = g^{M_2} \cdot r_2^n \mod n^2$$

... with random r_1 and r_2

Additive homomorphism

$$\begin{aligned} C_1 \cdot C_2 &= (g^{M_1} \cdot r_1^n \mod n^2) \cdot (g^{M_2} \cdot r_2^n \mod n^2) \\ &= g^{M_1+M_2} \cdot (r_1 \cdot r_2)^n \bmod n^2 \end{aligned}$$

$$\mathcal{E}(\mathbf{M_1}) \cdot \mathcal{E}(\mathbf{M_2}) = \mathcal{E}(\mathbf{M_1} + \mathbf{M_2})$$

Paillier cryptosystem (cont.)

Further properties:

- Homomorphic plaintext addition:

$$C_1 \cdot g^{M_2} = \mathcal{E}(M_1 + M_2)$$

- Homomorphic plaintext multiplication:

$$C_1^{M_2} = \mathcal{E}(M_1 \cdot M_2)$$

- However: No ciphertext – ciphertext multiplication!
  - No way to get $\mathcal{E}(M_1 \cdot M_2)$ from C_1 and C_2

Real World Usage

- Part of private voting schemes:
  - Vote with options: {Yes, Abstain, No}
  - Send encrypted vote $v_i \in \{1, 0, -1\}$
  - Add all votes homomorphically
  - Decrypt final result
    - Positve result implies Yes
    - Negative result implies No
  - Danger:
    - Insecure without additional measures!

Real World Usage cont.

- Part of private voting schemes cont.:
  - Decrypting party has secret key:
    - Could decrypt single votes and learn content

    $\Rightarrow$ Use a different party to sum votes
  - Parties could encrypt values like 1000
    - Make your vote count more

    $\Rightarrow$ Include range proof
    - Zero-knowledge proof that encrypted value is in $\{1, 0, -1\}$
  - Actual schemes: Helios, ...
- Outsourced statistics:
  - matrix multiplication (ct-ct addition and ct-pt multiplication)

Real World Usage cont.

- Part of private voting schemes cont.:
  - Decrypting party has secret key:
    - Could decrypt single votes and learn content

    $\Rightarrow$ Use a different party to sum votes
  - Parties could encrypt values like 1000
    - Make your vote count more

    $\Rightarrow$ Include range proof
    - Zero-knowledge proof that encrypted value is in $\{1, 0, -1\}$
  - Actual schemes: Helios, ...
- Outsourced statistics:
  - matrix multiplication (ct-ct addition and ct-pt multiplication)

Real World Usage cont.

- Part of private voting schemes cont.:
  - Decrypting party has secret key:
    - Could decrypt single votes and learn content
    - $\Rightarrow$ Use a different party to sum votes
  - Parties could encrypt values like 1000
    - Make your vote count more
    - $\Rightarrow$ Include range proof
    - Zero-knowledge proof that encrypted value is in $\{1, 0, -1\}$
  - Actual schemes: Helios, ...
- Outsourced statistics:
  - matrix multiplication (ct-ct addition and ct-pt multiplication)

Fully Homomorphic Encryption (FHE)

Fully Homomorphic Encryption (FHE)

- Dream: All operations are possible on encrypted data

$$\forall \star \, \exists * : \mathcal{E}(M_1) * \mathcal{E}(M_2) = \mathcal{E}(M_1 \star M_2)$$

- Previous schemes only offer partial homomorphism
- Concept known since 1977 (RSA)
- Theoretical requirement:
  - Addition and multiplication
  - Arbitrary times
- It was not clear if a FHE scheme could even exist …
  - Often even called the “Holy Grail” of cryptography

Fully Homomorphic Encryption (FHE)

- Dream: All operations are possible on encrypted data

$$\forall \star \; \exists \ast : \mathcal{E}(M_1) \ast \mathcal{E}(M_2) = \mathcal{E}(M_1 \star M_2)$$

- Previous schemes only offer partial homomorphism
- Concept known since 1977 (RSA)
- Theoretical requirement:
  - Addition and multiplication
  - Arbitrary times
- It was not clear if a FHE scheme could even exist . . .
  - Often even called the “Holy Grail” of cryptography

Fully Homomorphic Encryption (cont.)

- 2009: Craig Gentry's PhD thesis
  - "A fully homomorphic encryption system"
  - Based on lattices and hard problems over lattices
    - More on Lattices $\rightarrow$ Seminar: Mathematical Foundations of Cryptography
  - One of the biggest advances in modern cryptography
- Since 2009:
  - Many variations and improvements
  - 3 generations of (F)HE schemes

Idea behind Gentry's Scheme

Basic concept has two parts:

1. Somewhat Homomorphic Encryption (SHE)
   - Allows limited number of operations
   - Here: many additions, few multiplications
2. Bootstrapping
   - Refresh ciphertext to allow additional operations
   - Repeat for unlimited operations

(F)HE schemes and libraries

- 1. Generation
  - Gentry's scheme from 2009
- 2. Generation
  - BGV: Integers, implemented in HElib
  - BFV: Integers, implemented in SEAL
  - CKKS: Floating point operations, implemented in HElib/SEAL
- 3. Generation
  - Schemes optimized for boolean circuits and fast bootstrapping
  - GSW, TFHE: implemented in TFHE library

Modern SHE schemes

Learning With Errors (LWE)

2./3. generation schemes are based on Learning With Errors hardness assumption:

Definition (Learning With Errors)

- Secret vector $\boldsymbol{s}$, many random vectors $\boldsymbol{a_i}$, small noise vector $\boldsymbol{e}$, all elements in $\mathbb{Z}_q$
- Calculate noisy inner products: $b_i = \langle \boldsymbol{a_i}, \boldsymbol{s} \rangle + e_i$

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \vdots \\ \boldsymbol{a_k} \end{pmatrix}, \qquad \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$$

- Given $(\boldsymbol{A}, \boldsymbol{b})$:
  - Hard to find $\boldsymbol{s}$
  - Hard to distinguish $(\boldsymbol{A}, \boldsymbol{b})$ from uniform random $(\boldsymbol{A}, \boldsymbol{r})$

Noise Propagation

- LWE based encryption introduces noise into ciphertext
  - Security comes from noise
- Homomorphic operations:
  - Addition: negligible noise growth
  - Multiplication: significant noise growth
- Decryption removes noise again
  - Decryption fails if noise is too large

$\Rightarrow$ Limited amount of multiplications!

Homomorphic Operations

2. Generation:

- Partial decryption[1]: $\mu = \langle \boldsymbol{c}, \boldsymbol{s} \rangle$
- Addition: $\mu_1 + \mu_2 = \langle \boldsymbol{c_1}, \boldsymbol{s} \rangle + \langle \boldsymbol{c_2}, \boldsymbol{s} \rangle = \langle \boldsymbol{c_1} + \boldsymbol{c_2}, \boldsymbol{s} \rangle$
- Multiplication[2]:
  - $\mu_1 \cdot \mu_2 = \langle \boldsymbol{c_1}, \boldsymbol{s} \rangle \cdot \langle \boldsymbol{c_2}, \boldsymbol{s} \rangle = \langle \boldsymbol{c_1} \otimes \boldsymbol{c_2}, \boldsymbol{s} \otimes \boldsymbol{s} \rangle$
  - Tensoring ciphertexts equivalent to multiplying plaintexts
  - However: result decryptable under $\boldsymbol{s} \otimes \boldsymbol{s}$

  $\Rightarrow$ Expensive relinearization required:

$$\mu_1 \cdot \mu_2 = \langle \textsc{Relin}(\boldsymbol{c_1} \otimes \boldsymbol{c_2}), \boldsymbol{s} \rangle$$

[1] First part of decryption for all 2. generation schemes

[2] $\otimes$ …tensorproduct

Homomorphic Operations

2. Generation:

- Partial decryption[1]: $\mu = \langle \boldsymbol{c}, \boldsymbol{s} \rangle$
- Addition: $\mu_1 + \mu_2 = \langle \boldsymbol{c_1}, \boldsymbol{s} \rangle + \langle \boldsymbol{c_2}, \boldsymbol{s} \rangle = \langle \boldsymbol{c_1} + \boldsymbol{c_2}, \boldsymbol{s} \rangle$
- Multiplication[2]:
  - $\mu_1 \cdot \mu_2 = \langle \boldsymbol{c_1}, \boldsymbol{s} \rangle \cdot \langle \boldsymbol{c_2}, \boldsymbol{s} \rangle = \langle \boldsymbol{c_1} \otimes \boldsymbol{c_2}, \boldsymbol{s} \otimes \boldsymbol{s} \rangle$
  - Tensoring ciphertexts equivalent to multiplying plaintexts
  - However: result decryptable under $\boldsymbol{s} \otimes \boldsymbol{s}$

  $\Rightarrow$ Expensive relinearization required:

$$\mu_1 \cdot \mu_2 = \langle \textsc{Relin}(\boldsymbol{c_1} \otimes \boldsymbol{c_2}), \boldsymbol{s} \rangle$$

[1] First part of decryption for all 2. generation schemes
[2] $\otimes$ …tensorproduct

Homomorphic Operations (cont.)

3. Generation:

- Decryption based on eigenvectors/eigenvalues of Matrices
- If $\boldsymbol{s}$ is an eigenvector of $\boldsymbol{C}$ with eigenvalue μ, then:

$$\boldsymbol{C} \cdot \boldsymbol{s} = \mu \cdot \boldsymbol{s}$$

- Let $\boldsymbol{C}$ be the ciphertext, $\boldsymbol{s}$ the decryption key and μ the corresponding (noisy) plaintext:
  - Addition: $(\boldsymbol{C_1} + \boldsymbol{C_2}) \cdot \boldsymbol{s} = (\mu_1 + \mu_2) \cdot \boldsymbol{s}$
  - Multiplication: $(\boldsymbol{C_1} \cdot \boldsymbol{C_2}) \cdot \boldsymbol{s} = (\mu_1 \cdot \mu_2) \cdot \boldsymbol{s}$

Optimization: Polynomial Rings

- Polynomial Ring: $R_q = \mathbb{Z}_q[x]/(x^N + 1)$
  - Polynomials of $\deg < N$ and coefficients $\bmod q$
  - Elements $a \in R_q$ have form:
    $a = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{N-1} \cdot x^{N-1} = \sum_{i=0}^{N-1} a_i \cdot x^i, \qquad a_i \in \mathbb{Z}_q$
- LWE: $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$
  - ... with matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$, $\boldsymbol{s}$, $\boldsymbol{e}$, elements in $\mathbb{Z}_q$

$\Rightarrow$ Ring-LWE: $b = a \cdot s + e$

  - ... with $a, b, s, e \in R_q$
  - ... equivalent to LWE, when $\boldsymbol{A}$ is build from a

Optimization: Polynomial Rings

- Polynomial Ring: $R_q = \mathbb{Z}_q[x]/(x^N + 1)$
  - Polynomials of $\deg < N$ and coefficients $\bmod q$
  - Elements $a \in R_q$ have form:
    $a = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{N-1} \cdot x^{N-1} = \sum_{i=0}^{N-1} a_i \cdot x^i, \qquad a_i \in \mathbb{Z}_q$
- LWE: $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$
  - ... with matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}, \boldsymbol{s}, \boldsymbol{e}$, elements in $\mathbb{Z}_q$

$\Rightarrow$ Ring-LWE: $b = a \cdot s + e$

  - ... with $a, b, s, e \in R_q$
  - ... equivalent to LWE, when $\boldsymbol{A}$ is build from a

HE using Polynomial Rings

- In practice only Ring-variants relevant
  - Homomorphic operations rewritten to equivalent Ring operations
- Advantages:
  - Smaller public key (matrix $\boldsymbol{A}$ vs. $a \in R_q$)
  - SIMD Packing (later this lecture)
  - Faster multiplications (number theoretic transformation)

BGV

HE over $\mathbb{Z}_t$

BGV

- Parameters:
  - N: degree of reduction polynomial $(x^N + 1)$
  - t: Plaintext modulus
    - HE operations correspond to operations in $\mathbb{Z}_t$
  - q: Ciphertext modulus
    - Coefficient of ciphertext polynomials in $\mathbb{Z}_q$
- Key Generation: Key is essentially a Ring-LWE instance

$$
\begin{aligned}
& s, a, e \leftarrow R_q, \qquad b = a \cdot s + t \cdot e \\
\Rightarrow\ & pk = (b, a), \qquad sk = s
\end{aligned}
$$

BGV (cont.)

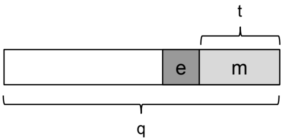
- Encrypt $m \in R_t$:

$$e_0, e_1 \leftarrow R_q, \qquad v \leftarrow R_2$$
$$c = \mathcal{E}_{pk}(m) = (c_0, c_1) = (v \cdot b + t \cdot e_0 + m, v \cdot a + t \cdot e_1)$$

- Decrypt:

$$m = \mathcal{D}_{sk}(c) = (c_0 - s \cdot c_1) \bmod t$$

- Coefficients of $(c_0 - s \cdot c_1)$:



BGV (cont.)

- Add(c, c'):

$$c_{add} = (c_0 + c'_0, c_1 + c'_1)$$

- Mul(c, c'):

$$c_{mul} = (\tilde{c_0}, \tilde{c_1}, \tilde{c_2}) = (c_0 \cdot c'_0, c_0 \cdot c'_1 + c_1 \cdot c'_0, c_1 \cdot c'_1)$$

$$\text{with } \mathcal{D}_{sk}(c_{mul}) = \tilde{c_0} - s \cdot \tilde{c_1} - s^2 \cdot \tilde{c_2}$$

  - Requires relinearization with a relinearization key (rk):

$$c_{mul} = \textsc{Relin}_{rk}(\tilde{c_0}, \tilde{c_1}, \tilde{c_2}) = (c_{mul,0}, c_{mul,1})$$

- Noise:

BGV (cont.)

- Add(c, c'):

$$c_{add} = (c_0 + c'_0, c_1 + c'_1)$$

- Mul(c, c'):

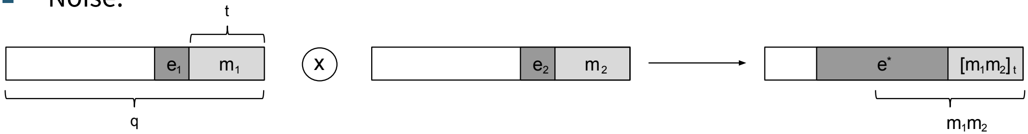
$$c_{mul} = (\tilde{c_0}, \tilde{c_1}, \tilde{c_2}) = (c_0 \cdot c'_0, c_0 \cdot c'_1 + c_1 \cdot c'_0, c_1 \cdot c'_1)$$

with $\mathcal{D}_{sk}(c_{mul}) = \tilde{c_0} - s \cdot \tilde{c_1} - s^2 \cdot \tilde{c_2}$

  - Requires relinearization with a relinearization key (rk):

$$c_{mul} = \textsc{Relin}_{rk}(\tilde{c_0}, \tilde{c_1}, \tilde{c_2}) = (c_{mul,0}, c_{mul,1})$$

- Noise:



Plaintext Encoding

- Plaintexts are polynomials $\in R_t$
- We want homomorphic encryption over $\mathbb{Z}_t$

  $\Rightarrow$ Scalars $a \in \mathbb{Z}_t$ need to be encoded before encryption!
- Many encodings exist
  - Scalar encoding
  - SIMD encoding
  - Integer encoding (optimized Scalar encoding)
  - Fractional encoding
  - ...

Scalar Encoding

- Encode plaintext $p \in \mathbb{Z}_t$ in constant term of the polynomial $p' \in R_t$
- Set other coefficients to 0

$\Rightarrow$ Addition and multiplication equal to operations in $\mathbb{Z}_t$

$$(p_0 + 0 \cdot x + \cdots + 0 \cdot x^{N-1}) \cdot (p_1 + 0 \cdot x + \cdots + 0 \cdot x^{N-1}) = (p_2 + 0 \cdot x + \cdots + 0 \cdot x^{N-1})$$

- Simple but inefficient
  - Unused coefficients, except constant term

SIMD Encoding

- Encodes a vector of integers $\in \mathbb{Z}_t$ into one polynomial $\in R_t$
  - ... via Chinese Remainder Theorem
  - Size of vector depends on HE parameters (several thousands possible)
- Addition/multiplication correspond to slotwise vector operations
  - Similar to Single Instruction Multiple Data (SIMD) instructions on CPU's

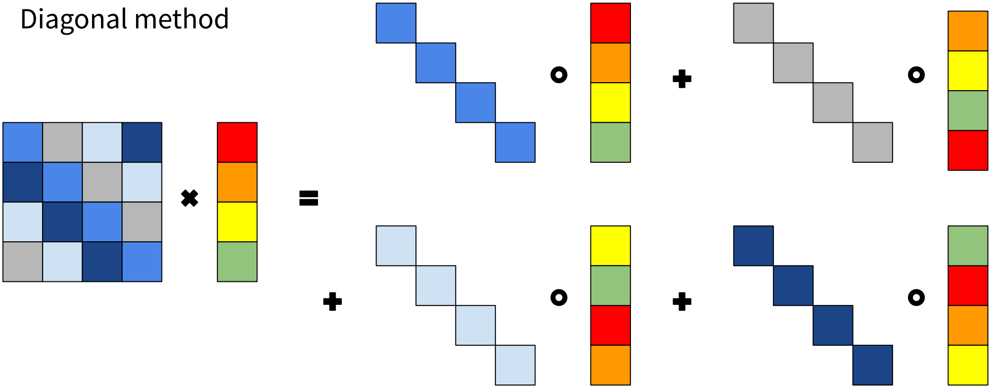
$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix} \overset{\text{encode}}{\rightarrow} a \in R_t, \qquad \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \overset{\text{encode}}{\rightarrow} b \in R_t : \qquad (a \cdot b) \overset{\text{decode}}{\rightarrow} \begin{bmatrix} a_1 \cdot b_1 \\ a_2 \cdot b_2 \\ \vdots \\ a_k \cdot b_k \end{bmatrix}$$

SIMD Encoding (cont.)

- Further operations:
  - Slot rotation
  - ... requires rotation key for each index
- However:
  - No access to individual slots
- Usage:
  - Optimize throughput (thousands operations in parallel)
  - Minimize latency by using slots to speed up one calculation
  - e.g.: Diagonal method for matrix-vector multiplication

Plain Matrix Times Encrypted Vector

- Diagonal method



- Requires *N* elementwise multiplications and *N* rotations
  - Furhter optimizable via Babystep-Giantstep algorithm

Plaintext Space

- Plaintexts $m \in \mathbb{Z}_t$
  - Integers modulo t
  - Only addition and multiplications
  - No comparison, branching, ect.
- Plaintexts $m \in \mathbb{Z}_2 = \{0, 1\}$
  - Possible Operations:

+	0	1
0	0	1
1	1	0

$\Rightarrow$ Addition equal to XOR gate

$\cdot$	0	1
0	0	0
1	0	1

$\Rightarrow$ Multiplication equal to AND gate

Plaintext Space

- Plaintexts $m \in \mathbb{Z}_t$
  - Integers modulo t
  - Only addition and multiplications
  - No comparison, branching, ect.
- Plaintexts $m \in \mathbb{Z}_2 = \{0, 1\}$
  - Possible Operations:

+	0	1
0	0	1
1	1	0

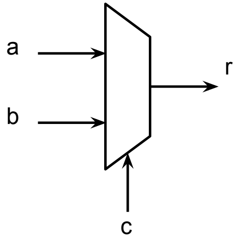
$\Rightarrow$ Addition equal to XOR gate

$\cdot$	0	1
0	0	0
1	0	1

$\Rightarrow$ Multiplication equal to AND gate

Binary Circuits

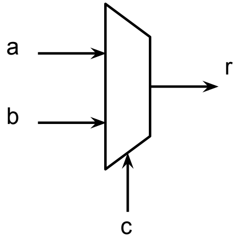
- Plaintext space $\mathbb{Z}_2$ supports evaluation of arbitrary binary circuits
  - ... realize every function f with arbitrary precision
- Multiplexer: $r = c\ ?\ a : b$
  - $r = b + c \cdot (a - b)$
  - multiplicative depth $d = 1$



- n-bit Adder:
  - Ripple Carry Adder (depth $d = n - 1$)
  - Carry Lookahead Adder (depth $d = \mathcal{O}(\log(n))$)
    - Depth-optimized, in total more additions/multiplications

Binary Circuits

- Plaintext space $\mathbb{Z}_2$ supports evaluation of arbitrary binary circuits
  - ... realize every function f with arbitrary precision
- Multiplexer: $r = c\ ?\ a : b$
  - $r = b + c \cdot (a - b)$
  - multiplicative depth $d = 1$
- n-bit Adder:
  - Ripple Carry Adder (depth $d = n - 1$)
  - Carry Lookahead Adder (depth $d = \mathcal{O}(\log(n))$)
    - Depth-optimized, in total more additions/multiplications



Binary Circuits (cont.)

- $n \times n = n$ bit multiplication:
  - multiplicative depth $d = n$
  - Worse for $n \times n = 2n$ bit
- Can also implement floating point operations, division, etc.

Plaintext Space (cont.)

- $\mathbb{Z}_2$ or $\mathbb{Z}_t$?
  - depends on use case!
- n-bit integer Addition/Multiplication:
  - 1 operation in $\mathbb{Z}_t$
  - a lot more in $\mathbb{Z}_2 \Rightarrow$ can be much slower!
- Branching/Comparisons
  - only possible in $\mathbb{Z}_2$!
- Switching between $\mathbb{Z}_2$ and $\mathbb{Z}_t$ only possible through decryption

CKKS

Approximated HE

CKKS – Approximated HE

- FHE problem: No floats

$\Rightarrow$ Fixed-point arithmetic: $3.1415 \to 314$ with scale $\Delta = 100$

- Multiplication:

$$(314, \Delta = 100) \cdot (272, \Delta = 100) = (85408, \Delta = 10000) \to 8.5408$$

  - Scale grows
  - Noise grows (Ring-LWE)
  - Big plaintext parts reserved for insignificant LSBs
    - $85408, \Delta = 10000 \Rightarrow 08$ rather insignificant
    - Worse with bigger scale

CKKS (cont.)

- Idea – Rounding:
  - Rounding operation after multiplication

$$
\begin{aligned}
\textsc{Round}((314, \Delta = 100) \cdot (272, \Delta = 100)) \\
= \textsc{Round}(85408, \Delta = 10000) \\
= (854, \Delta = 100) \rightarrow 8.54
\end{aligned}
$$

$\Rightarrow$ Rounding achieves:
  - Scale stays constant
  - Discards insignificant LSBs

- Idea – Encode noise in LSBs:
  - LSBs insignificant anyways
  - Rounding reduces noise

CKKS (cont.)

- Idea – Rounding:
  - Rounding operation after multiplication

$$\begin{aligned}\textsc{Round}((314, \Delta = 100) \cdot (272, \Delta = 100)) \\ = \textsc{Round}(85408, \Delta = 10000) \\ = (854, \Delta = 100) \rightarrow 8.54\end{aligned}$$

$\Rightarrow$ Rounding achieves:

  - Scale stays constant
  - Discards insignificant LSBs
- Idea – Encode noise in LSBs:
  - LSBs insignificant anyways
  - Rounding reduces noise

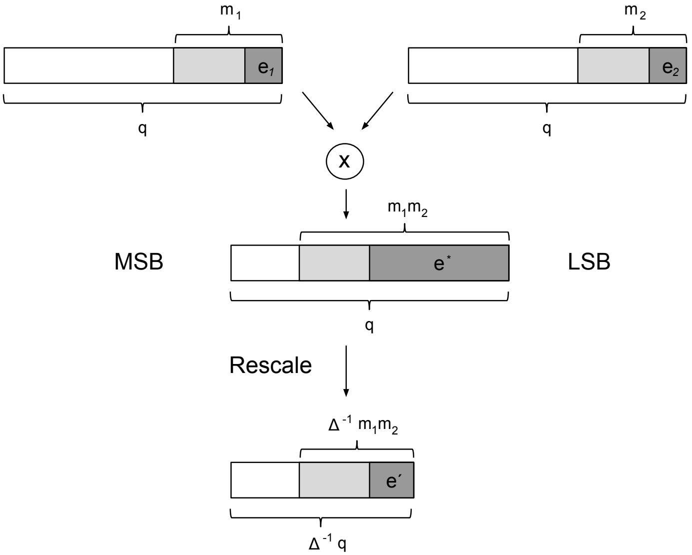
CKKS – Rescale

- Rescale operation:
  - Divide ciphertext by scale: $ct' = ct/\Delta$
  - Divide modulus by scale: $q_{\ell'} = q_\ell/\Delta$
- Result:
  - Rescale achieves rounding!
  - Rescale reduces noise
  - Smaller ciphertext modulus q after rescale

$\Rightarrow$ Size of q limits number of rescale operations

  - Limits the number of multiplications

CKKS – Rescale after Multiplication



CKKS (cont.)

- Similar to floating point operations in plain
- Result includes approximation error
- Security based on Ring-LWE
  - Plaintexts are polynomials in $R = \mathbb{Z}[x]/(x^N + 1)$
  - Supports SIMD packing
  - Supports vector rotations

$\Rightarrow$ Most promising scheme for HE machine learning

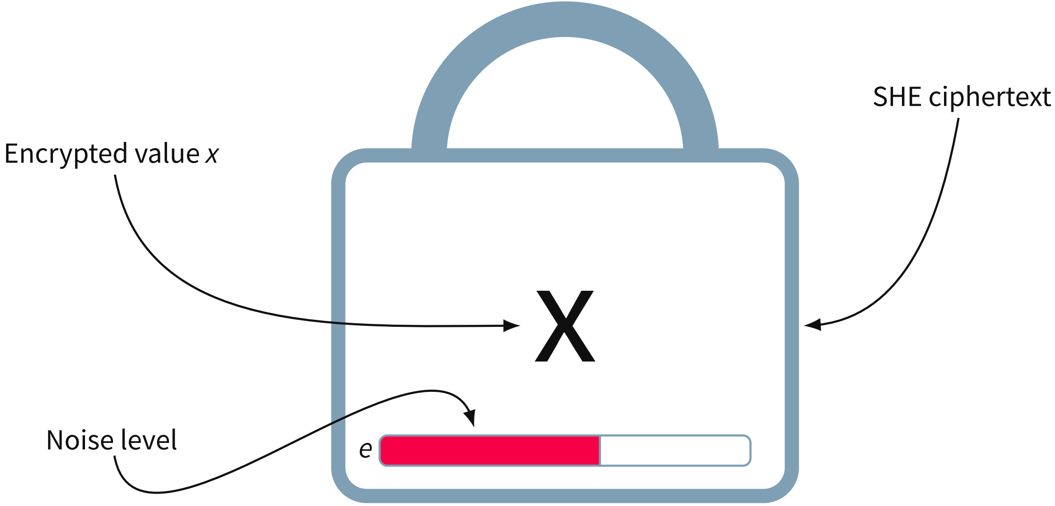
Bootstrapping

From SHE to FHE

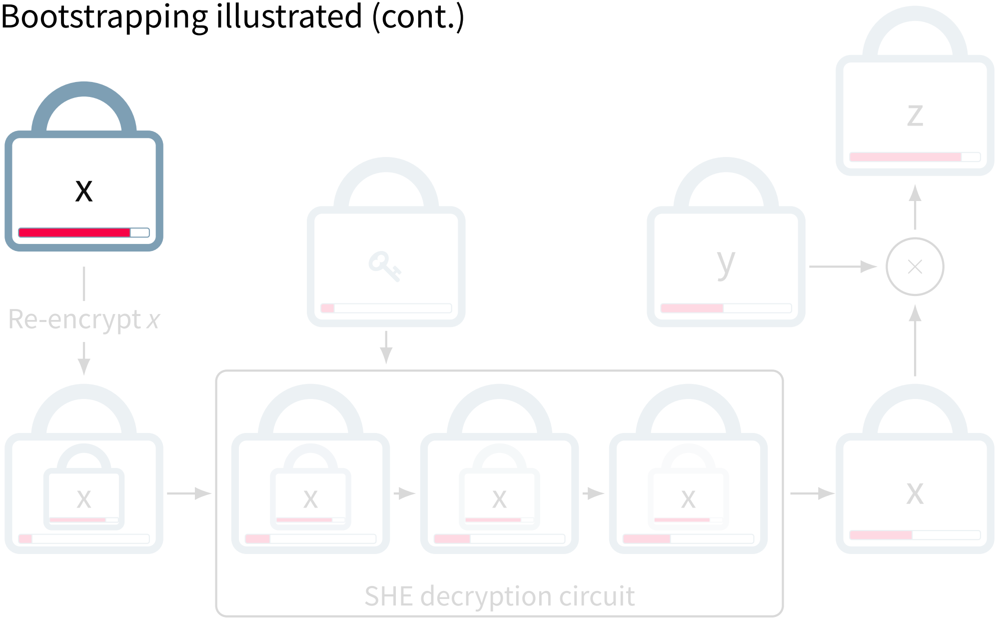
Bootstrapping

- "Re-crypt" ciphertext into new ciphertext with lower noise
  - Encrypt ciphertext again
  - Encrypt secret key
  - Homomorphically evaluate decryption circuit
- Choose parameters so we can evaluate the decryption circuit + 1 more gate
  - Can evaluate anything by repeated bootstrapping!
  - Requires shallow decryption circuit

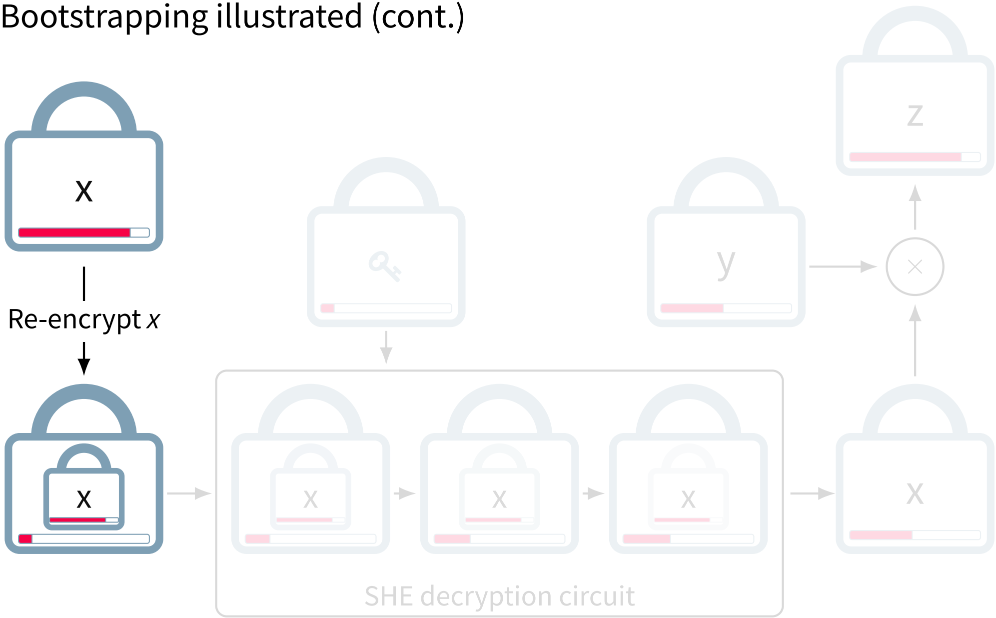
Bootstrapping illustrated



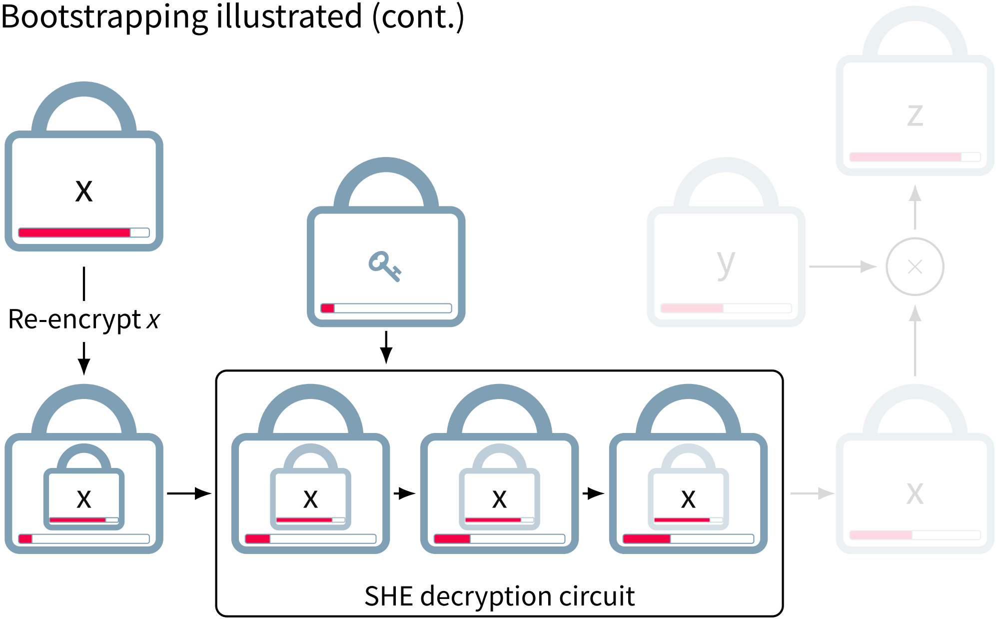
Bootstrapping illustrated (cont.)



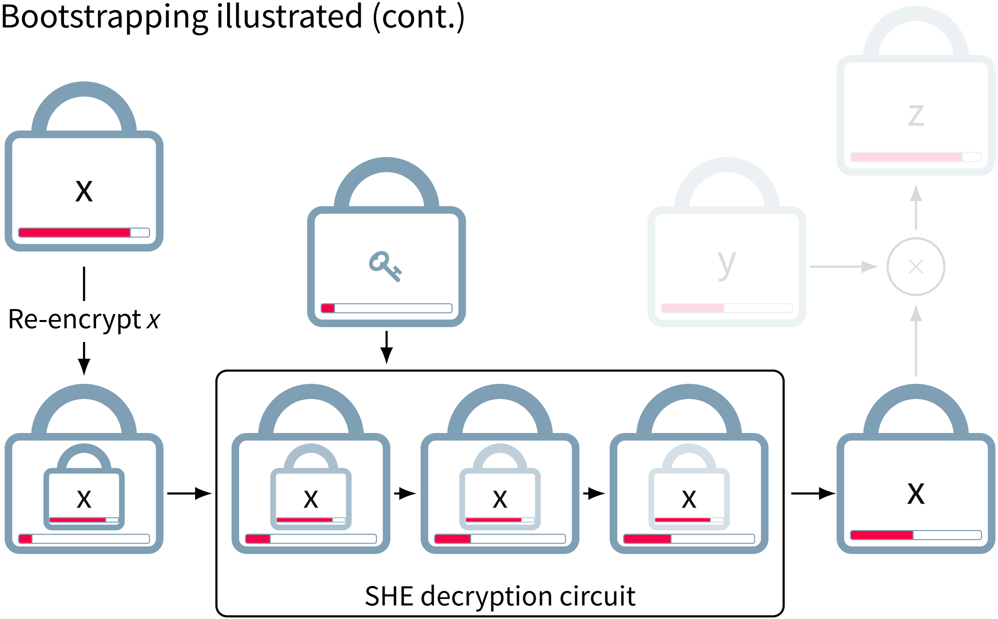
Bootstrapping illustrated (cont.)



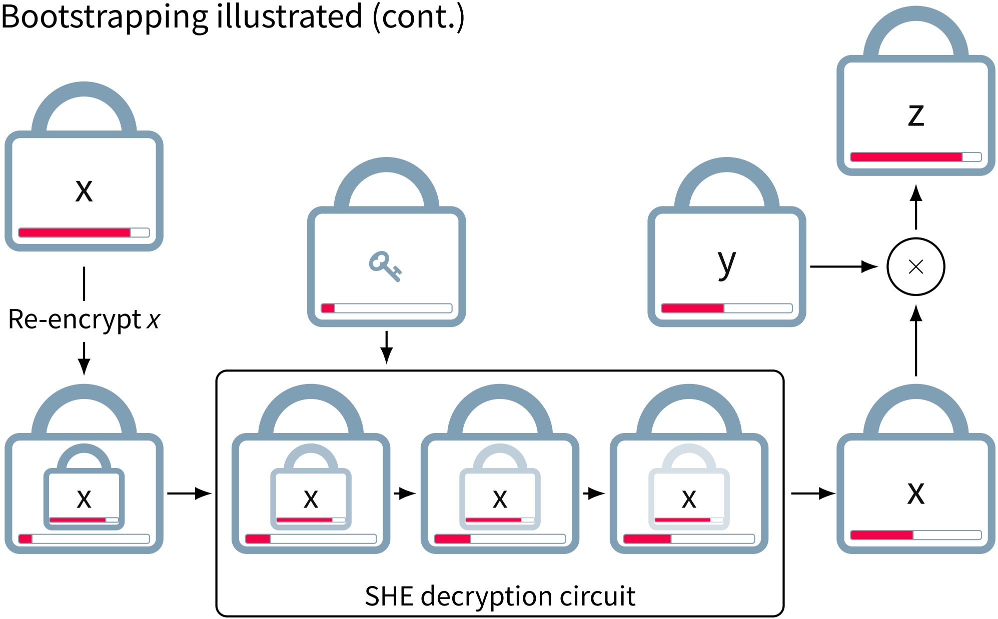
Bootstrapping illustrated (cont.)



Bootstrapping illustrated (cont.)



Bootstrapping illustrated (cont.)



SHE vs. FHE

- Bootstrapping
  - Very high performance overhead
  - Especially with big plaintext moduli
  - In many libraries not even implemented

$\Rightarrow$ In practice:
  - Use SHE with parameters big enough for use-case
  - For binary circuits: Use TFHE

TFHE

FHE for binary circuits

TFHE

- 3rd generation FHE scheme
  - Optimized for boolean circuits
  - Fast gate-bootstrapping
- One parameter set for a given security level
  - Enough for one boolean gate $+$ bootstrapping
  - Easy parameter selection
- However:
  - No SIMD-packing
  - No plaintexts in $\mathbb{Z}_t$ with $t > 2$

TFHE (cont.)

- Easy to use library
  - For given boolean circuits
- Many different gates:
  - AND, OR, XOR, …
  - Multiplexer
- Implementations in
  - C++
  - Rust
  - Nvidia graphic cards (CUDA)

Outlook

HE Parameters

- Plaintext modulus t
  - Boolean circuits?
  - Big enough, such that no computation overflows?
  - Bigger $\rightarrow$ more noise added
- Ciphertext modulus q
  - Defines available noise budget
  - Bigger $\rightarrow$ more noise budget, but less security!
- Reduction Polynomial degree N
  - Bigger $\rightarrow$ increases security (i.e., allows bigger q)
  - But: significantly increases runtime!

HE Parameters (cont.)

- Trade-off:
  - Security vs. performance vs. noise budget
- In TFHE:
  - One parameter set
  - But only boolean gates without packing
- Other:
  - Tedious parameter selection
  - Depends on use case!
  - Multiplicative depth

Practical Considerations

- Homomorphic operations:
  - Addition
  - Multiplication
  - Vector rotation
- How to calculate e.g. $\mathrm{ReLU}(x) = \mathtt{max}(0, x)$?

$\Rightarrow$ Approximate using polynomials!

$$\mathrm{ReLU}(x) \approx 0.1061 + 0.5000 \cdot x + 0.4244 \cdot x^2$$

- Consider degree (i.e., multiplicative depth)

Conclusion

- Homomorphic Encryption is powerful
  - ... allows to operate on encrypted data
  - ... but difficult to use
  - ... but still slow
- Schemes based on (R)LWE
  - Problem: noise management (SHE vs. FHE)
- Different schemes for
  - Integer arithmetic (BFV, BGV)
  - Floating point arithmetic (CKKS)
  - Boolean circuits (TFHE)

Questions

Bibliography I

[20] **Microsoft SEAL (release 3.5).** https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Apr. 2020.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. **(Leveled) fully homomorphic encryption without bootstrapping.** ITCS. ACM, 2012, pp. 309–325.

[Bra12] Zvika Brakerski. **Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP.** CRYPTO. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 868–886.

[CGGI16a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. **Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds.** ASIACRYPT (1). Vol. 10031. Lecture Notes in Computer Science. 2016, pp. 3–33.

[CGGI16b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. **TFHE: Fast Fully Homomorphic Encryption Library.** https://tfhe.github.io/tfhe/. 2016.

Bibliography II

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. **Homomorphic Encryption for Arithmetic of Approximate Numbers.** ASIACRYPT (1). Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 409–437.

[FV12] Junfeng Fan and Frederik Vercauteren. **Somewhat Practical Fully Homomorphic Encryption.** IACR Cryptology ePrint Archive 2012 (2012), p. 144.

[Gen09] Craig Gentry. **Fully homomorphic encryption using ideal lattices.** STOC. ACM, 2009, pp. 169–178.

[HS14] Shai Halevi and Victor Shoup. **Algorithms in HElib.** CRYPTO (1). Vol. 8616. Lecture Notes in Computer Science. Springer, 2014, pp. 554–571.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. **On Ideal Lattices and Learning with Errors over Rings.** EUROCRYPT. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 1–23.

[Pai99] Pascal Paillier. **Public-Key Cryptosystems Based on Composite Degree Residuosity Classes.** EUROCRYPT. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238.

Bibliography III

[Reg05] Oded Regev. **On lattices, learning with errors, random linear codes, and cryptography**. STOC. ACM, 2005, pp. 84–93.