A Survey of Fully Homomorphic Encryption

Jean-Sébastien Coron

University of Luxembourg

June 1st, 2017

Homomorphic Encryption

- Homomorphic encryption: perform operations on plaintexts while manipulating only ciphertexts.
  - Normally, this is not possible.

$$\begin{aligned} \mathrm{AES}_K(m_1) &= \texttt{0x3c7317c6bc5634a4ad8479c64714f4f8} \\ \mathrm{AES}_K(m_2) &= \texttt{0x7619884e1961b051be1aa407da6cac2c} \\ \mathrm{AES}_K(m_1 \oplus m_2) &= \ ? \end{aligned}$$

- For some cryptosystems with algebraic structure, this is possible. For example RSA:

$$\left. \begin{aligned} c_1 &= {m_1}^e \bmod N \\ c_2 &= {m_2}^e \bmod N \end{aligned} \right. \Rightarrow c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

Homomorphic Encryption

- Homomorphic encryption: perform operations on plaintexts while manipulating only ciphertexts.
  - Normally, this is not possible.

$$\begin{aligned}\mathrm{AES}_K(m_1) &= \texttt{0x3c7317c6bc5634a4ad8479c64714f4f8}\\ \mathrm{AES}_K(m_2) &= \texttt{0x7619884e1961b051be1aa407da6cac2c}\\ \mathrm{AES}_K(m_1 \oplus m_2) &= ?\end{aligned}$$

- For some cryptosystems with algebraic structure, this is possible. For example RSA:

$$\left.\begin{array}{l} c_1 = {m_1}^e \bmod N \\ c_2 = {m_2}^e \bmod N \end{array}\right. \Rightarrow c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

Homomorphic Encryption

- Homomorphic encryption: perform operations on plaintexts while manipulating only ciphertexts.
  - Normally, this is not possible.

$$
\begin{aligned}
\mathrm{AES}_K(m_1) &= \texttt{0x3c7317c6bc5634a4ad8479c64714f4f8} \\
\mathrm{AES}_K(m_2) &= \texttt{0x7619884e1961b051be1aa407da6cac2c} \\
\mathrm{AES}_K(m_1 \oplus m_2) &= \;?
\end{aligned}
$$

- For some cryptosystems with algebraic structure, this is possible. For example RSA:

$$
\left.\begin{aligned} c_1 &= {m_1}^e \bmod N \\ c_2 &= {m_2}^e \bmod N \end{aligned}\right. \Rightarrow c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N
$$

Homomorphic Encryption with RSA

- Multiplicative property of RSA.

$$\begin{aligned} c_1 &= {m_1}^e \bmod N \\ c_2 &= {m_2}^e \bmod N \end{aligned} \Rightarrow c = c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

- Homomorphic encryption: given c_1 and c_2, we can compute the ciphertext c for $m_1 \cdot m_2 \bmod N$
  - using only the public-key
  - without knowing the plaintexts m_1 and m_2.

Homomorphic Encryption with RSA

- Multiplicative property of RSA.

$$\begin{aligned} c_1 &= {m_1}^e \bmod N \\ c_2 &= {m_2}^e \bmod N \end{aligned} \Rightarrow c = c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

- Homomorphic encryption: given c_1 and c_2, we can compute the ciphertext c for $m_1 \cdot m_2 \bmod N$
  - using only the public-key
  - without knowing the plaintexts m_1 and m_2.

Paillier Cryptosystem

- Additively homomorphic: Paillier cryptosystem

$$\begin{array}{l} c_1 = g^{m_1} \bmod N^2 \\ c_2 = g^{m_2} \bmod N^2 \end{array} \Rightarrow c_1 \cdot c_2 = g^{m_1+m_2\,[N]} \bmod N^2$$

- Application: e-voting.
  - Voter i encrypts his vote $m_i \in \{0,1\}$ into:

$$c_i = g^{m_i} \cdot z_i^N \bmod N^2$$

  - Votes can be aggregated using only the public-key:

$$c = \prod_i c_i = g^{\sum_i m_i} \cdot z \bmod N^2$$

  - c is enventually decrypted to recover $m = \sum_i m_i$

Paillier Cryptosystem

- Additively homomorphic: Paillier cryptosystem

$$\begin{aligned} c_1 &= g^{m_1} \bmod N^2 \\ c_2 &= g^{m_2} \bmod N^2 \end{aligned} \Rightarrow c_1 \cdot c_2 = g^{m_1+m_2\ [N]} \bmod N^2$$

- Application: e-voting.
  - Voter i encrypts his vote $m_i \in \{0,1\}$ into:

$$c_i = g^{m_i} \cdot z_i^N \bmod N^2$$

  - Votes can be aggregated using only the public-key:

$$c = \prod_i c_i = g^{\sum_i m_i} \cdot z \bmod N^2$$

  - c is enventually decrypted to recover $m = \sum_i m_i$

Fully homomorphic encryption

- Multiplicatively homomorphic: RSA.

$$\begin{array}{l} c_1 = {m_1}^e \bmod N \\ c_2 = {m_2}^e \bmod N \end{array} \Rightarrow c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

- Additively homomorphic: Paillier

$$\begin{array}{l} c_1 = g^{m_1} \bmod N^2 \\ c_2 = g^{m_2} \bmod N^2 \end{array} \Rightarrow c_1 \cdot c_2 = g^{m_1+m_2 \, [N]} \bmod N^2$$

- Fully homomorphic: homomorphic for both addition and multiplication
  - Open problem until Gentry's breakthrough in 2009.

Fully homomorphic encryption

- Multiplicatively homomorphic: RSA.

$$\begin{array}{l} c_1 = {m_1}^e \bmod N \\ c_2 = {m_2}^e \bmod N \end{array} \Rightarrow c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

- Additively homomorphic: Paillier

$$\begin{array}{l} c_1 = g^{m_1} \bmod N^2 \\ c_2 = g^{m_2} \bmod N^2 \end{array} \Rightarrow c_1 \cdot c_2 = g^{m_1 + m_2 \,[N]} \bmod N^2$$

- Fully homomorphic: homomorphic for both addition and multiplication
  - Open problem until Gentry's breakthrough in 2009.

Fully homomorphic encryption

- Multiplicatively homomorphic: RSA.

$$\begin{array}{l} c_1 = {m_1}^e \bmod N \\ c_2 = {m_2}^e \bmod N \end{array} \Rightarrow c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

- Additively homomorphic: Paillier

$$\begin{array}{l} c_1 = g^{m_1} \bmod N^2 \\ c_2 = g^{m_2} \bmod N^2 \end{array} \Rightarrow c_1 \cdot c_2 = g^{m_1+m_2\ [N]} \bmod N^2$$

- Fully homomorphic: homomorphic for both addition and multiplication
  - Open problem until Gentry's breakthrough in 2009.

Fully homomorphic public-key encryption

- We restrict ourselves to public-key encryption of a single bit:
  - $0 \rightarrow 203\text{ef}6124\ldots23\text{ab}87_{16}$
  - $1 \rightarrow \text{b}327653\text{c}1\ldots\text{db}3265_{16}$
  - Obviously, encryption must be probabilistic.
- Fully homomorphic property
  - Given $E(b_0)$ and $E(b_1)$, one can compute $E(b_0 \oplus b_1)$ and $E(b_0 \cdot b_1)$ without knowing the private-key.
- Why is it important ?
  - Universality: any Boolean circuit can be written with Xors and Ands.
  - Once you can homomorphically evaluate both a Xor and a And, you can evaluate any Boolean circuit, any computable function.

Fully homomorphic public-key encryption

- We restrict ourselves to public-key encryption of a single bit:
  - $0 \rightarrow 203ef6124\ldots23ab87_{16}$
  - $1 \rightarrow b327653c1\ldots db3265_{16}$
  - Obviously, encryption must be probabilistic.
- Fully homomorphic property
  - Given $E(b_0)$ and $E(b_1)$, one can compute $E(b_0 \oplus b_1)$ and $E(b_0 \cdot b_1)$ without knowing the private-key.
- Why is it important ?
  - Universality: any Boolean circuit can be written with Xors and Ands.
  - Once you can homomorphically evaluate both a Xor and a And, you can evaluate any Boolean circuit, any computable function.

Fully homomorphic public-key encryption

- We restrict ourselves to public-key encryption of a single bit:
  - $0 \to 203\text{ef}6124\ldots23\text{ab}87_{16}$
  - $1 \to \text{b}327653\text{c}1\ldots\text{db}3265_{16}$
  - Obviously, encryption must be probabilistic.
- Fully homomorphic property
  - Given $E(b_0)$ and $E(b_1)$, one can compute $E(b_0 \oplus b_1)$ and $E(b_0 \cdot b_1)$ without knowing the private-key.
- Why is it important ?
  - Universality: any Boolean circuit can be written with Xors and Ands.
  - Once you can homomorphically evaluate both a Xor and a And, you can evaluate any Boolean circuit, any computable function.

Outsourcing Computation

- The cloud receives some data m in encrypted form.
  - It receives the ciphertexts c_i corresponding to bits m_i
  - The cloud doesn't know the m_i's
- The cloud performs some computation $f(m)$, but without knowing m
  - The computation of f is written as a Boolean circuit with Xors and Ands
  - Every Xor $z = x \oplus y$ is homomorphically evaluated from the ciphertexts c_x and c_y, to get ciphertext c_z
  - Every And $z' = x \cdot y$ is homomorphically evaluated from the ciphertexts c_x and c_y, to get ciphertext $c_{z'}$
- Eventally the cloud obtains a ciphertext c for $f(m)$
  - The user decrypts c to recover $f(m)$
  - The cloud learns nothing about m

Outsourcing Computation

- The cloud receives some data m in encrypted form.
  - It receives the ciphertexts c_i corresponding to bits m_i
  - The cloud doesn't know the m_i's
- The cloud performs some computation $f(m)$, but without knowing m
  - The computation of f is written as a Boolean circuit with Xors and Ands
  - Every Xor $z = x \oplus y$ is homomorphically evaluated from the ciphertexts c_x and c_y, to get ciphertext c_z
  - Every And $z' = x \cdot y$ is homomorphically evaluated from the ciphertexts c_x and c_y, to get ciphertext $c_{z'}$
- Eventally the cloud obtains a ciphertext c for $f(m)$
  - The user decrypts c to recover $f(m)$
  - The cloud learns nothing about m

Outsourcing Computation

- The cloud receives some data m in encrypted form.
  - It receives the ciphertexts c_i corresponding to bits m_i
  - The cloud doesn't know the m_i's
- The cloud performs some computation $f(m)$, but without knowing m
  - The computation of f is written as a Boolean circuit with Xors and Ands
  - Every Xor $z = x \oplus y$ is homomorphically evaluated from the ciphertexts c_x and c_y, to get ciphertext c_z
  - Every And $z' = x \cdot y$ is homomorphically evaluated from the ciphertexts c_x and c_y, to get ciphertext $c_{z'}$
- Eventally the cloud obtains a ciphertext c for $f(m)$
  - The user decrypts c to recover $f(m)$
  - The cloud learns nothing about m

What fully homomorphic encryption brings you

- You have a software that given the revenue, past income, headcount, etc., of a company can predict its future stock price.
  - I want to know the future stock price of my company, but I don't want to disclose confidential information.
  - And you don't want to give me your software containing secret formulas.
- Using homomorphic encryption:
  - I encrypt all the inputs using fully homomorphic encryption and send them to you in encrypted form.
  - You process all my inputs, viewing your software as a circuit.
  - You send me the result, still encrypted.
  - I decrypt the result and get the predicted stock price.
  - You didn't learn any information about my company.
- More generally:
  - Cool buzzwords like secure cloud computing.
  - Cool mathematical challenges.

What fully homomorphic encryption brings you

- You have a software that given the revenue, past income, headcount, etc., of a company can predict its future stock price.
  - I want to know the future stock price of my company, but I don't want to disclose confidential information.
  - And you don't want to give me your software containing secret formulas.
- Using homomorphic encryption:
  - I encrypt all the inputs using fully homomorphic encryption and send them to you in encrypted form.
  - You process all my inputs, viewing your software as a circuit.
  - You send me the result, still encrypted.
  - I decrypt the result and get the predicted stock price.
  - You didn't learn any information about my company.
- More generally:
  - Cool buzzwords like secure cloud computing.
  - Cool mathematical challenges.

What fully homomorphic encryption brings you

- You have a software that given the revenue, past income, headcount, etc., of a company can predict its future stock price.
  - I want to know the future stock price of my company, but I don't want to disclose confidential information.
  - And you don't want to give me your software containing secret formulas.
- Using homomorphic encryption:
  - I encrypt all the inputs using fully homomorphic encryption and send them to you in encrypted form.
  - You process all my inputs, viewing your software as a circuit.
  - You send me the result, still encrypted.
  - I decrypt the result and get the predicted stock price.
  - You didn't learn any information about my company.
- More generally:
  - Cool buzzwords like secure cloud computing.
  - Cool mathematical challenges.

Cloud Computing

- Goal: cloud computing
  - I encrypt my data before sending it to the cloud
  - The cloud can still search, sort and edit my data on my behalf
  - Data is kept in encrypted form in the cloud.
  - The cloud learns nothing about my data
- The cloud returns encrypted answers
  - that only I can decrypt

Fully Homomorphic Encryption Schemes

- 1. Breakthrough scheme of Gentry [G09], based on ideal lattices. Some optimizations by [SV10].
  - Implementation [GH11]: PK size: 2.3 GB, recrypt: 30 min.
- 2. RLWE schemes [BV11a,BV11b].
  - FHE without bootstrapping (modulus switching) [BGV11]
  - Batch FHE [GHS12]
  - Implementation with homomorphic evaluation of AES [GHS12]
  - And many other papers...
- 3. van Dijk, Gentry, Halevi and Vaikuntanathan's scheme over the integers [DGHV10].
  - Implementation [CMNT11]: PK size: 1 GB, recrypt: 15 min.
  - Public-key compression and modulus switching [CNT12]
  - Batch and homomorphic evaluation of AES [CCKLLTY13].

Fully Homomorphic Encryption Schemes

- 1. Breakthrough scheme of Gentry [G09], based on ideal lattices. Some optimizations by [SV10].
  - Implementation [GH11]: PK size: 2.3 GB, recrypt: 30 min.
- 2. RLWE schemes [BV11a,BV11b].
  - FHE without bootstrapping (modulus switching) [BGV11]
  - Batch FHE [GHS12]
  - Implementation with homomorphic evaluation of AES [GHS12]
  - And many other papers...
- 3. van Dijk, Gentry, Halevi and Vaikuntanathan's scheme over the integers [DGHV10].
  - Implementation [CMNT11]: PK size: 1 GB, recrypt: 15 min.
  - Public-key compression and modulus switching [CNT12]
  - Batch and homomorphic evaluation of AES [CCKLLTY13].

Fully Homomorphic Encryption Schemes

- 1. Breakthrough scheme of Gentry [G09], based on ideal lattices. Some optimizations by [SV10].
  - Implementation [GH11]: PK size: 2.3 GB, recrypt: 30 min.
- 2. RLWE schemes [BV11a,BV11b].
  - FHE without bootstrapping (modulus switching) [BGV11]
  - Batch FHE [GHS12]
  - Implementation with homomorphic evaluation of AES [GHS12]
  - And many other papers...
- 3. van Dijk, Gentry, Halevi and Vaikuntanathan's scheme over the integers [DGHV10].
  - Implementation [CMNT11]: PK size: 1 GB, recrypt: 15 min.
  - Public-key compression and modulus switching [CNT12]
  - Batch and homomorphic evaluation of AES [CCKLLTY13].

The DGHV Scheme

- Ciphertext for $m \in \{0,1\}$:

$$c = q \cdot p + 2r + m$$

  where p is the secret-key, q and r are randoms.

- Decryption:

$$(c \bmod p) \bmod 2 = m$$

- Parameters:

The DGHV Scheme

- Ciphertext for $m \in \{0, 1\}$:

$$c = q \cdot p + 2r + m$$

  where p is the secret-key, q and r are randoms.
- Decryption:

$$(c \bmod p) \bmod 2 = m$$

- Parameters:

The DGHV Scheme

- Ciphertext for $m \in \{0,1\}$:

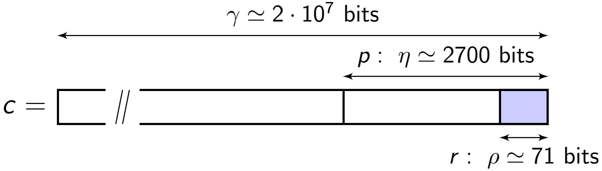
$$c = q \cdot p + 2r + m$$

where p is the secret-key, q and r are randoms.

- Decryption:

$$(c \bmod p) \bmod 2 = m$$

- Parameters:



Homomorphic Properties of DGHV

- Addition:

$$\begin{matrix} c_1 = q_1 \cdot p + 2r_1 + m_1 \\ c_2 = q_2 \cdot p + 2r_2 + m_2 \end{matrix} \Rightarrow c_1 + c_2 = q' \cdot p + 2r' + m_1 + m_2$$

  - $c_1 + c_2$ is an encryption of $m_1 + m_2 \bmod 2 = m_1 \oplus m_2$

- Multiplication:

$$\begin{matrix} c_1 = q_1 \cdot p + 2r_1 + m_1 \\ c_2 = q_2 \cdot p + 2r_2 + m_2 \end{matrix} \Rightarrow c_1 \cdot c_2 = q'' \cdot p + 2r'' + m_1 \cdot m_2$$

  with

$$r'' = 2r_1 r_2 + r_1 m_2 + r_2 m_1$$

  - $c_1 \cdot c_2$ is an encryption of $m_1 \cdot m_2$
  - Noise becomes twice larger.

Homomorphic Properties of DGHV

- Addition:

$$\begin{array}{l} c_1 = q_1 \cdot p + 2r_1 + m_1 \\ c_2 = q_2 \cdot p + 2r_2 + m_2 \end{array} \Rightarrow c_1 + c_2 = q' \cdot p + 2r' + m_1 + m_2$$

  - $c_1 + c_2$ is an encryption of $m_1 + m_2 \bmod 2 = m_1 \oplus m_2$

- Multiplication:

$$\begin{array}{l} c_1 = q_1 \cdot p + 2r_1 + m_1 \\ c_2 = q_2 \cdot p + 2r_2 + m_2 \end{array} \Rightarrow c_1 \cdot c_2 = q'' \cdot p + 2r'' + m_1 \cdot m_2$$

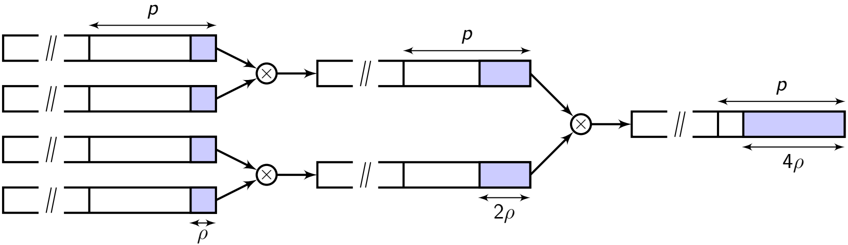
  with

$$r'' = 2r_1 r_2 + r_1 m_2 + r_2 m_1$$

  - $c_1 \cdot c_2$ is an encryption of $m_1 \cdot m_2$
  - Noise becomes twice larger.

Somewhat homomorphic scheme

- The number of multiplications is limited.
  - Noise grows with the number of multiplications.
  - Noise must remain $< p$ for correct decryption.



Gentry's technique

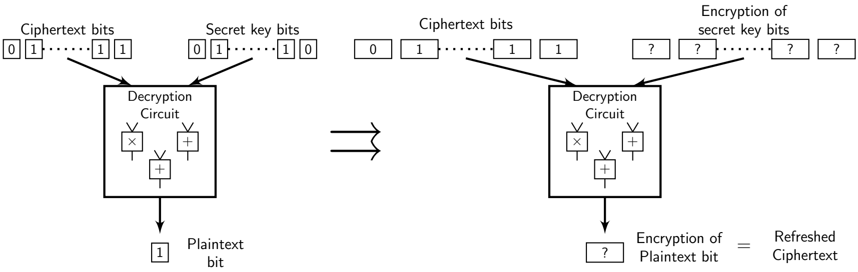
- To build a FHE scheme, start from the somewhat homomorphic scheme, that is:
    - Only a polynomial of small degree can be homomorphically applied on ciphertexts.
    - Otherwise the noise becomes too large and decryption becomes incorrect.
- Then, "squash" the decryption procedure:
    - express the decryption function as a low degree polynomial in the bits of the ciphertext c and the secret key sk (equivalently a boolean circuit of small depth).

Gentry's technique

- To build a FHE scheme, start from the somewhat homomorphic scheme, that is:
  - Only a polynomial of small degree can be homomorphically applied on ciphertexts.
  - Otherwise the noise becomes too large and decryption becomes incorrect.
- Then, "squash" the decryption procedure:
  - express the decryption function as a low degree polynomial in the bits of the ciphertext c and the secret key sk (equivalently a boolean circuit of small depth).

Ciphertext refresh: bootstrapping

- Gentry's breakthrough idea: refresh the ciphertext using the decryption circuit homomorphically.
  - Evaluate the decryption polynomial not on the bits of the ciphertext c and the secret key sk, but homomorphically on the encryption of those bits.
  - Instead of recovering the bit plaintext m, one gets an encryption of this bit plaintext, *i.e.* yet another ciphertext for the same plaintext.



Ciphertext refresh

- Refreshed ciphertext:
  - If the degree of the decryption polynomial is small enough, the resulting noise in this new ciphertext can be smaller than in the original ciphertext
- Fully homomorphic encryption:
  - Given two refreshed ciphertexts one can apply again the homomorphic operation (either addition or multiplication), which was not necessarily possible on the original ciphertexts because of the noise threshold.
  - Using this "ciphertext refresh" procedure the number of homomorphic operations becomes unlimited and we get a fully homomorphic encryption scheme.

Ciphertext refresh

- Refreshed ciphertext:
  - If the degree of the decryption polynomial is small enough, the resulting noise in this new ciphertext can be smaller than in the original ciphertext
- Fully homomorphic encryption:
  - Given two refreshed ciphertexts one can apply again the homomorphic operation (either addition or multiplication), which was not necessarily possible on the original ciphertexts because of the noise threshold.
  - Using this "ciphertext refresh" procedure the number of homomorphic operations becomes unlimited and we get a fully homomorphic encryption scheme.

Public-key Encryption with DGHV

- Ciphertext

$$c = q \cdot p + 2r + m$$

- Public-key: a set of τ encryptions of 0's.

$$x_i = q_i \cdot p + 2r_i$$

- Public-key encryption:

$$c = m + 2r + \sum_{i=1}^{\tau} \varepsilon_i \cdot x_i$$

for random $\varepsilon_i \in \{0, 1\}$.

Public-key Encryption with DGHV

- Ciphertext

$$c = q \cdot p + 2r + m$$

- Public-key: a set of τ encryptions of 0's.

$$x_i = q_i \cdot p + 2r_i$$

- Public-key encryption:

$$c = m + 2r + \sum_{i=1}^{\tau} \varepsilon_i \cdot x_i$$

for random $\varepsilon_i \in \{0, 1\}$.

Public-key Encryption with DGHV

- Ciphertext

$$c = q \cdot p + 2r + m$$

- Public-key: a set of τ encryptions of 0's.

$$x_i = q_i \cdot p + 2r_i$$

- Public-key encryption:

$$c = m + 2r + \sum_{i=1}^{\tau} \varepsilon_i \cdot x_i$$

for random $\varepsilon_i \in \{0, 1\}$.

The squashed scheme from DGHV

- The basic decryption $m \leftarrow (c \bmod p) \bmod 2$ cannot be directly expressed as a boolean circuit of low depth.
- Alternative decryption formula for $c = q \cdot p + 2r + m$
  - We have $q = \lfloor c/p \rceil$ and $c = q + m \pmod 2$
  - Therefore

$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$

- Idea (Gentry, DGHV). Secret-share $1/p$ as a sparse subset sum:

$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

The squashed scheme from DGHV

- The basic decryption $m \leftarrow (c \bmod p) \bmod 2$ cannot be directly expressed as a boolean circuit of low depth.
- Alternative decryption formula for $c = q \cdot p + 2r + m$
  - We have $q = \lfloor c/p \rceil$ and $c = q + m \pmod 2$
  - Therefore

$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$

- Idea (Gentry, DGHV). Secret-share $1/p$ as a sparse subset sum:

$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

The squashed scheme from DGHV

- The basic decryption $m \leftarrow (c \bmod p) \bmod 2$ cannot be directly expressed as a boolean circuit of low depth.
- Alternative decryption formula for $c = q \cdot p + 2r + m$
  - We have $q = \lfloor c/p \rceil$ and $c = q + m \pmod 2$
  - Therefore

$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$

- Idea (Gentry, DGHV). Secret-share $1/p$ as a sparse subset sum:

$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

Squashed decryption

- Alternative equation

$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$

- Secret-share $1/p$ as a sparse subset sum:

$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

with random public κ-bit numbers y_i, and sparse secret $s_i \in \{0,1\}$.

- Decryption becomes:

$$m \leftarrow [c]_2 \oplus \left[\left\lfloor \sum_{i=1}^{\Theta} s_i \cdot (y_i \cdot c) \right\rceil \right]_2$$

Squashed decryption

- Alternative equation

$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$

- Secret-share $1/p$ as a sparse subset sum:

$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

with random public κ-bit numbers y_i, and sparse secret $s_i \in \{0, 1\}$.

- Decryption becomes:

$$m \leftarrow [c]_2 \oplus \left[\left\lfloor \sum_{i=1}^{\Theta} s_i \cdot (y_i \cdot c) \right\rceil \right]_2$$

Squashed decryption

- Alternative equation

$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$

- Secret-share $1/p$ as a sparse subset sum:

$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

with random public κ-bit numbers y_i, and sparse secret $s_i \in \{0, 1\}$.

- Decryption becomes:

$$m \leftarrow [c]_2 \oplus \left[\left\lfloor \sum_{i=1}^{\Theta} s_i \cdot (y_i \cdot c) \right\rceil \right]_2$$

Squashed decryption

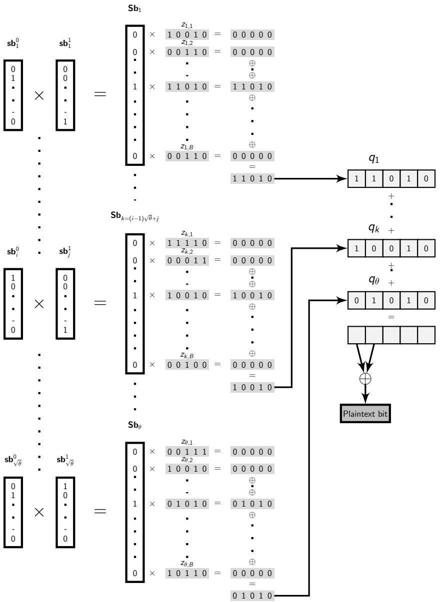
- Alternative decryption equation:

$$m \leftarrow [c]_2 \oplus \left[\left\lfloor \sum_{i=1}^{\Theta} s_i \cdot z_i \right\rceil \right]_2$$

where $z_i = y_i \cdot c$ for public y_i's

- Since s_i is sparse with $H(s_i) = \theta$, only $n = \lceil \log_2(\theta + 1) \rceil$ bits of precision for $z_i = y_i \cdot c$ is required
    - With $\theta = 15$, only $n = 4$ bits of precision for $z_i = y_i \cdot c$
- The decryption function can then be expressed as a polynomial of low degree (30) in the s_i's.

The decryption circuit

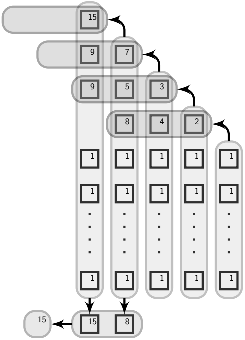


Grade School addition

- The decryption equation is now:

$$m \leftarrow c^* - \left\lfloor \sum_{k=1}^{\theta} q_k \right\rceil \mod 2$$

  - where the q_k's are rational in $[0, 2)$ with n bits of precision after the binary point.



Gentry's Bootstrapping

- The decryption circuit
  - Can now be expressed as a polynomial of small degree d in the secret-key bits s_i, given the $z_i = c \cdot y_i$.

$$m = C_{z_i}(s_1, \ldots, s_\Theta)$$

- To refresh a ciphertext:
  - Publish an encryption of the secret-key bits $\sigma_i = E_{pk}(s_i)$
  - Homomorphically evaluate $m = C_{z_i}(s_1, \ldots, s_\Theta)$, using the encryptions $\sigma_i = E_{pk}(s_i)$
  - We get $E_{pk}(m)$, that is a new ciphertext but possibly with less noise (a "recryption").
  - The new noise has size $\simeq d \cdot \rho$ and is independent of the initial noise.

PK size and timings

Instance	λ	ρ	η	γ	pk size	Recrypt
Toy	42	27	1026	$150 \cdot 10^3$	77 KB	0.41 s
Small	52	41	1558	$830 \cdot 10^3$	437 KB	4.5 s
Medium	62	56	2128	$4.2 \cdot 10^6$	2.2 MB	51 s
Large	72	71	2698	$19 \cdot 10^6$	10.3 MB	11 min

Conclusion

- Fully homomorphic encryption is a very active research area.
- Main challenge: make FHE pratical !
- Recent developments
  - FHE without bootstrapping (modulus switching) [BGV11]
  - Batch FHE [GHS12]
  - Implementation with homomorphic evaluation of AES [GHS12]
  - FHE based on matrix addition and multiplication [GSW13]
  - HElib: FHE library of Halevi and Shoup [HS14]
  - Faster Bootstrapping [AP13,AP14,DM15]