Fully Homomorphic Encryption Implementation Progresses and Challenges

Chiffrement homomorphe : une révolution en marche

Caroline Fontaine
CNRS, LSV
caroline.fontaine@lsv.fr
http://www.lsv.fr/~fontaine/

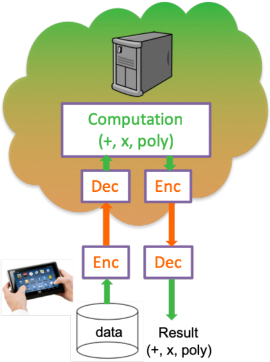
Masterclass, FIC 2019, Lille

Outline

Outline

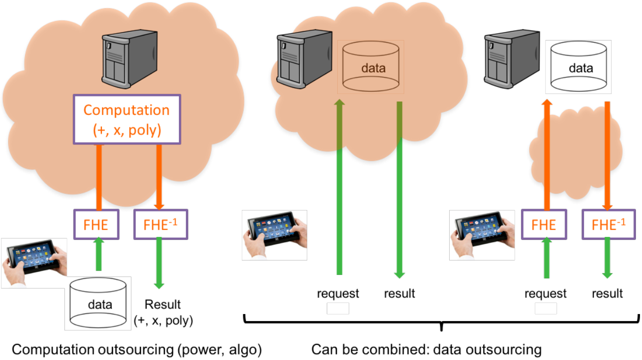
1. Context and Introduction
2. Applications and Practical Issues
   - Security
   - How to express high-level algorithms ?
   - Huge expansion of ciphertexts
   - Complexity
3. Conclusion

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



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

Homomorphic Encryption : we are dreaming of . . .

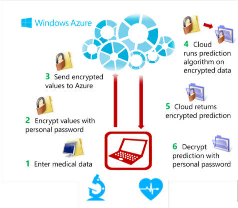


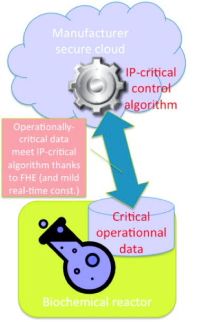
A revolution : data and/or services outsourcing without losing confidentiality !
Impact : citizens, administrations, companies, military, . . .
Domains : health care, power plants, multimedia content delivery, . . .
Computations : comparing, sorting/filtering, clustering, compressing, . . .

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

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

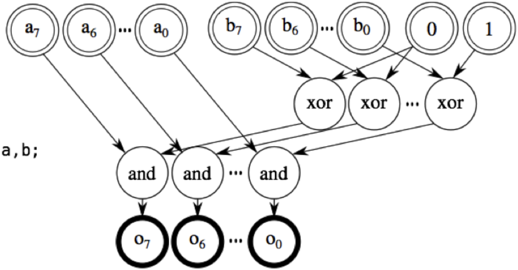
- Targeted **advertising**
- **Access Control** with respect to user profile
- **Biometric** authentication
- **Medical Diagnosis**
- **Critical engine** (reactor) **control**
- **Machine Learning** (deep learning)





Program's output = Circuit Eval = Polynomial Eval

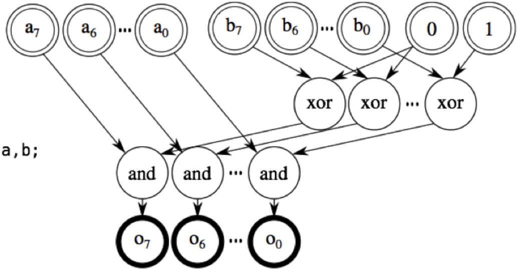
```
#include <iostream>
#include <stdint.h>
#include "integer.h"
void f
   (std::istream &i,
    std::ostream &o)
{
  SlicedInteger<int8_t> a,b;
  i >> a >> b;
  b = b ^ 0x01;
  a &= b;
  o << a;
}
```



$$F_i(x) = x_i x_{i+8} \qquad i = 1.\dots,7$$
$$F_0(x) = x_8(x_{16}+1)$$

Program's output = Circuit Eval = Polynomial Eval

```
#include <iostream>
#include <stdint.h>
#include "integer.h"
void f
   (std::istream &i,
    std::ostream &o)
{
  SlicedInteger<int8_t> a,b;
  i >> a >> b;
  b = b ^ 0x01;
  a &= b;
  o << a;
}
```



$$F_i(x) = x_i x_{i+8} \qquad i = 1. \dots, 7$$
$$F_0(x) = x_8(x_{16} + 1)$$

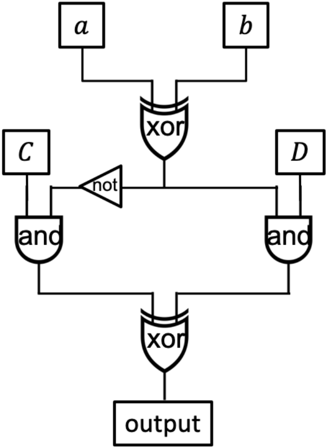
Then needing *Enc*() and *Dec*() satisfying

$$Dec\left(F\left(Enc(x_1), \dots, Enc(x_n)\right)\right) = F\left(x_1, \dots, x_n\right)$$

Focusing on the circuit

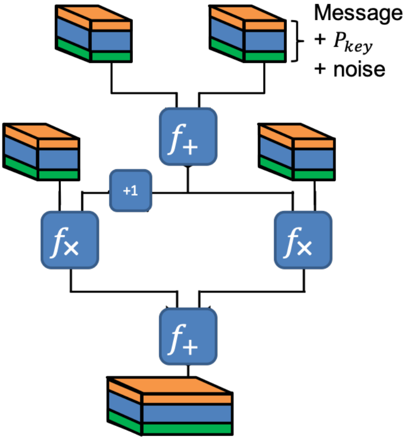
F **computed on plaintexts**

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

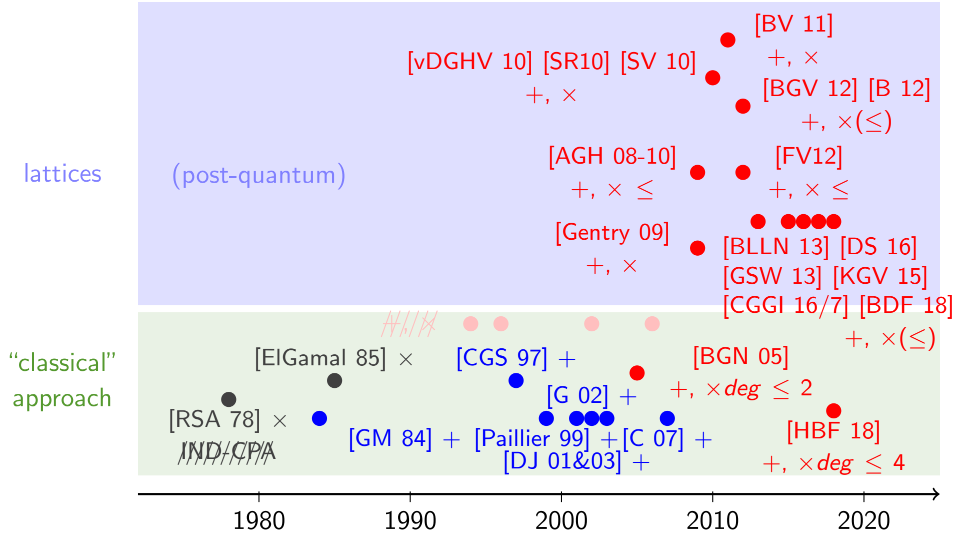


F **computed on ciphertexts**

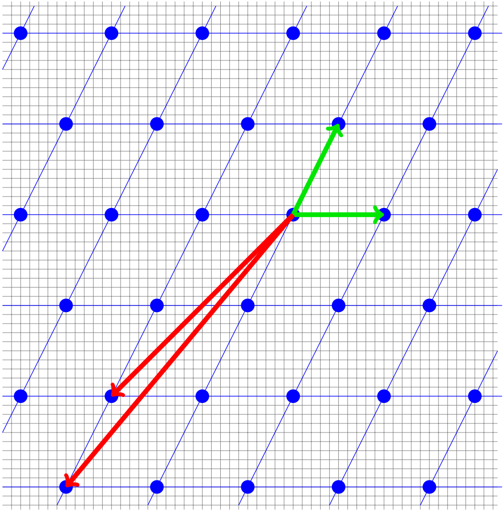
$$F\left(Enc(x_1), \ldots, Enc(x_n)\right)$$



It has been a long quest to handle polynomials



Lattice based S/FHE in a nutshell . . .



Lattice based S/FHE in a nutshell ...

Ex : FHE over the integers [vDGHV 10]

- Secret key (symmetric version here) : s
- Encryption of $m \in \{0, 1\}$: α, β random $\qquad c = m + 2\alpha + \beta s$
- Decryption : $c \bmod s = m + 2\alpha$ $\qquad m = (c \bmod s) \bmod 2$

Lattice based S/FHE in a nutshell ...

Ex : FHE over the integers [vDGHV 10]

- Secret key (symmetric version here) : s
- Encryption of $m \in \{0, 1\}$: α, β random $\qquad c = m + 2\alpha + \beta s$
- Decryption : $c \bmod s = m + 2\alpha$ $\qquad m = (c \bmod s) \bmod 2$
- Homomorphic addition : $c + c' = m + m' + 2(\alpha + \alpha') + (\beta + \beta')s$

Lattice based S/FHE in a nutshell . . .

Ex : FHE over the integers [vDGHV 10]

- Secret key (symmetric version here) : s
- Encryption of $m \in \{0,1\}$: α, β random $\quad c = m + 2\alpha + \beta s$
- Decryption : $c \bmod s = m + 2\alpha \quad m = (c \bmod s) \bmod 2$
- Homomorphic addition : $c + c' = m + m' + 2(\alpha + \alpha') + (\beta + \beta')s$

Condition :

To ensure a coherent decryption, we need : $m + m' + 2(\alpha + \alpha') < s$

Lattice based S/FHE in a nutshell ...

Ex : FHE over the integers [vDGHV 10]

- Secret key (symmetric version here) : s
- Encryption of $m \in \{0,1\}$: α, β random $\qquad c = m + 2\alpha + \beta s$
- Decryption : $c \bmod s = m + 2\alpha$ $\qquad m = (c \bmod s) \bmod 2$
- Homomorphic addition : $c + c' = m + m' + 2(\alpha + \alpha') + (\beta + \beta')s$

Condition :

To ensure a coherent decryption, we need : $m + m' + 2(\alpha + \alpha') < s$

If $2\alpha < s/2$, $2\alpha' < s/2$, and if c and c' are fresh ciphertexts, then it is ok.

Lattice based S/FHE in a nutshell . . .

Ex : FHE over the integers [vDGHV 10]

- Secret key (symmetric version here) : s
- Encryption of $m \in \{0,1\}$: α, β random $\quad c = m + 2\alpha + \beta s$
- Decryption : $c \bmod s = m + 2\alpha$ $\quad m = (c \bmod s) \bmod 2$
- Homomorphic addition : $c + c' = m + m' + 2(\alpha + \alpha') + (\beta + \beta')s$

Condition :

To ensure a coherent decryption, we need : $m + m' + 2(\alpha + \alpha') < s$

If $2\alpha < s/2$, $2\alpha' < s/2$, and if c and c' are fresh ciphertexts, then it is ok.

If c_i is not a fresh ciphertext, we might not be able to decrypt it properly (too much noise) !

Lattice based S/FHE in a nutshell . . .

Ex : FHE over the integers [vDGHV 10]

- Secret key (symmetric version here) : s
- Encryption of $m \in \{0, 1\}$: α, β random $\qquad c = m + 2\alpha + \beta s$
- Decryption : $c \bmod s = m + 2\alpha$ $\qquad m = (c \bmod s) \bmod 2$
- Homomorphic addition : $c + c' = m + m' + 2(\alpha + \alpha') + (\beta + \beta')s$

Condition :

To ensure a coherent decryption, we need : $m + m' + 2(\alpha + \alpha') < s$

If $2\alpha < s/2$, $2\alpha' < s/2$, and if c and c' are fresh ciphertexts, then it is ok.

If c_i is not a fresh ciphertext, we might not be able to decrypt it properly (too much noise) !

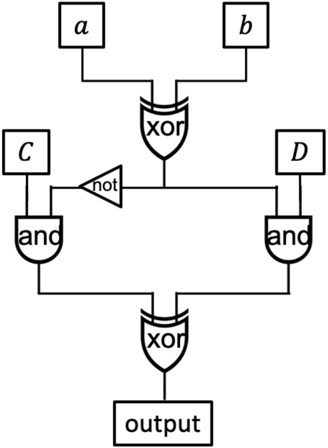
And it is even worse in the case of homomorphic multiplication !

The challenge is to keep control of this noise during computation.

Noise grows gate after gate...

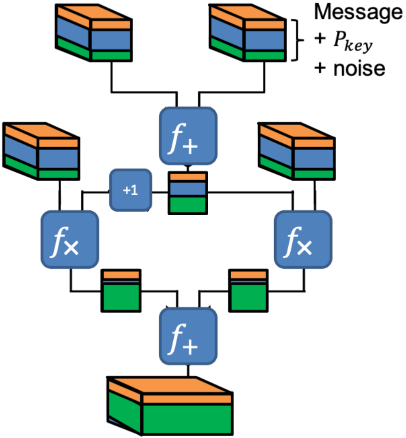
F computed on plaintexts

$F(x_1, \ldots, x_n)$



F computed on ciphertexts

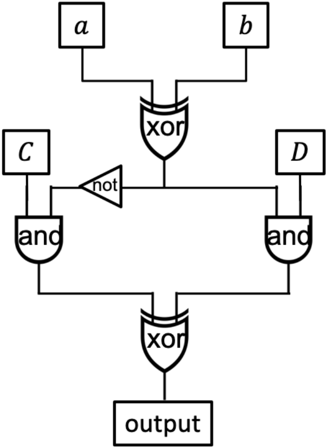
$F\left(Enc(x_1), \ldots, Enc(x_n)\right)$



If noise grows too much. . .

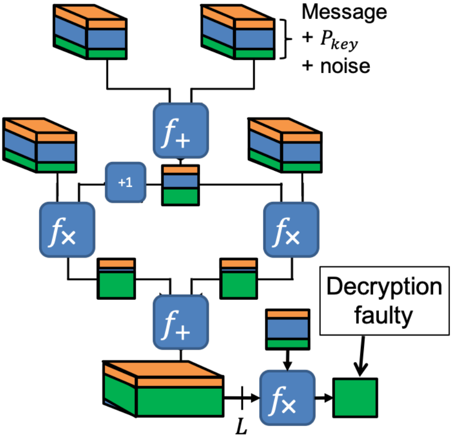
F computed on plaintexts

$F(x_1, \ldots, x_n)$

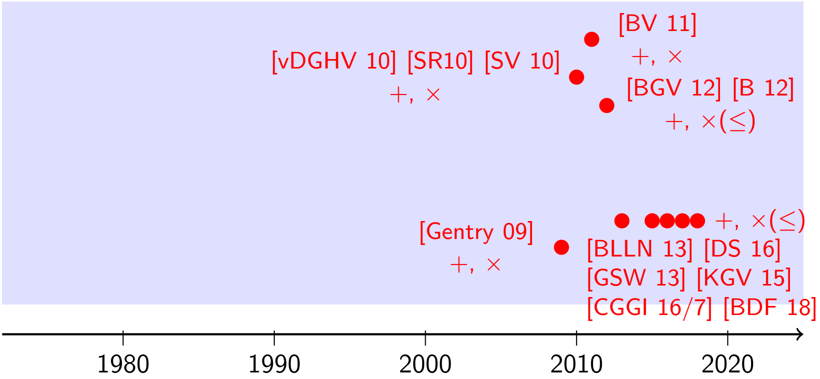


F computed on ciphertexts

$F\left(Enc(x_1), \ldots, Enc(x_n)\right)$

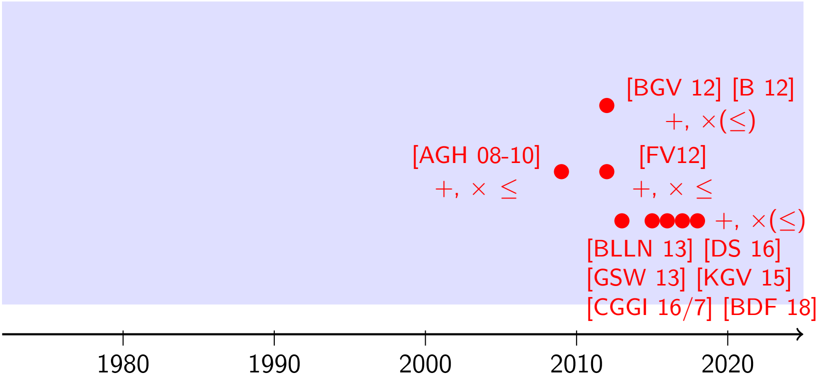


How to handle this noise ? (1/2)



- FHE : × unbounded → using bootstrapping
  - once the setting is fixed, "any" circuit can be evaluated
  - 2009-2014 : too complex to be used in practice
  - BUT recent improvements, *e.g.* [PV15] to opt. bootstrapping use, [CGGI16/7] to accelerate it...

How to handle this noise ? (2/2)



- Leveled FHE schemes : $\times$ bounded $\rightarrow$ without bootstrapping
  - a limited (but often sufficient) number of multiplications
  - maximum mult. depth is related to the setting (cannot be modified afterwards)
  - a lower complexity

How to handle this noise ? (genealogy)

Central problem : Noisy encryption $\longrightarrow$ noise management

Several noise management strategies $\Rightarrow$ 4 generations of ciphers :

1. 2009 [Gen 09] exponential in the circuit size
2. 2010-2012 (DGHV, BGV, FV, ~~YASHE~~ [BLLN 13]), polynomial
3. 2013-2016 (GSW, SHIELD [KGV 15] , ~~F-NTRU~~ [DS 16]), linear
4. since 2014 (FHEW, TFHE [CGGI 16] , HE6 [BDF 18]), constant

Remark : settings depend on target security level and circuit size

Outline

1. Context and Introduction

2. Applications and Practical Issues
   - Security
   - How to express high-level algorithms ?
   - Huge expansion of ciphertexts
   - Complexity

3. Conclusion

Want to play ? (1/2)

2011 : open-source implementation of [SV10] by [PBS11] http://www.hcrypt.com

2012 : private implem. of [BGV12] dedicated to AES homo. eval. [GHS12]

2013-* : platform at CEA [AFFGS13,FSFAG13] , home-made implem. of [BGV12] (vect and poly) and [FV12] + HElib and more recent open-source libraries + compilation chain ⇒ private platform at beginning, open-source since Jan 2018 ! https://github.com/CEA-LIST/Cingulata

2013 : open-source implementation of [vDGHV10] with the improvements from [CNT12] : https://github.com/coron/fhe

2013 : private implementation in [CLT 13] dedicated to AES homomorphic evaluation using an improved version of [vDGHV10]

2013 : private implementation of [BLLN 13], with good performances with 2 or 3 multiplicative depth

2013-* : open-source implem. of [SV10] and [BGV12] called HElib by Halevi et al. http://shaih.github.io/HElib/

Want to play ? (2/2)

2014 : open-source implem. of [FV12] and [BLLN13] YASHE, compared in [LN14] https://github.com/tlepoint/homomorphic-simon

2015 : open-source library called SEAL1.0, based on YASHE' http://sealcrypto.codeplex.com/

2016-* : SEAL1.0 is replaced by SEAL2.1, and now SEAL3.1.0 based on another implementation of [FV12] http://sealcrypto.org/

2016 : open-source library to efficiently handle polynomials, called NFLlib https://github.com/quarkslab/NFLlib

2016 : open-source implementation of [FV12] based on NFLlib https://github.com/CryptoExperts/FV-NFLlib

2016 open-source multi-precision moduli library, called HElib-MP https://github.com/tricosset/HElib-MP, based on HElib

2016 : private implem. of FV with RNS [BEHZ16]

2017-* : open-source implem. of TFHE [CGGI16] https://github.com/tfhe/tfhe

2018 : open-source implem. of HE6 [BDF18] https://github.com/gbonnoron/Borogrove

Outline

Which kind of security ?

Semantic Security

Semantic security is necessary !
(and as S/FHE schemes are malleable, IND-CCA2 can never be achievable).

$\Rightarrow$ probabilistic encryption

Which kind of security ?

Semantic Security

Semantic security is necessary !
(and as S/FHE schemes are malleable, IND-CCA2 can never be achievable).

$\Rightarrow$ probabilistic encryption
$\Rightarrow$ expansion (ciphertexts are longer than plaintexts)
and parameters setting has a huge impact on expansion !

Which kind of security ?

Semantic Security

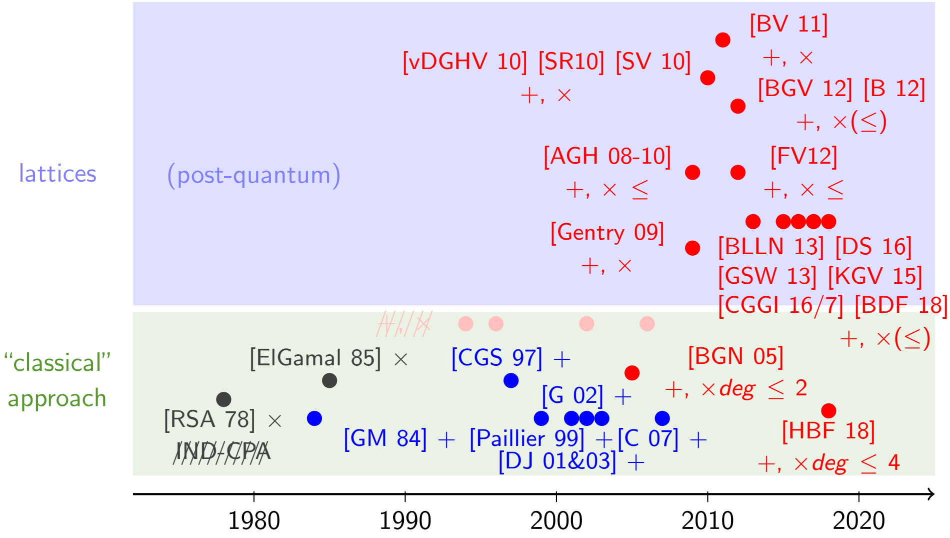
Semantic security is necessary !
(and as S/FHE schemes are malleable, IND-CCA2 can never be achievable).

$\Rightarrow$ probabilistic encryption
$\Rightarrow$ expansion (ciphertexts are longer than plaintexts)
and parameters setting has a huge impact on expansion !

e.g. for 128-bits security level, expansion is (without batching) :

- equal to 2 with Paillier cryptosystem (only $+$)
- around $5,000$ with elliptic curve based solution BGN-F-CF [HBF18] $(+, \times deg \leq 4)$
- between $50,000$ and $1,000,000$ for lattice-based S/FHE ! $(+, \times(\leq))$

It has been a long quest to handle polynomials



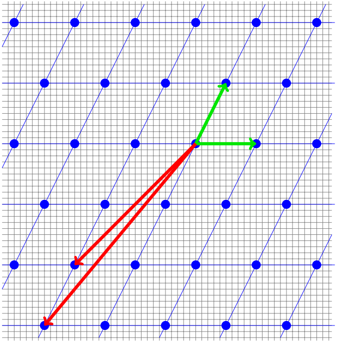
Which security level ?

Security Analysis of elliptic curve based schemes

Computational Security (w.r.t. DLP). Well understood and studied.

Security Analysis of lattice based schemes

Computational Security (w.r.t. hard problems as LWE, R-LWE,. . .)
Theoretical studies essentially focus on asymptotic and generic estimations (may be not so close to real S/FHE situations).
Some experiments (based on LLL, BKZ,. . .) provide estimations (but may remain too optimistic today).



See *e.g.* [Alb15,ABD16][Peik16][KF17][BF17][Alb17][AN17] .
⇒ Due to some of them, YASHE AND F-NTRU are down !

Which security level for lattice based S/FHE ?

See the (online) estimator provided by Martin Albrecht (always evolving) :
`https://bitbucket.org/malb/lwe-estimator`

SageMathCell

Type some Sage code below and press Evaluate.

```
load("https://bitbucket.org/malb/lwe-estimator/raw/HEAD/estimator.py")
n, alpha, q = 256, 0.000976562500000000, 65537
set_verbose(1)
_ = estimate_lwe(n, alpha, q)
```

Evaluate   Language: Sage

Share

```
  mitm   bop:  ≈2^917.3,  oracle:        370,    mem:  ≈2^904.3,    rop:  ≈2^913.3
   bkw   bop:  ≈2^148.7,  oracle:  ≈2^131.5,      m:  ≈2^124.7,    mem:  ≈2^132.5,    rop:  ≈2^144.7,      b:         8,    t1:          8,     t2:         15,     l:
   sis sieve:  ≈2^142.9,  oracle:   ≈2^28.6,  delta_0: 1.0045951,   bkz2:  ≈2^224.3,   beta:        325,     lp:  ≈2^212.2,  quantum_sieve:  ≈2^134.1,    |v|: 1360.4505
   dec   rop:  ≈2^122.8,  oracle:   ≈2^18.8,  sieve:  ≈2^121.7,  delta_0: 1.0050017,   bkz2:  ≈2^183.4,   beta:        286,     lp:  ≈2^180.4,  quantum_sieve:  ≈2^113.9
kannan sieve:  ≈2^114.4,  oracle:   ≈2^15.0,  delta_0: 1.0051468,   bkz2:  ≈2^170.4,   beta:        274,     lp:  ≈2^169.6,  quantum_sieve:  ≈2^107.0,      m:        744
```

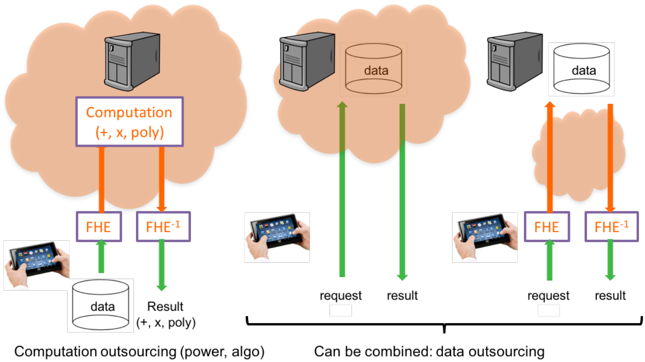
Help | Powered by SageMath

⇒ it is really hard today to know how to choose the right parameters to ensure a given security level (*e.g.* 128) and we really need more targeted attacks and studies to derive precise guidelines for the choice of parameters (see [MBF18] for an attempt, based on the current state-of-the-art).

Outline

1. Context and Introduction
2. Applications and Practical Issues
   - Security
   - How to express high-level algorithms ?
   - Huge expansion of ciphertexts
   - Complexity
3. Conclusion

Applications : we are dreaming of . . .



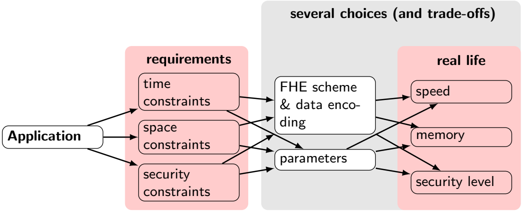
A revolution : data and/or services outsourcing without losing confidentiality !
Impact : citizens, administrations, companies, military, . . .
Domains : health care, power plants, multimedia content delivery, . . .
Computations : comparing, sorting/filtering, clustering, compressing, . . .

How to help programers ?

Our goal

To help programers (not crypto specialists !) to use S/FHE in the development of their software/hardware stuff [AFF+13][FAR+13][CS14]...

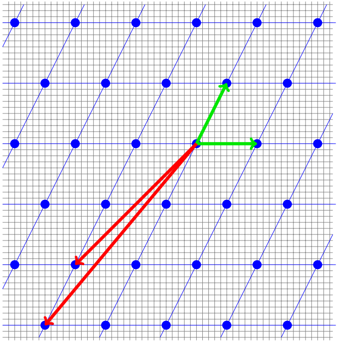
1. Cryptographers are necessary to help choosing the most appropriate S/FHE scheme & data encoding & parameters :



2. This being done, programers must be able to go further alone, without interacting with cryptographers !

With cryptographers : choosing data encoding (1/3)

Your (sliced) data (bits, integers/floats) ⟷



Each piece of (sliced) data has to be related with one plaintxt (a point of the lattice, *i.e.* integers or polynomials)

With cryptographers : choosing data encoding (2/3)

Your data : managing bits or integers/floats ? (slicing)

Processing integers/floats may seem more interesting at a first glance, BUT in some cases using integers/floats will reduce the set of algorithms one can execute in the encrypted domain, *e.g.* `if-then-else` implies a management at the bit-level for Generations 1-2-3.

	bit-level		**integer/float-level**	
Operations	**?**	$\times$ **depth**	**?**	$\times$ **depth**
addition	yes	$n-1$	yes	0
multiplication	yes	$n-1$	yes	1
scalar division	yes	dep. on scalar	yes	1
scalar multiplication	yes	dep. on scalar	yes	0
shift	yes	0	yes	1
comparison	yes	$\log_2 n$	no	-
cond. assignment	yes	$\log_2 n$	no	-

With cryptographers : choosing data encoding (3/3)

In case we choose an encoding at the bit-level, we need to redefine integers/floats encoding to get operators on integers/floats (based on those on bits, with 2's complement, sign bit, . . .), for :

`addition multiplication substraction ≪ ≫`

Batching (packing several plaintexts into one)

To process several bits (resp. integers/floats) at the same time, *e.g.* using Chinese Remaining Theorem.

Programers are not obliged to implem. S/FHE

From Armadillo platform [AFF+13][FAR+13][CS14] , now Cingulata :

Definition of C++ classes `ClearBit` and `CryptoBit` written with the help of cryptographers (link with data encoding and S/FHE scheme) :

```
class C++ template<typename bit, int size>
```

Any programer can then use them :

Example

Applying a bubble sort on data in clear :

```
bsort<Integer<ClearBit,8> >(arr,n);
```

Applying the **same** bubble sort on encrypted data :

```
bsort<Integer<CryptoBit,8> >(arr,n);
```

Software Compilation Process and Optimization

initial algorithm

↓ *code modification by the programer*

equivalent algorithm in C++ using `ClearBit`/`CryptoBit` templates

↓ *data slicing*

equivalent Boolean circuit

↓ *optimization module*

optimized Boolean circuit (especially with decreased multiplicative depth)

↓

C++ code for sequential or parallel execution

Choosing the right algorithm

It is important to choose the algorithm with the best worst-case complexity (not usual!) if tests have to been performed over the encrypted data.

Software Compilation Process and Optimization

initial algorithm

↓ *code modification by the programer*

equivalent algorithm in C++ using `ClearBit`/`CryptoBit` templates

↓ *data slicing*

equivalent Boolean circuit

↓ *optimization module*

optimized Boolean circuit (especially with decreased multiplicative depth)

↓

C++ code for sequential or parallel execution

C++ classes `ClearBit` and `CryptoBit`

Applying a bubble sort on data in clear :

```
bsort<Integer<ClearBit,8> >(arr,n);
```

Applying the **same** bubble sort on encr. data :

```
bsort<Integer<CryptoBit,8> >(arr,n);
```

Software Compilation Process and Optimization

initial algorithm

↓ *code modification by the programer*

equivalent algorithm in C++ using `ClearBit`/`CryptoBit` templates

↓ *data slicing*

equivalent Boolean circuit

↓ *optimization*

optimized Boolean circuit (especially with decreased multiplicative depth)

↓

C++ code for sequential or parallel execution

Going down at the Boolean level

Data slicing & conversion Pgm → Boolean circuit.

Use XOR and AND for ClearBit
Use HE-ADD and HE-MULT for CryptoBit

Program → Boolean circuit

Comparisons of Encrypted Data

How to perform tests and express `if-then-else` ?

Boolean bitwise operators : $\begin{cases} \texttt{a < b} : \text{MSB of } \texttt{a+(-b)} \\ \texttt{a > b} : \text{MSB of } \texttt{b+(-a)} \\ \texttt{a = b} : \texttt{(a < b) NOR (a > b)} \end{cases}$

"`if c then x = a else x = b`" can be achieved through the following operator : $x = \texttt{select}(c,a,b) = \begin{cases} a & \text{if } c=1 \\ b & \text{otherwise} \end{cases}$

$x =$ `select(c,a,b) = (c AND a) XOR ((NOT c) AND b)`

- no data leakage ;-)
- BUT bit-level encoding + worst-case complexity as we have to evaluate the whole circuit (all the branches of the circuit)

Bubble sort : a meaningful example

Classical bubble sort :

```
void bsort(int *arr,int n)
{
  for(int i=0;i<n-1;i++)
  {
    for(int j=1;j<n-i;j++)
      if(arr[j-1]>arr[j])
      {
        int t=arr[j-1];
        arr[j-1]=arr[j];
        arr[j]=t;
      }
  }
}
```

Rewritten bubble sort :

```
void bsort(int *arr,int n)
{
  for(int i=0;i<n-1;i++)
  {
    for(int j=1;j<n-i;j++)
      {
        int gt=arr[j-1]>arr[j];
        int t=gt*arr[j-1]^(!gt*arr[j]);
        arr[j-1]=gt*arr[j]^(!gt*arr[j-1]);
        arr[j]=t;
      }
  }
}
```

Software Compilation Process and Optimization

initial algorithm

↓ *code modification by the programer*

equivalent algorithm in C++ using `ClearBit`/`CryptoBit` templates

↓ *data slicing*

equivalent Boolean circuit

↓ *optimization module*

optimized Boolean circuit (especially with decreased multiplicative depth)

↓

C++ code for sequential or parallel execution

Optimization (acc. to Generations 1-2-3-4)

Minimization of the multiplicative length (also taking care of the width of the circuit and the total number of multiplications and additions).

Optimizing the Boolean circuit

Characterization of # add, # mul, × depth

Estimation and optimization possible with the help of `ClearBit`.

Some values for classical algorithms (before optimization) :

	$\sum_{i=1}^{10} t[i]$ (4 bits)	threshold (4 bits)	$b^2 - 4ac$ (4 bits)	bubble sort (10x4 bits)	FFT (256x32 bits)
# add	99	390	126	2372	7291592
# mul	27	60	32	238	5296128
× depth	4	5	7	69	166
	(16 bits)		(16 bits)	(10x8 bits)	
# add	423		1188	3240	
# mul	279		1126	2790	
× depth	16		32	136	

⇒ `ClearBit` class helps to debug the implementation and to optimize it !

Circuit optimization (included in Cingulata)

Multiplicative depth should be kept < 30

- Main goal : to reduce multiplicative depth (the most critical)
- Secondary goal : to reduce the number of multiplicative gates

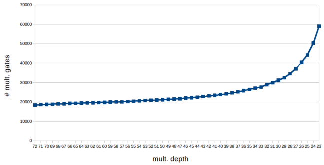
By iteratively applying local circuit rewriting operators.

E.g. Medical diagnosis :

- reducing multiplicative depth from around 20 to 8 !
- then we can add transcryption (Kreyvium adds $\times$ depth of 12

E.g. Running Length Encoding :

- reducing multiplicative depth from 70 to 20 ! Here at a cost in terms of total multiplicative gates.



Software Compilation Process and Optimization

initial algorithm

↓ *code modification by the programer*

equivalent algorithm in C++ using ClearBit/CryptoBit templates

↓ *data slicing*

equivalent Boolean circuit

↓ *optimization module*

optimized Boolean circuit (especially with decreased multiplicative depth)

↓

C++ code for sequential or parallel execution

Outline

An awful expansion factor !

Expansion (without batching)

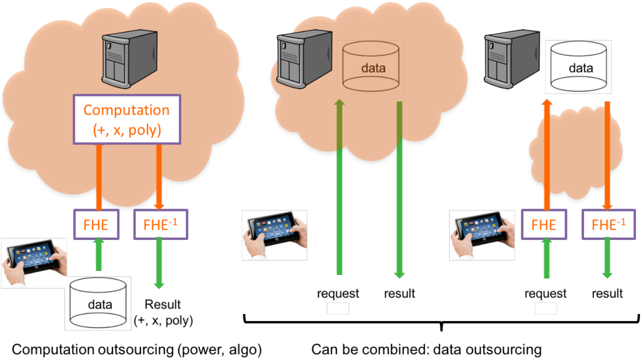
Current estimations of security parameters lead to an expansion factor

- equal to 2 with Paillier cryptosystem (only $+$)
- around $5,000$ with elliptic curve based solution BGN-F-CF [HBF18] $(+, \times deg \leq 4)$
- between $50,000$ and $1,000,000$ for lattice-based S/FHE ! $(+, \times(\leq))$

$\Rightarrow$ pb to store and process, and to transmit data encrypted with S/FHE !

1. it would be very nice to design new schemes with a lower expansion,
2. we can help by choosing a good data representation and pack several plaintexts together (batching : CRT, SIMD, RNS),
3. we also have to do our best to manage huge ciphertexts, *e.g.* properly combining classical symmetric encryption with S/FHE.

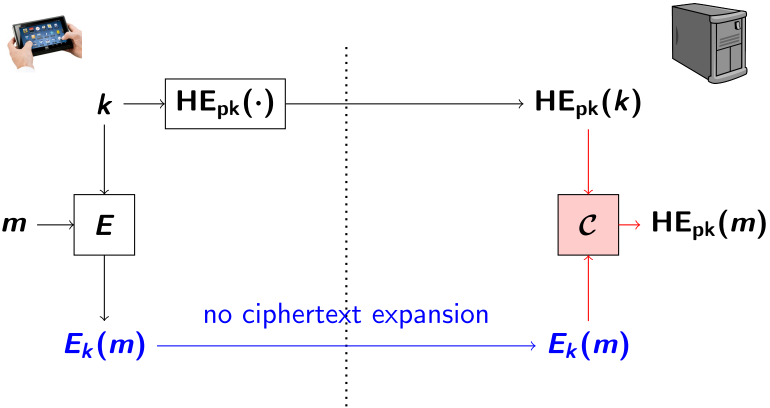
Applications : we are dreaming of . . .



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, . . .

How to efficiently upload S/FHE ciphertext ?

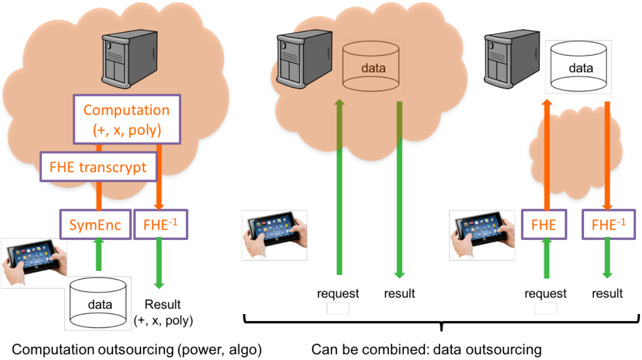
[NLV11]



$$\mathcal{C} = \mathsf{HE.Eval}_{E^{-1}}$$

What kind of symmetric encryption is the most appropriate ?

This leads to. . .



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, . . .

HE-friendly ciphers ? (1/2)

Main goal

To minimize the multiplicative depth of the decryption function.

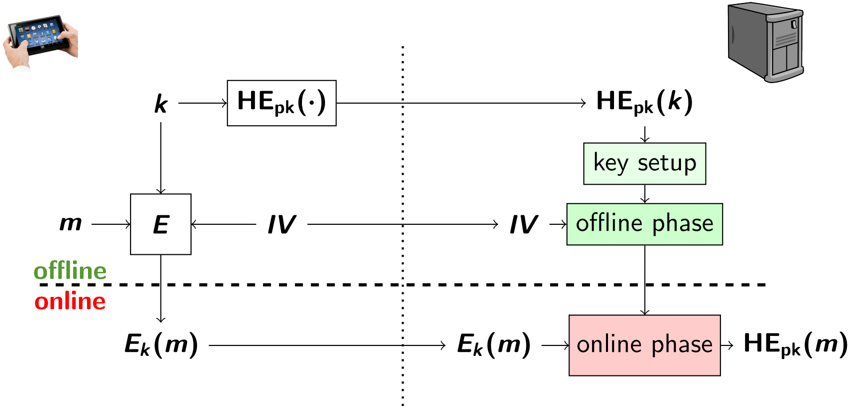
First concrete proposals have been block ciphers

- Already existing block ciphers :
  - Optimized implementations of AES [GHS12][CCKL+13][DHS14]
    $\rightarrow$ but AES's $\times$ depth remains too large ($\rightarrow$ too slow)
  - Lightweight block ciphers : SIMON [LN14] , PRINCE [DSES14]
    $\rightarrow$ SIMON behaves better than AES
    $\rightarrow$ PRINCE behaves better than SIMON, but remains too slow
- Dedicated block cipher : Low-MC-80 and Low-MC-128 [ARSTZ15]
  $\rightarrow$ but subject to some interpolation attacks (sparse ANF)
  $\Rightarrow$ a tweaked version has been presented at FSE 2016's rump session (more rounds), but security remains not clear (≤ 118)

Ciphertext decompression with IV-based encryption

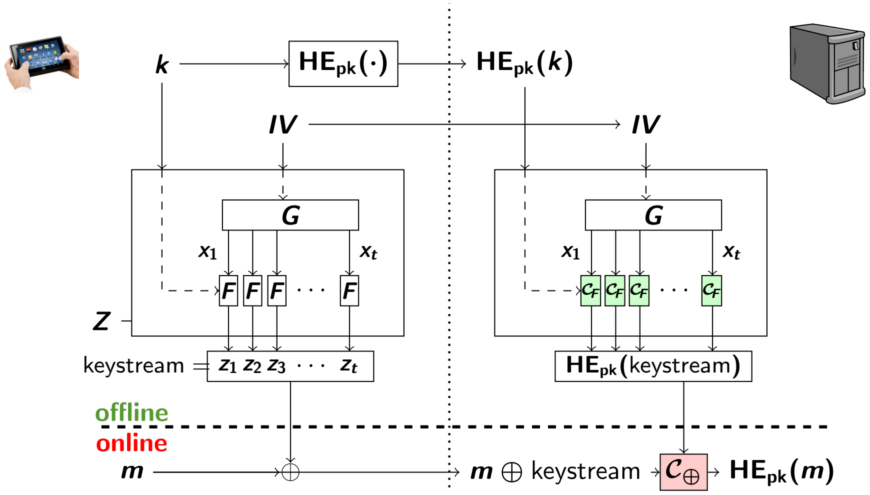
A new approach [CCCF+16,CCCF+18]

to reduce the online phase to a minimum . . .



Ciphertext decompression with IV-based encryption

. . . with an additive stream cipher ;-)



HE-friendly ciphers ? (2/2)

Using a stream cipher reduces on-line phase to the minimum.
Current candidates for function *F* are :

[CCCF+16,CCCF+18] :

- Trivium : coming from eSTREAM (2008), firmly established security, 80 bits security
- Kreyvium : based on Trivium, same security confidence, 128 bits security

[MJSC 16] :

- Flip : lower complexity, but security should be more deeply analyzed [DLR 16]

According to today's state-of-the-art, Kreyvium seems to be the best available solution (but may be replaced by Flip if new security analysis is good).

Outline

1. Context and Introduction
2. Applications and Practical Issues
   - Security
   - How to express high-level algorithms ?
   - Huge expansion of ciphertexts
   - Complexity
3. Conclusion

Complexity issues

Complexity

High computation complexity related to the noise management.

Cryptographic issues :

⇒ it should be nice to have less complex S/FHE schemes, even if a huge effort has still been done and complexity already decreased a lot, and to optimize the use of bootstrapping, modulus switching, re-linearization, etc (*e.g.* see [PV15] for bootstrapping opt. and the hope arising from [CGGI16/7][BDF18]).

Application related issues :

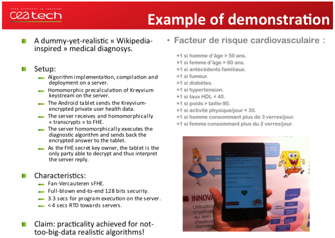
⇒ for a given target, we need to carefully choose the right algorithm (with the best worst-case complexity !)

⇒ we need to optimize the implementation (circuit optimization, bits/integers & batching, software/hardware implementation).

Examples of FHE Practical Achievements

. . . order of magnitude (dep. on security level, batching, optimization) . . .

- Energy-consumption profile classification : < 1 second
- Various medical diagnosis : 3 seconds $<$ 2 minutes
- Genome-based diagnosis : $<$ 10 minutes
- Running Length Encoding (step for image/video compression) : $\simeq 30$ minutes with 48 cores
- . . .



Outline

Conclusion 1/2

Nice

Very nice applications + post-quantum encryption :-)
A lot of efforts and progresses (everything is moving really fast).
Quite a lot of implementations available now.

Making small applications affordable ! We are on the right way :-)

Conclusion 2[/2

BUT still a lot of (theoretical and practical) work to be done :

- security (to be better understood)
- expansion (to be better decreased and managed)
- complexity (new schemes, worst-case complexity, bootstrapping optimization, etc)
- implementation optimization (Boolean circuit, software & hardware)
- help programers to choose the right scheme with an adapted setting /* comparison is not easy at all ! */

And beyond. . .

Functional Encryption

Similar but different paradigm to compute over encrypted data while giving access in clear to some computation results, according to different public keys.

Obfuscation

Strong links between FHE and indistinguishable Obfuscation.

Software certification in FHE context

Not yet efficiently addressed (only starting), but important in real applications if we want to trust computation !

Questions ?

Thanks to all co-authors and collaborators (academic & industry)

French activities :

- design (S/FHE + friendly symmetric)
- security analysis
- batching
- compilation : software, hardware
- benchmarking and parameters setting

Selected personal contributions

[CCCF+18] Stream Ciphers : A Practical Solution for Efficient Homomorphic-Ciphertext Compression, A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, R. Sirdey. Journal of Cryptology, Volume 31, Issue 3, pp 885-916, July 2018.
[MBF18] Practical Parameters for Somewhat Homomorphic Encryption Schemes on Binary Circuits, V. Migliore, G. Bonnoron, C. Fontaine. IEEE Transactions on Computers, Special Section on Cryptographic Engineering in a Post-Quantum World, 2018.
[HBF18] Design and Implementation of Low Depth Pairing-based Homomorphic Encryption Scheme, V. Herbert, B. Biswas, C. Fontaine. Journal of Cryptographic Engineering, 2018.
[MMRL+18] Hardware/Software co-Design of an Accelerator for FV Homomorphic Encryption Scheme using Karatsuba Algorithm, V. Migliore, M. Méndez Real, V. Lapôtre, A. Tisserand, C. Fontaine, G. Gogniat. IEEE Transactions on Computers 67(3) :335-347, IEEE (2018).
[AFF+13] Recent advances in homomorphic encryption : a possible future for signal processing in the encrypted domain, C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, R. Sirdey. IEEE Signal Processing Magazine, Number 2, Volume 30, pp. 108-117 (2013), special issue "Signal Processing in the Encrypted Domain : When Cryptography Meets Signal Processing".