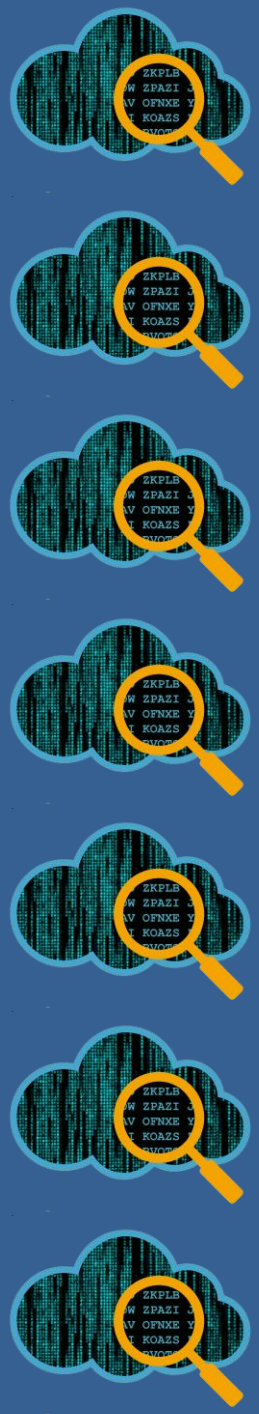


Towards practical homomorphic cryptocomputing

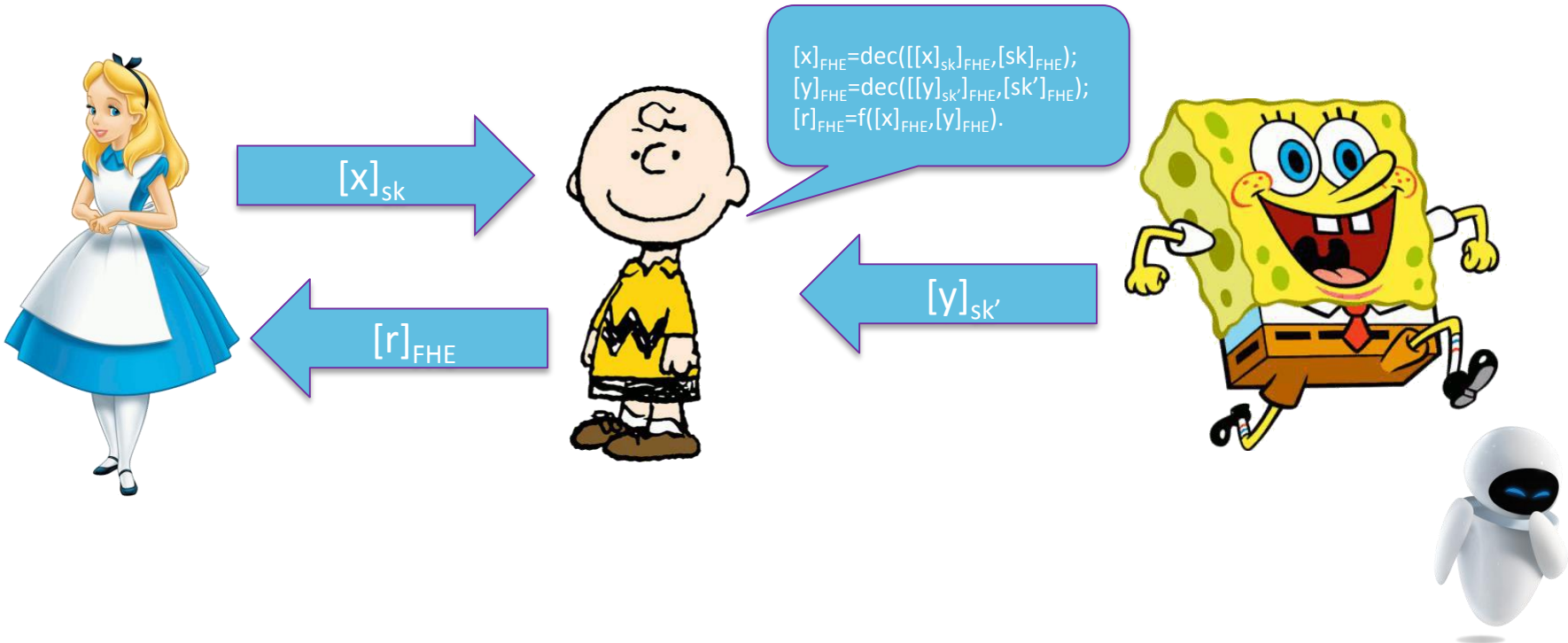
SEC2 2016
Lorient

Renaud Sirdey
CEA LIST
(work in collaboration with other people cited at the end)

July 2016

The dream

- Can Charlie do something useful for Alice using both Alice and Bob data but without revealing them (the data) to him (Charlie) ?



FHE in a nutshell

- On top of allowing to encrypt and decrypt data, an FHE scheme allows to perform (any) **calculations in the encrypted domain**.
  - Without access to either intermediate or final calculations results by the computer.

- Although the first generation of systems (2009) were too costly, practicality is now (2015-16) being achieved for **a first round of applications**.

Cryptosystem API:
- $enc_{pk} : Z_2 \rightarrow \Omega$.
- $dec_{sk} : \Omega \rightarrow Z_2$.
- $add_{pk} : \Omega \times \Omega \rightarrow \Omega$.
- $mul_{pk} : \Omega \times \Omega \rightarrow \Omega$.

where Ω is a large cardinality set e.g. $Z_q n$.

Key properties: for all $m_1 \in Z_2$ and all $m_2 \in Z_2$
- $dec_{sk}(add_{pk}(enc_{pk}(m_1),enc_{pk}(m_2))=m_1 \oplus m_2)$.
- $dec_{sk}(mul_{pk}(enc_{pk}(m_1),enc_{pk}(m_2))=m_1 \otimes m_2$.

(and these properties hold long enough…)

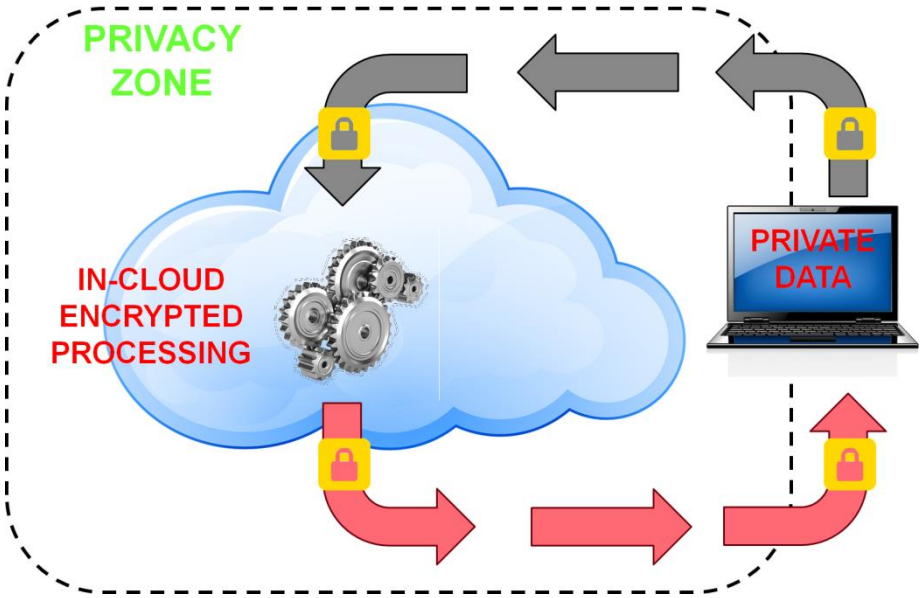
Further dreaming

- New settings where users can benefit from cloud services taking into account privacy-critical data, still without effectively giving them away.
  - Undisclosed cross-valorization of data (and algorithms).
  - Intrinsic data protection on vulnerable platforms.
  - Privacy-preserving outsourcing.
  - Etc.

- And, as an engineer, I lack imagination…

Trust model

- In the most basic settings, two parties are involved:
  - The user: owner of some private data.
  - The server: owner of an algorithm and possibly some data which it is willing to inject in the calculation.
  - However, the server has complete control over the algorithm.
  - So the user must trust that the server will perfom consistently with a functional specification – although it has no access to the algorithm details.



A simple cryptosystem

- Key:
  - An odd integer p randomly chosen in $[2^{\eta-1},2^{\eta}[$.
- Encryption of $m \in \{0,1\}$:
  - Randomly choose a large q and r ($2r<p/2$) and let $c := qp+2r+m$.
- Decryption:
  - $m := (c \bmod p) \bmod 2$.

- Semantically secure under the hardness assumption of the approximate GCD problem.

The problem of noise

- FHE schemes are necessarily probablistic.
  - I.e. some noise is added in the encryption process.
- All known FHE are intrinsically unstable.
  - The noise amplitude grows with the homomorphic calculations until decryption is no more possible!
    - Usually noise growth is faster with muls than with adds.
- Part of the intrinsic complexity of FHE schemes is due to noise managment.

Blueprint 1 : noise managment by means of self-reference

- Assume an asymmetric scheme (pk,sk).
- Let [x] be an encrypted value obtained after some homomorphic operation(s).
  - With an arbitrary noise level below the decryption threshold.
- Superencrypt [x] to get [[x]].
  - [[x]] is a noise-free encryption of a noisy encryption of x.
- Then homomorphically execute the decryption circuit: dec([[x]],[sk])=[x]'.
  - [x]' is an encryption of x with a constant noise.
  - This is called bootstrapping.

Blueprint 1 : noise managment by means of self-reference

- As long as a cryptosystem is homomorphic enough to evaluate one operation followed by its own decryption circuit it can compute forever (and such systems do exist…).

- Open questions:
  - Is it safe to encrypt sk with pk?
  - Is efficient boostrapping possible?

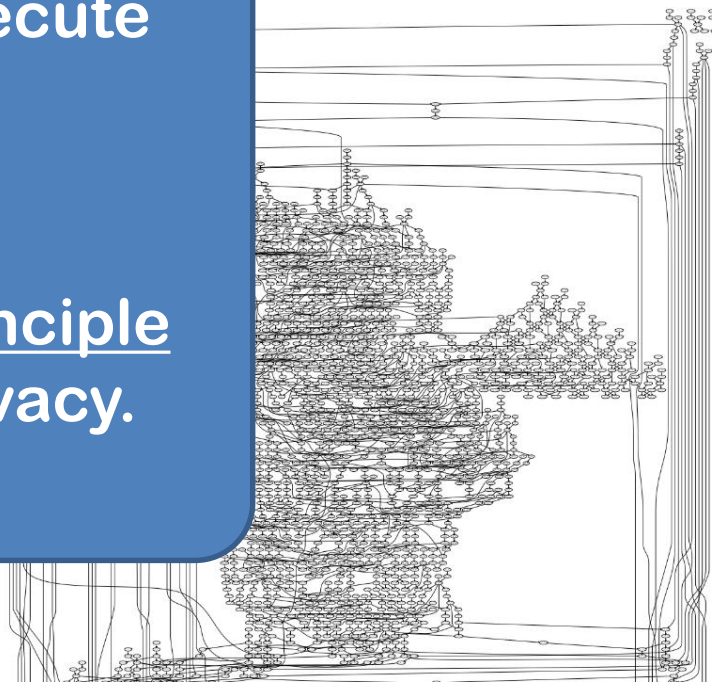
Blueprint 2: somewhat fully homomorphic encryption

- In practice we do not (yet) know how to practically achieve bootstrapping.
  - Still there is hope (e.g. FHEW and recent extensions).
- So the approach is to use cryptosystems which can be rendered homomorphic-enough to execute an a priori given (class of) algorithms.
  - This can automatically be done « at compile time » (more on that later).
- We now have several reasonably efficient such cryptosystems:
  - BGV (implemented in HELib), Fan-Vercauteren (my personal favorite), YASHE, GSW, and a few others.
  - Some of them with bitslicing-type parallelism (batching).

The quest for universality

- So we can
  circuits.
  - I.e. dire
    which v
    inputs,
    (XOR, A
- Boolean
  control st
  oblivious
- Oblivious
  are Turi
- Hence
  everything computable!

- And, b. t. w., it is also possible to homomorphically execute an encrypted Turing machine.
- Hence, we can <u>in principle</u> ensure algorithm privacy.



The « strange » FHE computer

- No ifs (unless regularized by conditionnal assignment).
- No data dependant loop termination (need upper bounds and fixed-points).
- Array dereferencing/assignment in O(n) (vs O(1)).
- Algorithms always realize (at least) their wort-case complexity!
  - Complexity of dichotomic search?
- Can handle only a priori (multiplicative) bounded-depth programs.

Example: bubble sorting

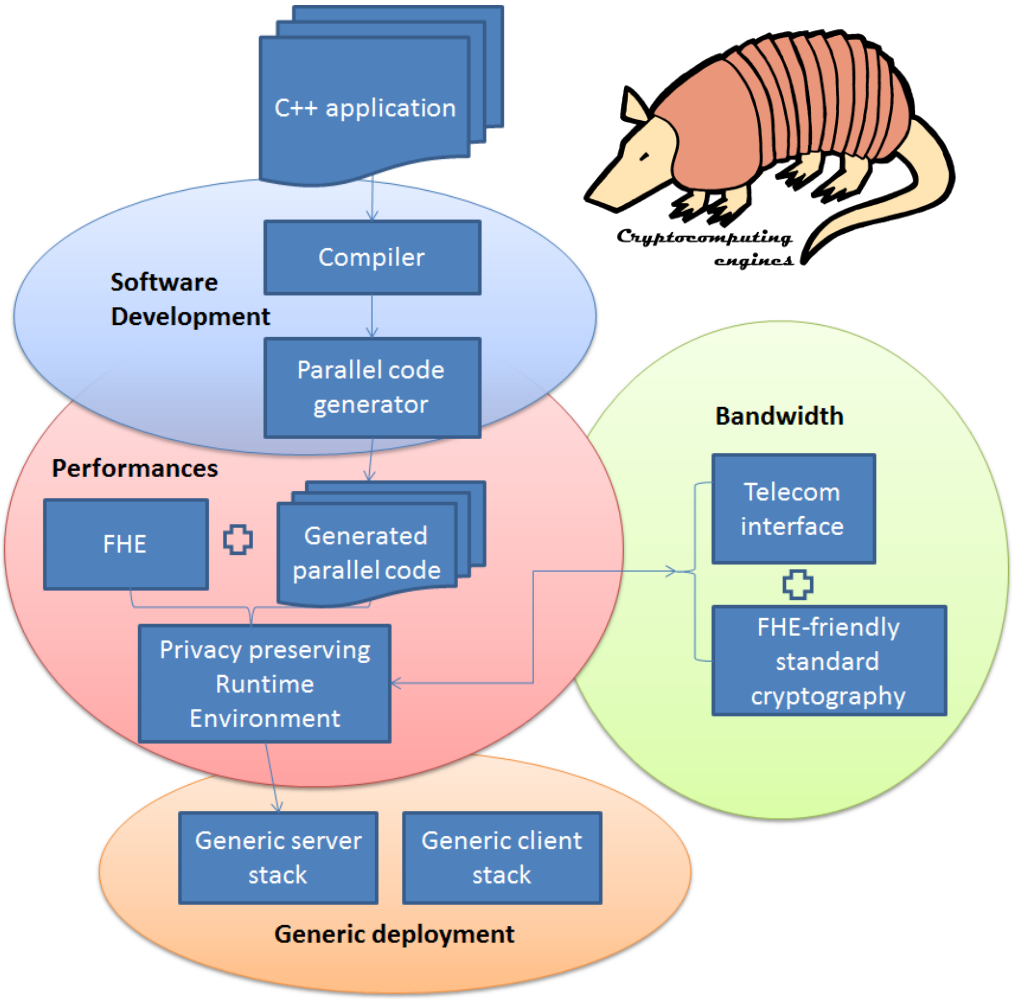
- Regularization of the inner if-then-else using a conditional assignment operator.
- Static control structure hence systematic worst case complexity.
  - A price to pay for not leaking any information.
- Still, this is generic C++ code.

```
template<typename integer>
void bsort(integer * const arr,
  const int n)
{
 assert(n>0);

 for(int i=0;i<n-1;i++)
 {
  for(int j=1;j<n-i;j++)
  {
   integer swap=arr[j-1]>arr[j];
   integer t=select(swap,arr[j-1],arr[j]);
   arr[j-1]=select(swap,arr[j],arr[j-1]);
   arr[j]=t;
  }
 }
}
```

Where select(c,a,b)≡c?a:b.

The Armadillo compiler & RTE…

- A compiler infrastructure for high-level cryptocomputing-ready programming, taking C++ code as input.
- Boolean circuit optimization (ABC-based), parallel code generation and « cryptoexecution » runtime environment.
- Optimized prototypes of the most efficient FHE systems known so far.
  - Also, with support of open source libs such as HELIB.



ASIACCS'15 (IACR Report 2014/988).

A magic trick

If AES^{-1}

then

$AES^{-1}([[$ $) =$

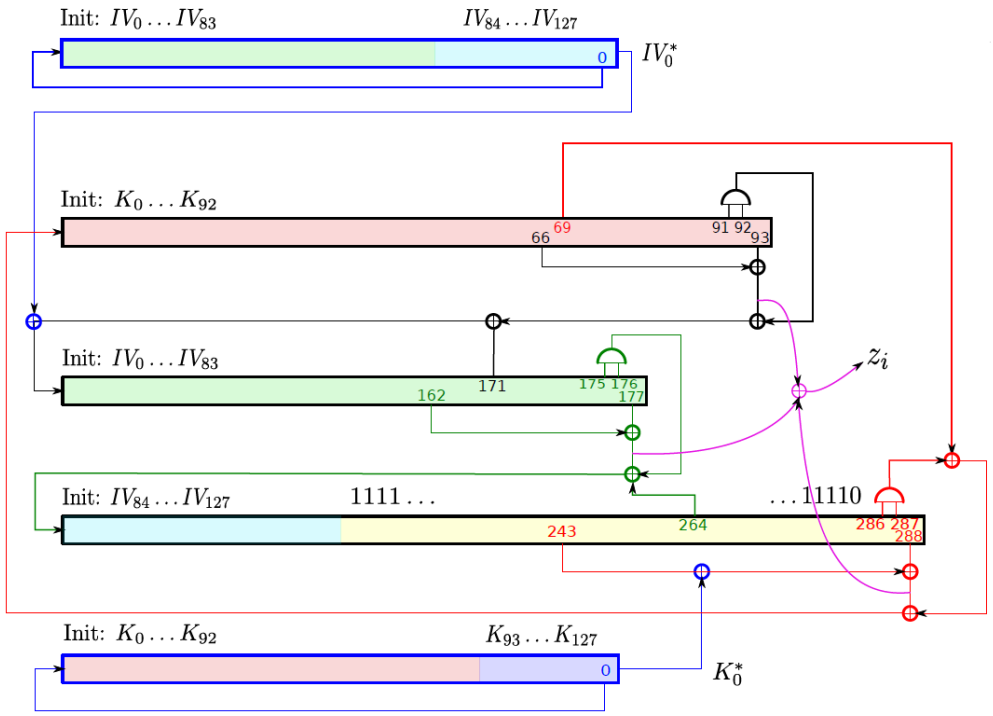
- Still, homomorphically executing an AES decryption still takes 18 mins with our best implementation (no batching).
  - With an intrinsic multiplicative depth of 40.
- Hence, more homomorphically-friendly symmetric systems are required.

Stream ciphers for « efficient » transciphering

- Keystream bits must be multiplicatively bounded.
  - This is the case if keystream bits are independent by chuncks (which is good for parallelism & batching).
- Keystream bits can be homomorphically « mined » independently of the data.
  - Hence, transciphering induces almost no latency (it's just an homomorphic XOR!) as long as keystream mining has been done in advance.
- Basic pattern:
  - Use an IV-based (FHE-friendly) stream cipher in « counter mode ».

« FHE-friendly » stream ciphers

- **TRIVIUM:**
  - A respected 80-bits key lightweight stream cipher.
    - Part of the ESTREAM portfolio (+ ISO/IEC 29192).
- **KREYVIUM:**
  - A « conservative » 128-bits key extension of TRIVIUM.



Algorithm	λ	FV			BGV		
		#ANDs	#XORs	keystream	#ANDs	#XORs	keystream
Trivium-12	80	3237	15019	57	3183	14728	45
Trivium-13	80	3474	16537	136	3474	16537	136
Kreyvium-12	128	3311	18081	46	3288	17934	42
Kreyvium-13	128	3564	19878	125	3561	19866	124

FSE'16 (& IACR Report 2015/113).

HELIB performances

Table 1. Latency and throughput for the algorithms using HElib on a single core of a mid-end 48-core server (4 x AMD Opteron 6172 processors with 64GB of RAM).

Algorithm	security level κ	N	used × depth	#slots	latency sec.	throughput bits/min
Trivium-12	80	45	12	600	1417.4	1143.0
			19	720	4420.3	439.8
Trivium-13	80	136	13	600	3650.3	1341.3
			20	720	11379.7	516.3
Kreyvium-12	128	42	12	504	1715.0	740.5
			19	756	4956.0	384.4
Kreyvium-13	128	124	13	682	3987.2	1272.6
			20	480	12450.8	286.8
LowMC-128	$? \leq 118$	256	13	682	3608.4	2903.1
			20	480	10619.6	694.3
LowMC-128 [ARS+15]	$? \leq 118$	256	13	682	3368.8	3109.6
			20	480	9977.1	739.0

FV performances

Table 2. Latency of our construction when using the FV scheme on a mid-end 48-core server (4 x AMD Opteron 6172 processors with 64GB of RAM).

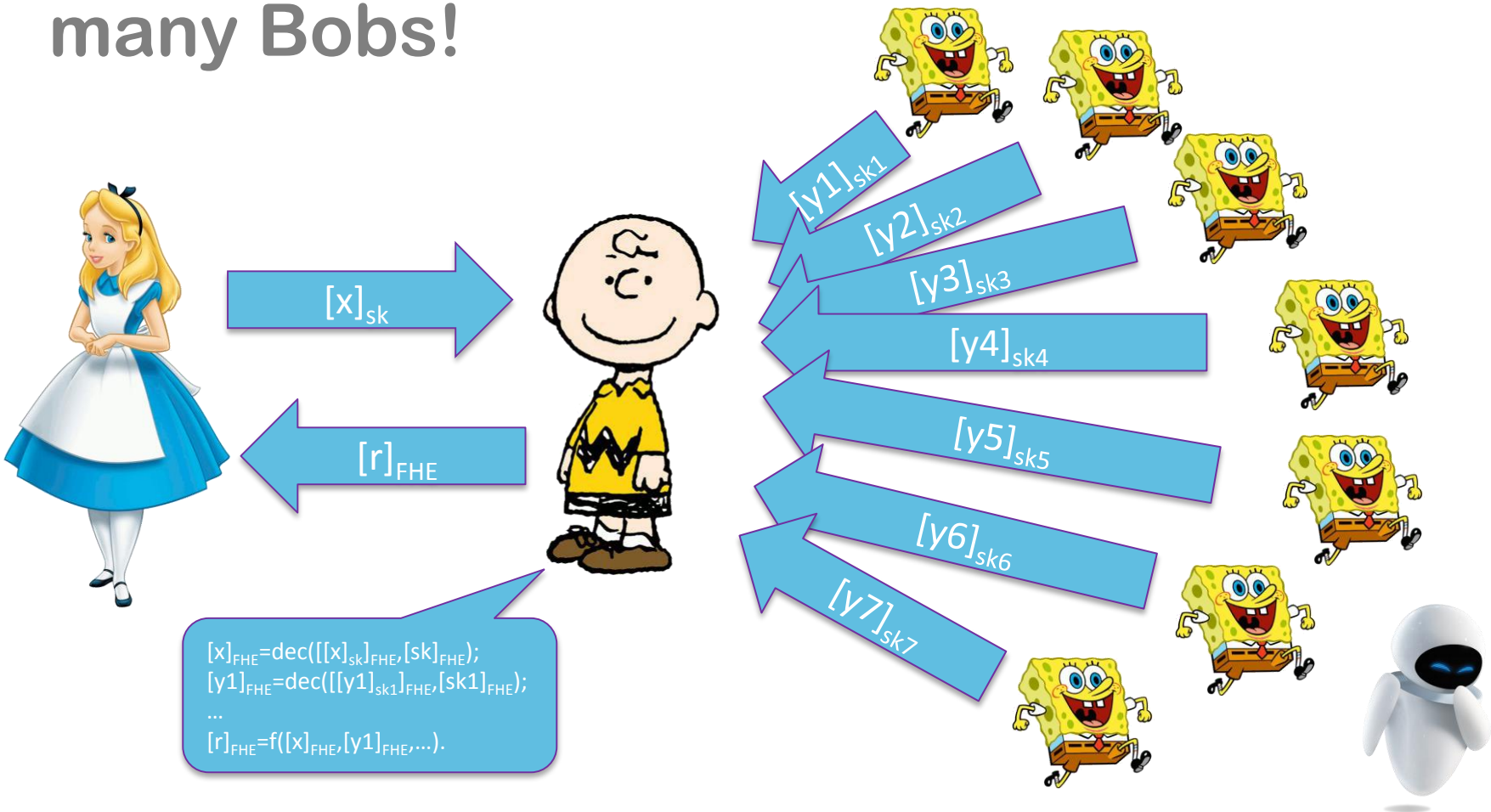
Algorithm	security level κ	N	used $\times$ depth	latency (sec.) 1 core	latency (sec.) 48 cores	Speed gain
Trivium-12	80	57	12	681.5	26.8	$\times$ 25.4
			19	2097.1	67.6	$\times$ 31.0
Trivium-13	80	136	13	888.2	33.9	$\times$ 26.2
			20	2395.0	77.2	$\times$ 31.0
Kreyvium-12	128	46	12	904.4	35.3	$\times$ 25.6
			19	2806.3	82.4	$\times$ 34.1
Kreyvium-13	128	125	13	1318.6	49.7	$\times$ 26.5
			20	3331.4	97.9	$\times$ 34.0
LowMC-128	$? \leq 118$	256	14	1531.1	171.0	$\times$ 9.0
			21	3347.8	329.0	$\times$ 10.2

Hey, but you are not doing all this just to execute crypto algorithms on encrypted data!

- No there are indeed a number of genuine reasons:
  - To avoid the computational burden of FHE-encryption on the client device (bad reason).
  - To avoid the intrinsic bandwidth inflation of transmitting FHE-encrypted data from the device (bad reason).
  - To (almost) transparently interface the client device with a remote « cryptocomputer » (good reason).
  - To use (almost) standard crypto on the client device (good reason).

And so…

- Charlie can do something useful for Alice by **blindly** aggregating the data of many Bobs!



Example of pilot

- A dummy-yet-realistic « Wikipedia-inspired » medical diagnosys.

- Setup:
  - Algorithm implementation, compilation and deployment on a server.
  - Homomorphic precalculation of Kreyvium keystream on the server.
  - The Android tablet sends the Kreyvium-encrypted private user health data.
  - The server receives and homomorphically « transcrypts » to FHE.
  - The server homomorphically executes the diagnostic algorithm and sends back the encrypted answer to the tablet.
  - As the FHE secret key owner, the tablet is the only party able to decrypt and thus interpret the server reply.

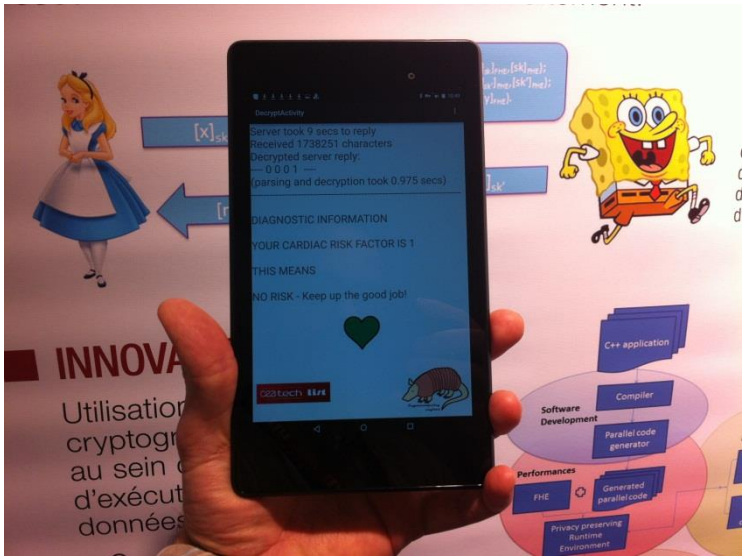
- Characteristics:
  - Fan-Vercauteren sFHE.
  - Full-blown end-to-end 128 bits security.
  - 3.3 secs for program execution on the server (with 8 cores activated).
  - < 4 secs RTD towards servers.

- Claim: practicality achieved for not-too-big-data realistic algorithms!

IEEE CLOUD'16.

- Facteur de risque cardiovasculaire :

+1 si homme d'âge > 50 ans.
+1 si femme d'âge > 60 ans.
+1 si antécédents familiaux.
+1 si fumeur.
+1 si diabètes.
+1 si hypertension.
+1 si taux HDL < 40.
+1 si poids > taille-90.
+1 si activité physique/jour < 30.
+1 si homme consommant plus de 3 verres/jour.
+1 si femme consommant plus du 2 verres/jour.



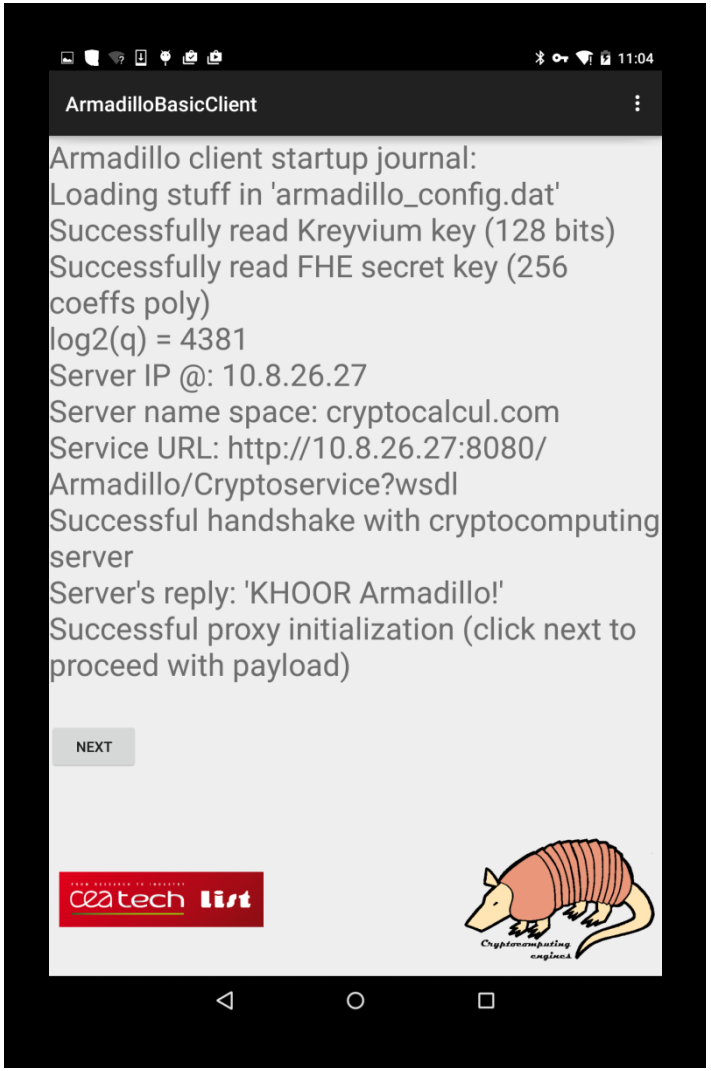

Institut d'Optique
École Polytechnique
Avenue Borotra
Avenue Le Chatelier
Rue Charles Gounod
Rue Alfred de Musset
Boulevard des Maréchaux
Avenue Fresnel
Avenue Coriolis
D128
N118
Google
FortiClient VPN
ArmadilloBasic...
11:04

Server side:

- FHE.pk
- $[SYM.sk]_{FHE}$
  - I.e. SYM.sk encrypted under FHE.pk

User side:

- FHE.pk
- FHE.sk
- SYM.sk (symmetric key)

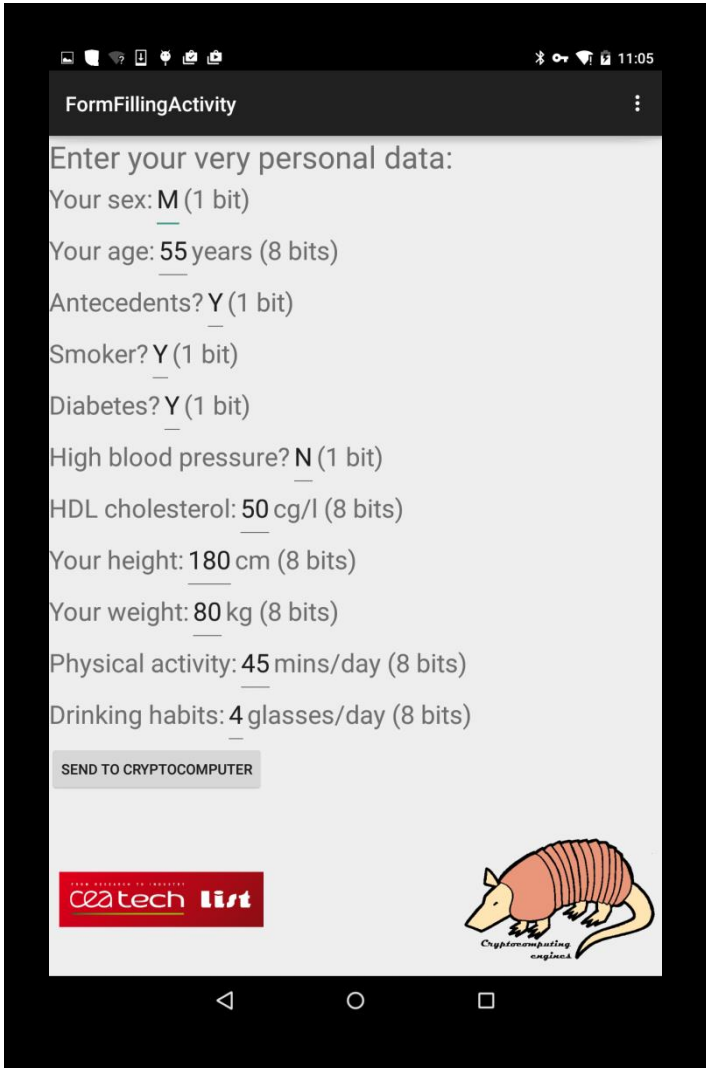


Server side:

- **FHE.pk**
- **$[SYM.sk]_{FHE}$**

User side:

- **FHE.pk**
- **FHE.sk**
- **SYM.sk**

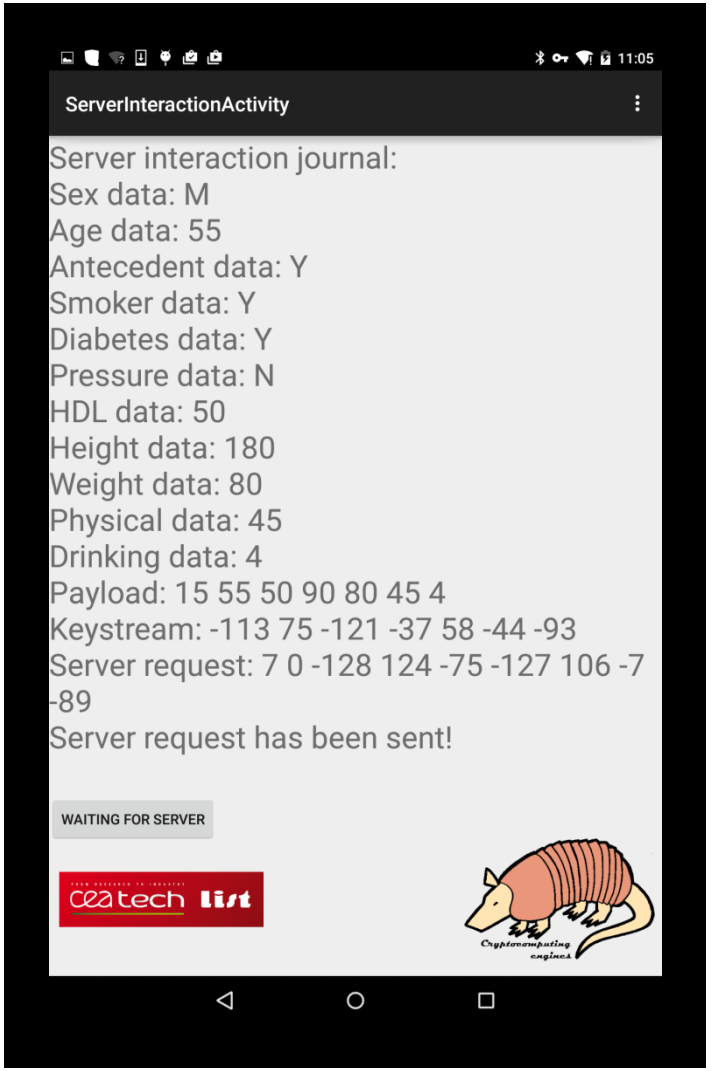


Server side:
- **FHE.pk**
- **$[SYM.sk]_{FHE}$**

User side:
- **FHE.pk**
- **FHE.sk**
- **SYM.sk**

Uses $[SYM.sk]_{FHE}$ to « transcipher » data encrypted under SYM.sk to data encrypted under FHE.pk

Private data encrypted with SYM.sk



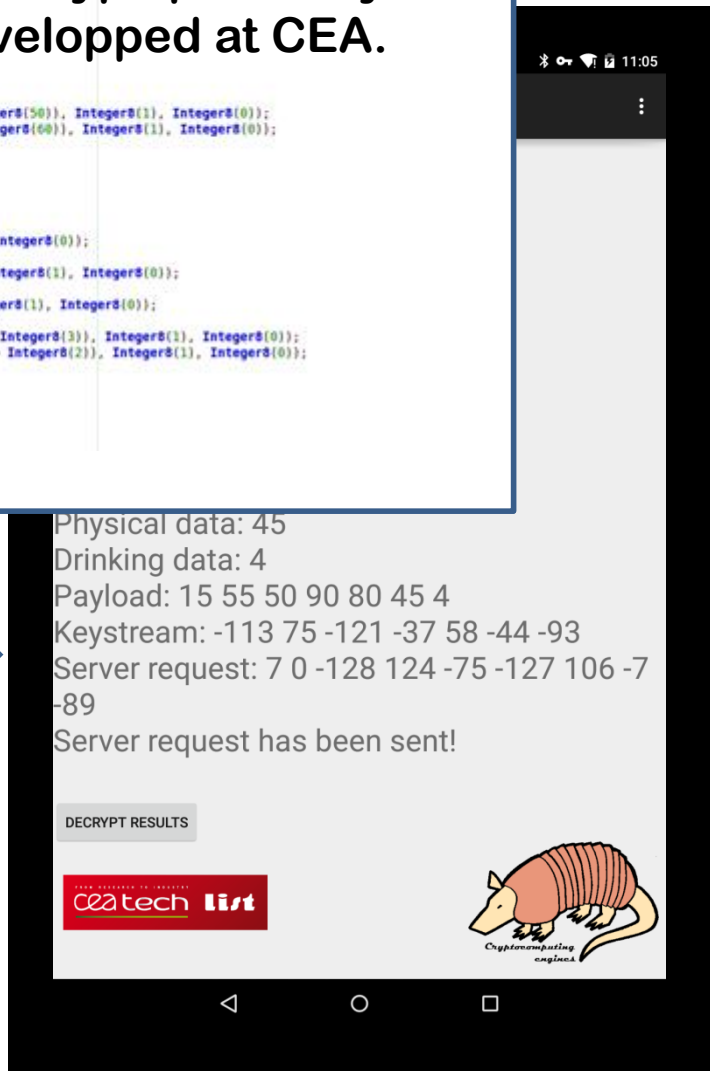
Server side:

- FHE.pk
- $[SYM.sk]_{FHE}$

```
cin >> flags;
cin >> age;
cin >> hdl;
cin >> height;
cin >> weight;
cin >> physical_act;
cin >> drinking;

Integer8 keystream[7];
// Read the pre-calculated keystream.
for(int i=0;i<7;i++)
    cin>>keystream[i];

flags ^= keystream[0];
age ^= keystream[1];
hdl ^= keystream[2];
height ^= keystream[3];
weight ^= keystream[4];
physical_act ^= keystream[5];
drinking ^= keystream[6];

risk = risk + select((flags[SEX_FIELD]) && (age > Integer8(50)), Integer8(1), Integer8(0));
risk = risk + select((!flags[SEX_FIELD]) && (age > Integer8(60)), Integer8(1), Integer8(0));

risk = risk + Integer8(flags[ANTECEDENT_FIELD]);
risk = risk + Integer8(flags[SMOKER_FIELD]);
risk = risk + Integer8(flags[DIABETES_FIELD]);
risk = risk + Integer8(flags[PRESSURE_FIELD]);

risk = risk + select(hdl < Integer8(40), Integer8(1), Integer8(0));

risk = risk + select(weight - height > Integer8(10), Integer8(1), Integer8(0));

risk = risk + select(physical_act < Integer8(30), Integer8(1), Integer8(0));

risk = risk + select((flags[SEX_FIELD]) && (drinking > Integer8(3)), Integer8(1), Integer8(0));
risk = risk + select((!flags[SEX_FIELD]) && (drinking > Integer8(2)), Integer8(1), Integer8(0));

//for (int i = 3; i >= 0; --i)  {
   //cout << risk[i];
//}
cout << risk << endl;
```

In the end, it is real code passed through a working FHE compiler prototype presently developped at CEA.

Homomorphically encrypted results



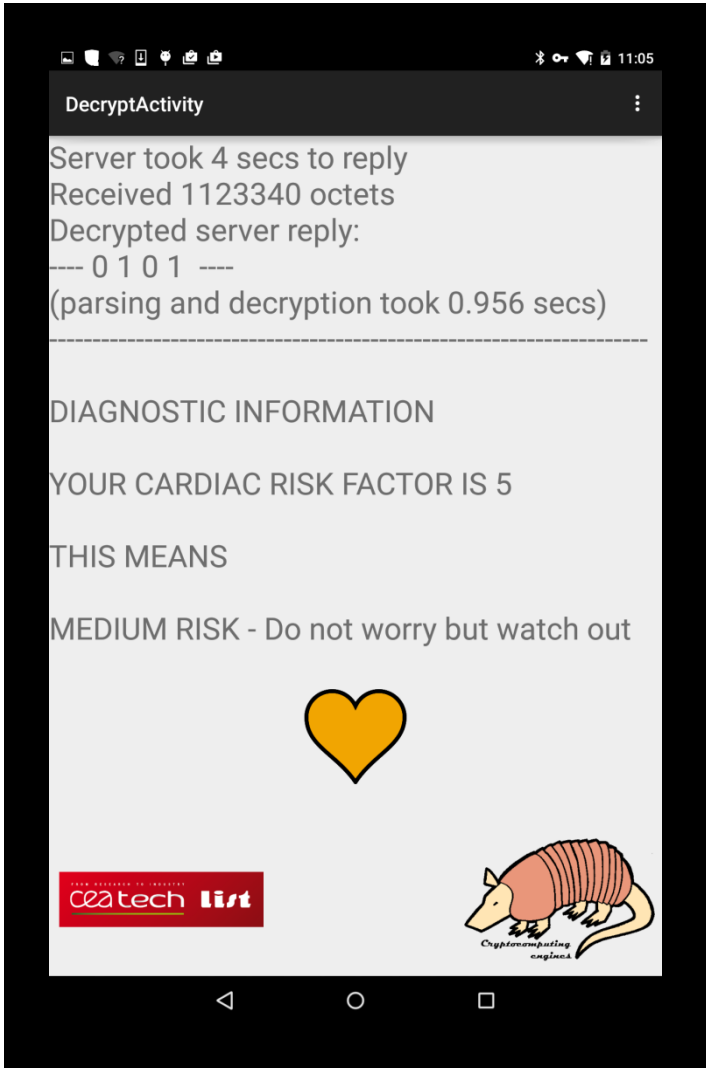
Server side:

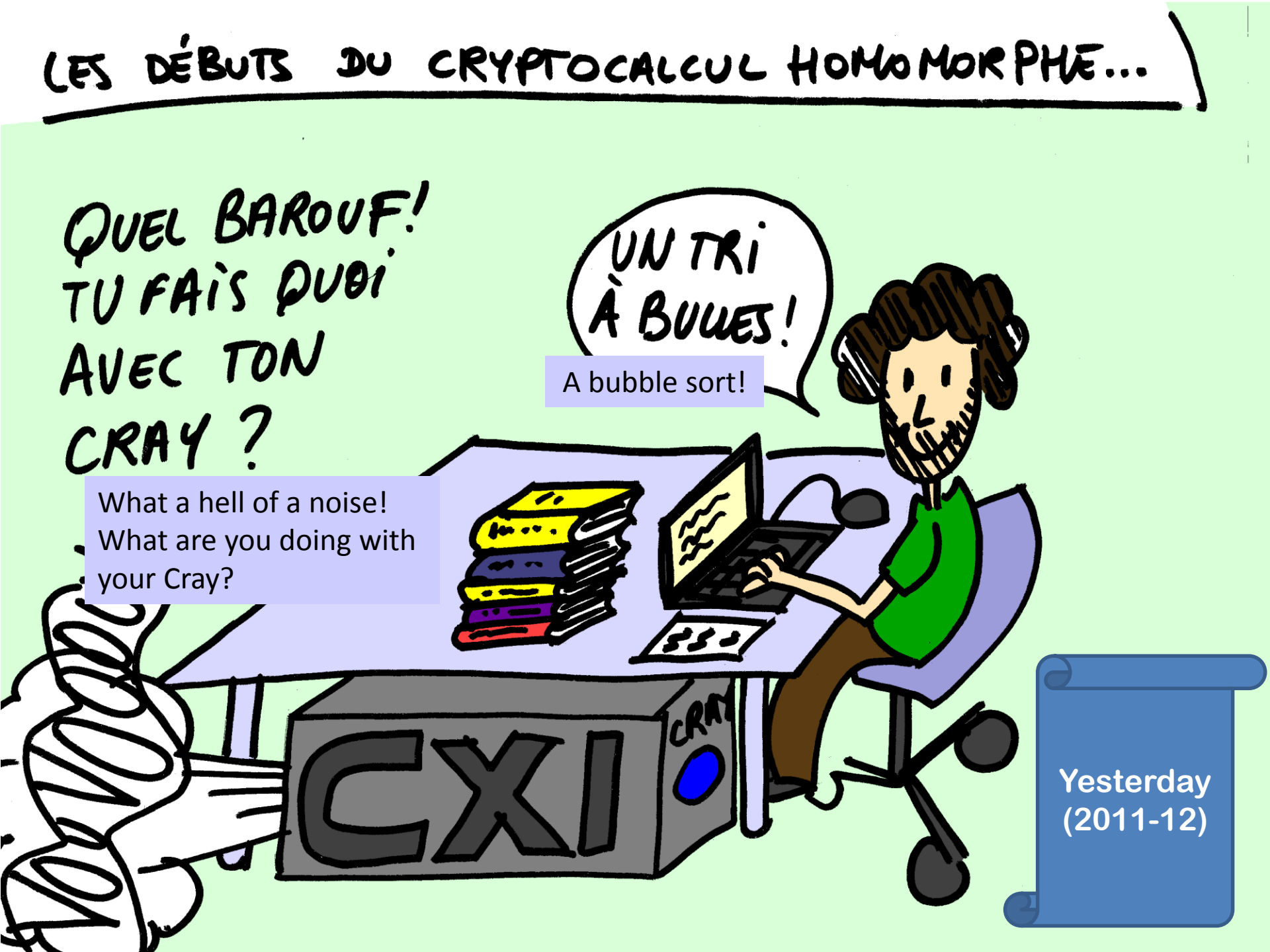
- **FHE.pk**
- **$[SYM.sk]_{FHE}$**

User side:

- **FHE.pk**
- **FHE.sk**
- **SYM.sk**

Uses FHE.sk to decrypt and interpret results




LES DÉBUTS DU CRYPTOCALCUL HOMOMORPHE...
QUEL BAROUF! TU FAIS QUOI AVEC TON CRAY ?
What a hell of a noise! What are you doing with your Cray?
UN TRI À BULLES !
A bubble sort!
CRAY
Yesterday (2011-12)


DecryptActivity
Server took 9 secs to reply
Received 1738251 characters
Decrypted server reply:
---- 0 0 0 1 ----
(parsing and decryption took 0.975 secs)
DIAGNOSTIC INFORMATION
YOUR CARDIAC RISK FACTOR IS 1
THIS MEANS
NO RISK - Keep up the good job!
C++ application
Software Development
Performances
FHE
Today (2015-16)

And perhaps tomorrow…

Acknowledgments

- C. Aguilar (IRIT) ; A. Canteaut (INRIA) ; S. Carpov (CEA) ; G. Costantino (CNR) ; P. Dubrulle (CEA) ; S. Fau (CEA) ; C. Fontaine (LABSTICC) ; G. Gogniat (LABSTICC) ; M. Izabachène (CEA) ; T. Lepoint (CryptoExperts) ; D. Ligier (CEA) ; F. Martinelli (CNR) ; M. Naya-Plasencia (INRIA) ; P. Paillier (CryptoExperts) ; T. Nguyen (CEA) ; O. Stan (CEA).