Carlos Aguilar-Melchor, Simon Fau, Caroline Fontaine, Guy Gogniat, and Renaud Sirdey

# Recent Advances in Homomorphic Encryption

[A possible future for signal processing in the encrypted domain]




Since the introduction of the notion of privacy homomorphism by Rivest et al. in the late 1970s, the design of efficient and secure encryption schemes allowing the performance of general computations in the encrypted domain has been one of the holy grails of the cryptographic community. Despite numerous partial answers, the problem of designing such a powerful primitive has remained open until the theoretical breakthrough of the fully homomorphic encryption (FHE) scheme published by Gentry in the late 2000s. Since then, progress has been fast-paced, and it can now be reasonably said that practical homomorphic encryption-based computing will become a reality in the near future.

This article surveys recent advances in (somewhat) FHE both from a cryptographic and software engineering point of view. It also provides practical experimental results obtained with an implementation of one of the (so far) most promising somewhat FHE scheme.

## INTRODUCTION

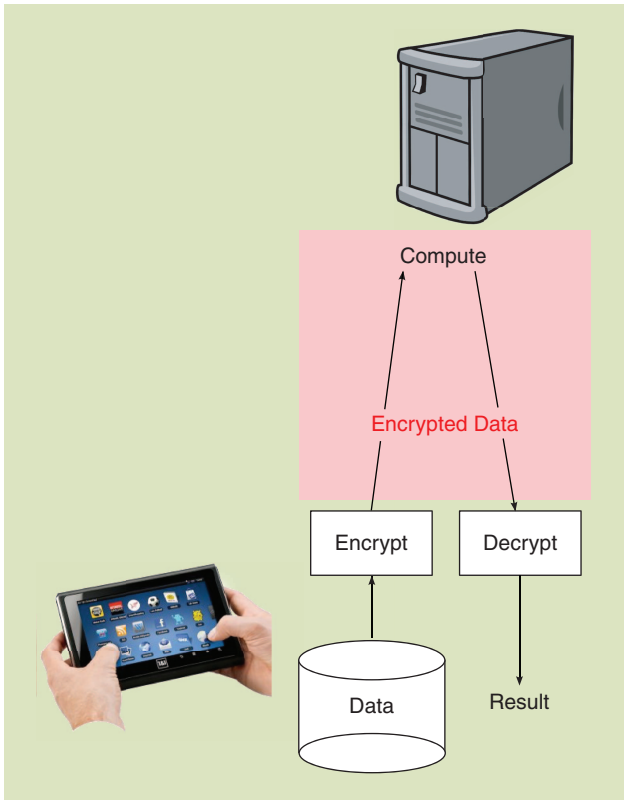
Traditionally, data confidentiality is a matter of cryptographers and is addressed through the design and use of encryption schemes. But while there is a permanent need of common encryption methods like RSA or the Advanced Encryption Standard (AES), interest in specific schemes has grown and spread during the last 30 years to more and more fields, encompassing signal processing. This is most notably due to the deployment of multimedia content distribution platforms, the development of biometric techniques, and the widespread adoption of the cloud computing model for more and more critical applications.

In such applications, some parties (often end users) may want to preserve the privacy of the data they outsource, or of their requests to servers. As a straightforward example, an end user might want to preserve the confidentiality of his e-mails while still being able to set up filters or to perform searches. Usually, such privacy issues are addressed with the help of encryption schemes. This leads to a need for encryption techniques that must be compliant with the storage and processing of outsourced encrypted data in the cloud, private information retrieval, (private) search on or analysis of encrypted data, proxy processing of broadcasted encrypted signals, etc. Figures 1–3 illustrate some generic scenarios that can be combined and encompass such use cases. Of course, the servers that will have to process such encrypted data or requests must provide as good and relevant results as if data were not encrypted. Technically speaking, the results provided by the server have to be decrypted

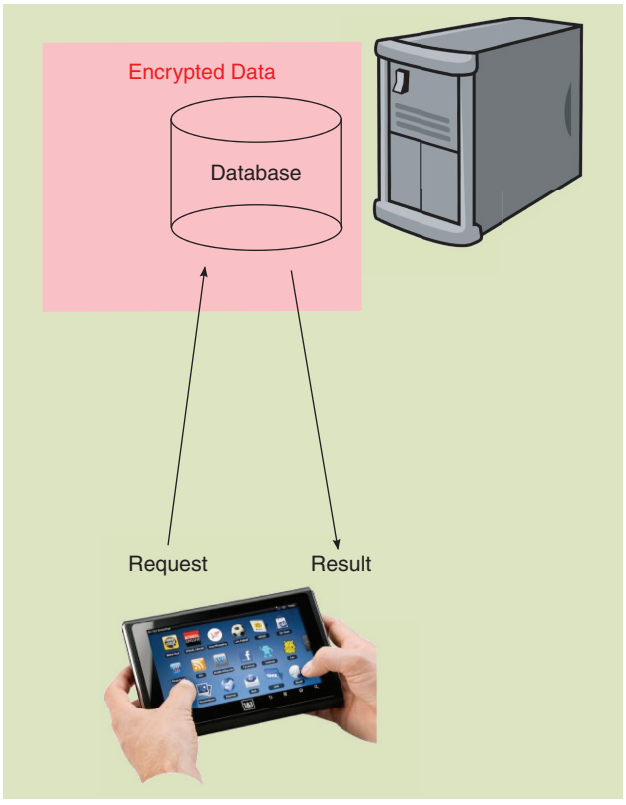
*Digital Object Identifier 10.1109/MSP.2012.2230219*
*Date of publication: 12 February 2013*



1053-5888/13/$31.00©2013IEEE




[FIG1] A need for processing encrypted data in a scenario of outsourced computation. For example, applying a one-shot enhancement algorithm on a private image or, more futuristically, performing an automatic on-the-fly translation of private voice calls.




[FIG2] A need for preserving the privacy of outsourced data, while being able to proceed to some requests on them. A typical example is keyword searching or filtering on an outsourced encrypted e-mail box.

by the client to be usable. For consistency, the decrypted result has to be equivalent to the intended computed value if performed on the original data.

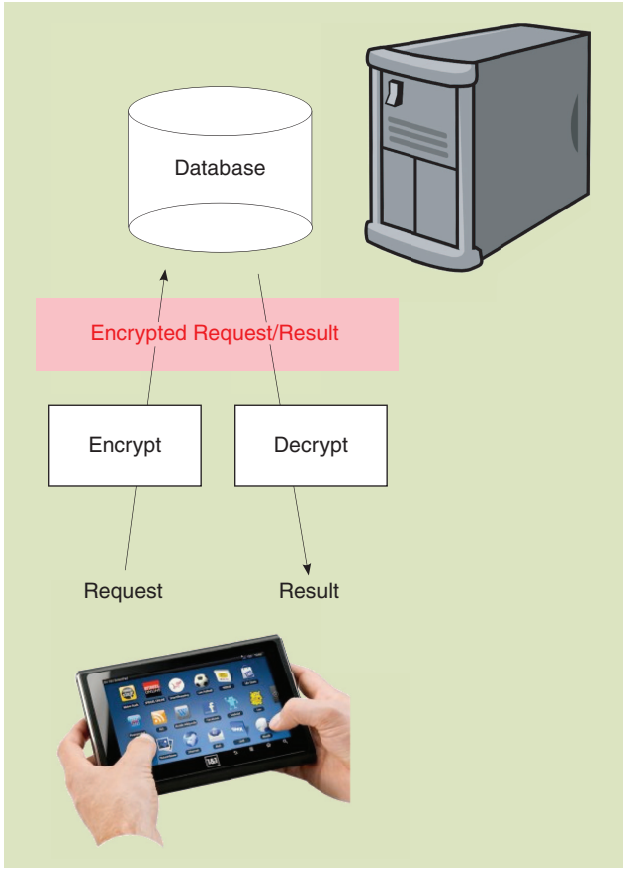
### *(SOMEWHAT) (FULLY) HOMOMORPHIC ENCRYPTION'S STORY IN A NUTSHELL*

In 1978, Rivest, Adleman, and Dertouzos proposed to solve these privacy issues through what is now called homomorphic encryption [28]. Since this seminal paper, the design of efficient and secure homomorphic encryption schemes has been one of the holy grails of the cryptographic community.

Before going deeper in the subject, it is important to notice that for security reasons such encryption schemes are necessarily probabilistic. This means that for a given encryption key, each plain text can be encrypted in several different ciphertexts. This implies that the set of possible ciphertexts is significantly larger than the set of possible plain texts. In other words, this implies that the ciphertexts are longer than the plain texts. For a given probabilistic encryption scheme, the ratio between these two lengths is called the expansion of the scheme. Of course, designers try to propose schemes with the smallest possible expansion for a given security level.

From 1978 to 2008, several homomorphic encryption schemes have been published, e.g., the famous Paillier's scheme and its derivatives, which are able to process encrypted data but with only one kind of operator (additions or multiplications) at a time [15]. In 2009, Gentry proposed the first not yet broken FHE scheme [16]. FHE refers to cryptosystems that are able to process both additions and multiplications in the encrypted domain. With such schemes, any polynomial function over encrypted data can be computed. It was really a breakthrough after 30 years of huge efforts, as it opens the way to many more powerful real applications than before. Nevertheless, due to its huge algorithmic complexity, large key size, and ciphertext expansion, current FHE schemes remain today not efficient in practice. Since 2009, many publications provided variants and improvements. In particular, several so-called somewhat FHE cryptosystems have been proposed, which allow any number of additions but a bounded number of multiplications [2], [19]. These schemes are really interesting as they are less complex than the fully homomorphic ones and are able to process a number of multiplications that are sufficient for most applications. Hence, they are considered today as the most promising schemes for practical applications.

But despite these promising characteristics, their overhead remains today too high to make them directly usable in practice. There are mainly two ways to improve their efficiency. The first one is to propose new tricky variants that are less complex. The other one is to find some crafty way to implement them. Unfortunately, very few implementations have been




[FIG3] **A need for performing private requests on public data. A typical example would be deep packet inspection without revealing, for example, the traced IP address.**

published and publicly discussed yet, to measure how far we stand from their use in real applications. The original experimental results presented in this article contribute in answering this precise question.

### *WHAT IS THE IMPACT ON SIGNAL PROCESSING?*

Processing signals in the encrypted domain is an important challenge. In recent years, more and more researchers designed specially tailored solutions dedicated to many applications. Without being exhaustive, one can mention privacy-enhanced face recognition [11], privacy-preserving elecrocardiogram signal classification [4], privacy protection of biometric data [3], [14], buyer-seller protocols [22], [26], [20], and zero-knowledge watermark detection [1], [25], [29]. In parallel, other works developed some general tools for processing some particular operations on encrypted signals, which can be useful in many applications, e.g., Gram-Schmidt orthogonalization [13], discrete cosine transform computation [5], and discrete Fourier transform computation [6]. Finally, one can mention general discussions on the processing of encrypted signals [12] and attempts to find adequate representations for such processing, as in [7].

These publications rely on regular homomorphic encryption. Hence, when needed, computing overencrypted data functions involving both additions and multiplications is really tricky. It requires linearizing the computation in an ad hoc manner and using multiparty computation techniques. This demands the use of heavy protocols, designed precisely for each application. Moreover, these protocols need many interactions between the parties to do the job correctly. For more details on the issues of privacy in signal processing applications and on how homomorphic encryption can help to solve them, we refer the reader to [23].

With an accessible (somewhat) FHE scheme, it would be possible to compute polynomial functions directly, without linearization or multiparty computation techniques. Of course, the cost to pay would be directly related to the complexity of the (somewhat) FHE scheme used. But according to recent works, the complexity of such schemes is currently dropping down faster than expected, even one year ago. There is still much work to do to get a very efficient scheme, but each step forward makes real applications closer than before.

Many algorithmic improvements have been and will be published to make implementations of (somewhat) FHE schemes more and more tractable. Nevertheless, providing practical implementations of such schemes remains a main challenge. Indeed, these schemes are difficult to understand and even harder to implement. It explains why there are very few implementations available today. Another difficulty to face is their rapid evolution and improvement. Hence, implementations need to evolve as quickly as the schemes, to take advantage of each important improvement.

In this article, we focus on the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [8], which seems today the most promising somewhat FHE scheme, but which single previous implementation discussed in the literature is focused on the evaluation of AES [19]. After explaining why it is more interesting in practice to use somewhat FHE schemes instead of FHE schemes, we briefly present BGV. As it operates on bits and, as in most applications, we need to process integers, we then discuss how we can deal with integers instead of bits. In a following section, an implementation framework is proposed to minimize the design effort of the programmer when implementing functions that process encrypted data. Finally, we provide the first practical results of a general implementation of the BGV scheme and compare it with the few other available implementations of other schemes. The purpose of this last section is to provide the reader with some concrete data about the implementation and performances of such a scheme.

## SOMEWHAT FULLY HOMOMORPHIC ENCRYPTION

The efficiency of the first FHE schemes let little hope for practicality. To understand why FHE schemes have such a big overhead, let us first look at the scheme introduced by van Dijk et al. in 2010 [30].

This scheme is built upon integers. In its symmetric version, the secret key is an odd integer $p$ (where size depends on security criteria). To encrypt a bit $m \in \{0,1\}$, we pick random

integers $q$ and $r$ (into ranges defined by security concerns) such that $2r < p/2$ and set

$$c = Enc(m) = qp + 2r + m.$$

The plain text $m$ can be retrieved by computing $c \bmod p \bmod 2$.

Now let us consider two plain texts $m_1$ and $m_2$ and the corresponding ciphertexts $c_1$ and $c_2$. As the purpose of an FHE scheme is to perform computations on encrypted data, we now discuss what happens when we add or multiply two ciphertexts

$$c_1 + c_2 = (q_1 + q_2)p + 2(r_1 + r_2) + m_1 + m_2.$$

Decrypting $c_1 + c_2$, one needs to get $m_1 + m_2$. This is possible when the condition

$$2(r_1 + r_2) + m_1 + m_2 \leq p \tag{1}$$

is verified. In this case, we have

$$c_1 + c_2 \bmod p = 2(r_1 + r_2) + m_1 + m_2,$$

and $m_1 + m_2$ can be retrieved by computing $c_1 + c_2 \bmod p \bmod 2$. Since we pick $r$ such that $2r < p/2$, condition (1) is satisfied when $c_1$ and $c_2$ are "fresh" ciphertexts. On the contrary, if $c_1$ is for example a sum of other ciphertexts, the condition might not be met and it is impossible to decrypt $c_1 + c_2$. The same phenomenon happens to an even worse degree when we multiply two ciphertexts. Constructing an FHE scheme then requires managing the remaining random part called noise [for example, $2(r_1 + r_2)$] by keeping it under a certain limit to ensure decryption.

The first way to solve this noise problem is called bootstrapping and was used in Gentry's first FHE scheme. The idea behind bootstrapping is to modify a somewhat FHE scheme so it can homomorphically run its own decryption procedure. Including the public key an encryption of the secret key, bootstrapping allows the transformation of a given ciphertext into a new ciphertext that encrypts the same bit but has lower noise. Unfortunately, bootstrapping implies a growth of the public key and the procedure to transform the ciphertexts is prohibitively heavy, as shown by the few results published on an implementation of an FHE scheme using bootstrapping [17]. Thus, obtaining schemes that allow to evaluate polynomials without bootstrapping is an interesting line of research (even if this implies to bound the degree of the polynomial we are able to manage).

Recently, Brakerski et al. (based on works of Brakerski and Vaikuntanathan [10], [9]) used the tensorial product approach introduced by Aguilar et al. [2] in a new alternative way that radically improves the performance of the scheme. Based on this approach and using two optimizations, Brakerski et al. [8] proposed a somewhat FHE scheme (BGV) that can be parameterized to compute homomorphically multivariate polynomials of bounded degree $d$, where $d$ can be virtually chosen as large as needed. Since the bootstrapping technique is hard to achieve in practice and its interest in terms of performance is not obvious, the solution brought by the BGV scheme of [8] seems one of the most promising at the time of this writing.

## OVERVIEW OF THE BGV SCHEME

BGV is an asymmetric encryption scheme that encrypts bits. Like most (somewhat) FHE schemes, it is based on lattices. There are two versions of the cryptosystem: one dealing with integer vectors [the security of which is linked with the hardness of the learning with errors (LWE) problem] and the other one with integer polynomials [the security of which is linked with the hardness of the ring-learning with errors (R)-LWE problem]. In a few words, the LWE [resp. (R)-LWE problem] problem consists of distinguishing between a distribution of $(a_i, b_i)$ sampled uniformly in $\mathbb{Z}_q^n \times \mathbb{Z}_q$ (resp. in the ring $\mathbb{A} = \mathbb{Z}_q^n/F(X)$ and a distribution of $(a_i, < a_i, s > + e_i)$, where $a_i$ and $s$ are sampled uniformly from $\mathbb{Z}_q^n$ (resp. $\mathbb{A}_q^n$) and $e_i$ is sampled according to a Gaussian distribution. For more precisions on the (R)-LWE problem, we refer the reader to [27]. In the sequel, we will focus on the polynomial version of the BGV encryption scheme, which seems more promising in terms of performances.

We consider the polynomial ring $\mathbb{A} = \mathbb{Z}[X]/F(X)$, where $F(X)$ is a cyclotomic polynomial of degree $d = 2^k$ and a chain of odd moduli $q_1 < \cdots < q_L$ and their corresponding subrings $\mathbb{A}_{q_i} = \mathbb{A}/q_i\mathbb{A}$ of polynomials of $\mathbb{A}$ with integers coefficients into the range $]-q_i/2, q_i/2]$. In practice, elements in $\mathbb{A}_{q_i}$ will be polynomials represented by the $d$-vector of their coefficients.

### BASIC ENCRYPTION FUNCTIONS

The private key *Priv* is sampled in $\mathbb{A}$. A public key *Pub* consists of the private key masked by a noise component: $Pub = aPriv + 2e \in \mathbb{A}_{q_L}^N$, where $N = O(\log q_L)$, $a \in \mathbb{A}_{q_L}^N$ and the noise $e$ is sampled from a "discrete" Gaussian distribution over $\mathbb{A}^N$ ("discrete" meaning here that we sample from a Gaussian distribution and round to the nearest integer). Here follows a set of black box descriptions of the main functions associated with the encryption scheme. We have decided not to include the exact algorithms to avoid drowning the important issues in technical descriptions. If interested, the reader can refer to [8] and [18] for a precise algorithmic description.

**`Encrypt`(Plain Text $m$, PublicKey *Pub*): Ciphertext $c$**

The integers we manipulate need to be encrypted 1 b at a time. For $m \in \{0, 1\}$, the resulting ciphertext $c$ is a pair of two elements in $\mathbb{A}_{q_L}$ derived from the plain text $m$, the public key *Pub* and a random seed (since it is a probabilistic scheme). In the following, a ciphertext can be transformed into a pair of two elements in any subring $\mathbb{A}_{q_i}$. In our implementation, each ciphertext carries its level, i.e., the information that indicates in which subring it lies.

**`Decrypt`(Ciphertext $c$, PrivateKey *Priv*): Plain Text $m$**

The decryption function is a simple dot product between the ciphertext $c \in \mathbb{A}_{q_i}$, and the private key followed by a modular

reduction into the range $]-q_i/2, q_i/2]$ and finally a parity test to retrieve the plain text $m$. As in the example given in the section "Somewhat Fully Homomorphic Encryption," the noise must be under a certain level for the decryption to be correct.

### *LEVEL SHIFTING OPERATIONS*

**`Rescale`** (Cipertext $c$): Ciphertext $c'$
The function transforms the ciphertext $c \in \mathbb{A}^2_{q_i}$ into a ciphertext $c' \in \mathbb{A}^2_{q_{i-1}}$. The resulting ciphertext has a reduced noise.

**`SwitchKey`**(Augmented Cipertext $c$): Ciphertext $c'$
The tensored product of two ciphertexts $c_1 \otimes c_2$ results in an "augmented ciphertext" $c \in \mathbb{A}^3_{q_i}$. To retrieve a regular ciphertext in $\mathbb{A}^2_{q_i}$, we essentially multiply $c$ by a public matrix (a different one for each level $1 < i < L$). Then we call the `Rescale` function to get $c' \in \mathbb{A}^2_{q_{i-1}}$ (with low noise).

### *HOMOMORPHIC OPERATIONS*

**`Add`**(Ciphertext $c_1$, Ciphertext $c_2$): Ciphertext $c_{sum}$
For two ciphertexts $c_1, c_2$ where $c_1 \in \mathbb{A}^2_{q_{i1}}$ and $c_2 \in \mathbb{A}^2_{q_{i2}}$, we follow these steps:

if $i_1 \neq i_2$ (for example $i_1 < i_2$) then
| do $c_2' \leftarrow$ `Rescale` $(c_2)$ $i_2 - i_1$ times; (at this point we have $c_1, c_2$ at the same level $i_1$)
end
do $c_{sum} \leftarrow c_1 + c_2'$; (simply by adding the coefficients of the polynomials modulo $q_{i_1}$)

**`Mul`**(Ciphertext $c_1$, Ciphertext $c_2$): Ciphertext $c_{mul}$
For two ciphertexts $c_1 \in \mathbb{A}^2_{q_{i1}}$ and $c_2 \in \mathbb{A}^2_{q_{i2}}$, we follow the steps:

if $i_1 \neq i_2$ (for example $i_1 < i_2$) then
| call $c_2' \leftarrow$ `Rescale` $(c_2)$ $i_2 - i_1$ times; (at this point we have $c_1, c_2$ at the same level $i_1$)
end
do $c_3 \leftarrow c_1 \otimes c_2'$; $(c_3 \in \mathbb{A}^3_{q_{i1}})$
do $c_{mul} \leftarrow$ `SwitchKey` $(c_3)$; $(c_{mul} \in \mathbb{A}^2_{q_{i_1-1}})$

The tensored product applied on $c_1$ and $c_2$ consists of adding and multiplying polynomials of $\mathbb{A}_{q_{i_1}}$, which can be very expensive as we will see.

### *PARAMETERS*

The size of the ciphertexts and therefore the cost of additions and multiplications on those ciphertexts depends on the size of the $\{q_i\}_i$ and on the size of the ring $\mathbb{A}$ (i.e., the size of $d$ or $n$). To give an idea of the cost of these operations, we want to stress that each bit is encrypted by a pair of polynomials that can be of degree $d > 10,000$ and have coefficients of size $> 200$ b. For security and noise management reasons, these parameters grow as the number of `Mul` increases (as shown in [8]). More precisely, the key value to dimension the cryptosystem is the multiplicative depth. In a Boolean circuit, the multiplicative depth is defined as the maximal number of multiplication gates on any path.

We can also already point out that the order in which we perform the homomorphic operations may have an impact on the number of times we have to call the `Rescale` and `SwitchKey` functions, therefore on the number of levels (multiplicative depth) we need.

## MANIPULATING INTEGERS IN THE ENCRYPTED DOMAIN

(Somewhat) FHE schemes allow the evaluation of any (bounded degree) polynomial from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2$ or, equivalently, any Boolean circuit. Recall that a Boolean circuit consists of a directed acyclic graph where vertices are either inputs, outputs, or operators (`AND` or `XOR`) and where edges represent data dependencies. In higher-level programming terms, working with (somewhat) FHE schemes restricts us to programs or algorithms having bounded input and a control flow that is independent of encrypted data. In particular, this a priori excludes (encrypted) data-dependent `if-then-else` statements as well as loop termination criteria. At first, this may seem highly restrictive. However, control depending on encrypted data can still be performed to some extent, as we shall see in this section.

Let us first see how an FHE scheme permits the implementation of pretty much any of the classical integer manipulation operators. Additions and multiplications can be implemented following textbook recipes for $n$-bit adders and multipliers (although choosing the most appropriate design for execution over an FHE scheme is not so straightforward). Because multiplications (`ANDS`) are particularly costly, the multiplier itself should be optimized when either both or one of the (encrypted) operands are Boolean, in which case there is only one layer of bit-level multiplication (`ANDS`), or when one of the operand is available in the clear, in which case the multiplication becomes a sequence of additions of shifted versions of the encrypted domain input.

Bitwise logical operators (`AND`,`OR`, `XOR` etc.) turn out to be easy to implement using the two basic cryptosystem operations. Negation (minus) can be implemented using the textbook trick of two-complementing: `XORing` all "cryptobits"—*cbits* in the sequel—with an encryption of one, to complement them, and adding an encryption of one (with carry propagation) to the result. This allows to implement an $n$-bit subtraction operator using an $n$-bit adder. Also, when subtraction is implemented that way, the most significant *cbit* provides the sign of the integer, a fact that can be known and used by the "cryptocomputer" despite the fact that it has no access to the effective value of that bit as it is itself locked in the encrypted domain.

It is then also possible to perform comparisons hermetically in the encrypted domain. Although there are a number of ways to implement comparison operators, we have designed our operators so as to avoid multiplications (`ANDS`) as much as possible. Our solution thus consists of starting from the `less than` operator, which can be implemented by subtracting the two operands and then by producing a result which consists of

$n-1$ leading encryptions of zero followed by the most significant bit of the subtraction result, i.e., the aforementioned sign *cbit* (which is in this case stored in the least significant bit). The `greater than` operator is performed similarly. Note that following the execution of such an operator, the "cryptocomputer" knows (legally) that there is only 1 b of payload in the result and can exploit that fact in further calculations (most importantly in multiplier optimizations as already stated). The (Boolean) `not` operator can be obtained by `XOR`ing the least significant bit with an encryption of one. Having both the `less than`, `greater than`, and `not` operator, the `equal to` operator can be performed as well [which allows the implementation of the $\delta$ function used in (3) and (4) below] in a fashion that is suboptimal with respect to the number of gates but much less involved in terms of multiplications than more classical designs.

Finally, left and right bitshift operators can also be obtained hermetically in the encrypted domain. The left bitshift operator requires copying the relevant rightmost *cbits* of its operand and then (right) padding with as many encryptions of zero as required. The right bitshift operator, on the other hand, requires copying the relevant leftmost *cbits* of its operand and then (left) padding with as many copies of the most significant *cbit* (i.e., the sign *cbit*) of that operand that lives in the encrypted domain. Left and right rotations can also be implemented by moving *cbit* around.

Now that these classical operators are available, we can go back to the data-dependant control issue. Let us consider a selection operator $\texttt{select}: \mathbb{Z}_2 \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ such that

$$\texttt{select}\ (c, a, b) = \begin{cases} a & \text{if } c = 1 \\ b & \text{otherwise.} \end{cases}$$

Such an operator can then straightforwardly be rewritten as follows:

$$\texttt{select}\ (c, a, b) = ca + (1 - c)b. \tag{2}$$

Provided the implementations of addition, multiplication and negation mentioned earlier in this section, (2) translates as

$$\texttt{select}\ (c, a, b) = ca\ \texttt{xor}\ (\texttt{not}\ c)\ b.$$

As this construction allows to perform a conditional assignment operator, it enables the implementation of a wide range of algorithms. As an example, consider the following simple (although quite demonstrative) example of a bubble sort algorithm that may be expressed as follows in C-style programming languages:

```
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;
   }
 }
}
```

Using the selection operator of (2), this algorithm can be rewritten in a suitable fashion for execution over an FHE scheme, that is, without requiring any access to the value of the test `arr[j−1]>arr[j]`:

```
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=select(gt,arr[j-1],arr[j]);
   arr[j-1]=select(gt,arr[j],arr[j-1]);
   arr[j]=t;
  }
 }
}
```

Still, it should be emphasized that, expressed as above, the bubble sort algorithm always achieves its worst-case $O(n^2)$ complexity: this is a price to be paid unless one accepts leaking information about the sorted data.

This bubble sort example is demonstrative and reveals that pretty complex algorithms can be realized over an FHE scheme. Furthermore, recall that sorting is a naïve algorithm for computing the median of a sample [21], thus allowing to construct nonlinear digital signal processing primitives such as a median filter. At that point, it should be clear that almost any nonlinear signal or image processing primitive (thresholding, mathematical morphology operator, etc.) can be performed. As a more advanced example, as long as one is able to homomorphically evaluate the objective function of an optimization problem, at least in theory, then a full-blown simulated annealing algorithm, which is often used to solve inverse problems in both signal and image processing, can be performed homomorphically. The selection operator allows for keeping track of the best solution encountered while executing the algorithm and also allows the performance of the randomized temperature-driven acceptance rule for a new solution

$$\text{if } u \le e^{-\frac{c(\omega') - c(\omega)}{T}} \text{ then } \omega = \omega',$$

where $u$ is chosen uniformly in $[0, 1]$, $\omega$ and $\omega'$ respectively denote the current and the candidate solution, $c(\omega)$ denotes the cost of solution $\omega$, and $T$ denotes the temperature.

Now, if $\omega, \omega', c(\omega)$, and $c(\omega')$ need to remain private, we use encryptions of these values, noted $\overline{\omega}$, $\overline{\omega'}$, $\overline{c(\omega)}$, and $\overline{c(\omega')}$. In the encrypted domain, the condition above then translates to

$$Enc(u) \leq e^{-\frac{\overline{c(\omega')}-\overline{c(\omega)}}{Enc(T)}},$$

where $\leq$ is performed homomorphically as described earlier. Let us call $\overline{\alpha}$ an encryption of the Boolean associated to this condition. Thanks to the selection operator, we can then perform (homomorphically)

$$\overline{\omega} = \texttt{select}(\overline{\alpha}, \overline{\omega'}, \overline{\omega}).$$

It turns out that array dereferencing and assignment with encrypted indices is also possible. Indeed,

$$t[i] = \sum_{j=1}^{n} \delta(i, j)\, t[j], \tag{3}$$

with $\delta(i, j) = 1$ if $i = j$ and zero otherwise. Similarly, array assignment $(t[i] = v)$ can be done by performing

$$t[j] = \delta(i, j)v \oplus (1 - \delta(i, j))t[j], \forall j. \tag{4}$$

Of course, both operations are done in $O(n)$ rather than $O(1)$ in the clear index case. It should also be emphasized that, as a result of an assignment, all the array entries change although all but one of them decrypts to the same value as before the assignment. Again, this is a price to pay for index privacy.

Some of the above operators involve inserting encryptions of zero or creating multiple copies of certain *cbit* such as the sign *cbit* of a difference. Note that, due to the probabilistic nature of the FHE scheme underlying the calculation, the cryptocomputer loses track of these values as soon as they are involved in a further operation. For example, adding (XORing) a *cbit*, say $c_0$, known to be an encryption of one (because the encryption has been performed by the cryptocomputer as part of the data it injects in the calculation) to another *cbit* of unknown value necessarily leads, by construction of the cryptosystem, to a result which has nothing to do with $c_0$ and, thus, which does not allow to (practically) infer any information about the value of the *cbit* of unknown value.

As already hinted at the beginning of this section, thanks to the above machinery, we are well armed to express many high-level algorithms: cryptographic ones (in particular, we have been able to write SHA-1 and RC4 within this framework) or noncryptographic ones (as discussed later). We shall now see how all these can be put together so as to obtain a full solution from a software engineering point of view.

## EXPRESSING HIGH-LEVEL ALGORITHMS

Having defined integer manipulation operators, we are now (in theory) in a position to express many high-level algorithms in a natural fashion. This can easily be done using the operator overloading features of object-oriented programming languages such as C++, for example via a `CryptoBit` class provided with + and * operators and by using it to build a `CryptoInt` class provided with the operators specified in the previous section.

However, from a software engineering point of view, it is desirable to be able to do more and in particular to be able from a single code to perform the following tasks:

1) test and debug an algorithm in the clear domain (either at the integer level or at the bit level)
2) characterize an algorithm so as to both obtain dimensioning parameters for the underlying FHE scheme (e.g., the multiplicative depth of the algorithm) and predict performances
3) execute literally an algorithm in the encrypted domain
4) generate compilation data (e.g., the Boolean circuit topology) for further optimizations of the calculation and later executions on an ad hoc, nonliteral, execution support.

Again, this can be achieved by using the type parameterization feature of object-programming languages (such as the so-called templates provided in the C++ language) by creating an `integer` class parameterized by both a `bit` type and a size. The `bit` type representing either clear bits (in which case the operators + and * are trivial), instrumented clear bits (see `ClearBit` below) or cryptobits (in which case the + and * operators are implemented with respect to the underlying FHE scheme). As an example, in this framework, the bubble sort code sample of the previous section simply becomes

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

and this unique code is either invoked as

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

for execution in the clear to, e.g., sort an array (of public size) of 8-b integers or as

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

to do the same thing in the encrypted domain (of course in that case `arr` contains 8-b integers encrypted at the bit level with the underlying FHE scheme).

Since (as already emphasized) we are dealing only with programs with a static control structure, any execution in the clear domain allows to infer the relevant characteristics of an algorithm. For example, `ClearBit` objects can be instrumented so as to track the depth and multiplicative depth of each bit involved in the calculation. By depth of a bit, we mean, similarly to the circuit depth, the length of the longest path from the circuit inputs to the operator that computes the said bit. Straightforwardly, the depth of the result of either the `XOR`ing or the `and`ing of 2 b of depth $d_1$ and $d_2$ is $1+\max(d_1,d_2)$ and the multiplicative depth of the result of the `XOR`ing (respectively the `and`ing) of 2 b of multiplicative depth $d'_1$ and $d'_2$ is $\max(d'_1,d'_2)$ (respectively, $1+\max(d'_1,d'_2)$). The maximum depth and multiplicative depth can be tracked along an initial clear domain execution so as to dimension the number of levels of a BGV-style cryptosystem for later executions in the encrypted domain.

Also, the `ClearBit` objects can be instrumented so as to explicitly build the acyclic directed graph representing the Boolean circuit underlying the algorithm. This is a very convenient representation at least for two reasons. First, it reveals a high degree of parallelism, as the so-called equivalence classes with respect to a topological ordering of the graph vertices reveal (potentially) large sets of operators that can be performed in parallel, which is a good strategy to mitigate the performance hit of using homomorphic encryption. Second, this representation allows the performance of fine grain optimized scheduling of the calculations to maximize the efficiency of certain mechanisms such as the depth caching technique discussed in the next section.

## SOME PRELIMINARY EXPERIMENTAL RESULTS

We have developed a prototype of the compilation and execution infrastructure sketched in the previous section and (seamlessly) interfaced it with two somewhat fully homomorphic cryptosystem implementations: our own implementation of the vectorial flavor of the BGV cryptosystem and a public domain implementation of the Smart-Vercauteren cryptosystem [24] available at http://www.hcrypt.com.

Our prototype supports all the functions that have been presented in this section, including Boolean circuit generation and parallel execution.

As far as the implementation of the BGV cryptosystem is concerned, to avoid redundant level shifts (i.e., calls to `Rescale` on an $i$th level ciphertext when there already is an $i-1$th version of said ciphertext), we have implemented a depth caching technique whereby each `CryptoBit` object remembers all its different-level copies in a small associative data structure keyed by level. This technique results approximately in speedups of around 45%.

Table 1 provides characterization data for a number of elementary algorithms obtained using instrumented clear domain bit-level executions. For each algorithm, the number of bit-level additions (# add), number of bit-level multiplications (# mul), depth, multiplicative depth (× depth) as well as the average number of operations per topological equivalence classes of the underlying Boolean circuit (a number which gives an idea of the amount of circuit-level parallelism and is labeled "av. //") are given. The multiplicative depth is necessary to parametrize the BGV scheme (it tells how many levels we need to be able to handle). The other figures can be used to try to predict the performances of a homomorphic evaluation of these algorithms (or at least what we should expect about the level of performances).

**[TABLE 1] CHARACTERIZATION OF A FEW ELEMENTARY ALGORITHMS.**

| | $b^2-4ac$ **(8 b)** | $b^2-4ac$ **(16 b)** |
|---|---|---|
| # ADD | 332 | 1,188 |
| # MUL | 302 | 1,126 |
| DEPTH | 43 | 83 |
| × DEPTH | 16 | 32 |
| AV. // | 14.74 | 27.88 |
| | $\sum_{i=1}^{10} t[i]$ **(8 b)** | $\sum_{i=1}^{10} t[i]$ **(16 b)** |
| # ADD | 207 | 423 |
| # MUL | 135 | 279 |
| DEPTH | 24 | 48 |
| × DEPTH | 8 | 16 |
| AV. // | 6.75 | 14.62 |
| | **B. SORT (10 × 4 b)** | **B. SORT (10 × 8 b)** |
| # ADD | 1,620 | 3,240 |
| # MUL | 1,350 | 2,790 |
| DEPTH | 214 | 350 |
| × DEPTH | 68 | 136 |
| AV. // | 13.88 | 17.23 |
| | **FFT (256 × 32 b)** | |
| # ADD | 7,291,592 | |
| # MUL | 52,96,128 | |
| DEPTH | 674 | |
| × DEPTH | 166 | |
| AV. // | 18,676.10 | |

Parallelism is handled in two (so far exclusive) different ways, either internally to the cryptosystem or externally at the Boolean circuit level.

Internal parallelism is handled via an OpenMP parallel for pragma in the outer loop of the matrix product in `SwitchKey` (which as already emphasized is the main hot point, performancewise). This parallelization strategy results in further speedups of around 41% on an average dual core laptop and seems to be the optimal strategy for these kinds of machines.

Table 2 provides experimental results obtained on a laptop with a 2 GHz Intel dual core processor, using both the aforementioned depth cache and `SwitchKey` parallel `for`. The metrics given are the execution time ("CPU"), percentage of depth cache hits ("cache eff.") as well as the size of the overall public key ("pubk size"), which accounts for the size of the public keys of the cryptosystem at each level and the key switching matrices. Last, for the sake of completeness, Table 2 also presents the execution times we have obtained on the same set of elementary algorithms using the HCRYPT library (www.hcrypt.com) of Brenner et al.

**[TABLE 2] EXECUTION TIMES FOR A NUMBER OF ELEMENTARY ALGORITHMS ON AN AVERAGE DUAL CORE LAPTOP.**

| | $b^2-4ac$ **(8 b)** | $b^2-4ac$ **(16 b)** |
|---|---|---|
| CPU | 0.406 S | 4.124 S |
| CACHE EFF. | 46% | 40% |
| PUBK SIZE | 1.1 MB | 7.8 MB |
| HCRYPT | 58.9 S | 3 M 39 S |
| | $\sum_{i=1}^{10} t[i]$ **(8 b)** | $\sum_{i=1}^{10} t[i]$ **(16 b)** |
| CPU | 0.125 S | 0.562 S |
| CACHE EFF. | 47% | 47% |
| PUBK SIZE | 196 KB | 1.1 MB |
| HCRYPT | 27.2 S | 55.4 S |
| | **B. SORT (10 × 4 b)** | **B. SORT (10 × 8 b)** |
| CPU | 5.219 S | 18.110 S |
| CACHE EFF. | 64% | 64% |
| PUBK SIZE | 68.5 MB | 525 MB |
| HCRYPT | 5 M 5 S | 9 M 41 S |

External parallelism, i.e., parallelism at the Boolean circuit topological equivalence classes level, is intended to target the execution of heavier algorithms on higher-end multicore machines. Although we cannot report on a speedup measurement, this external parallelism strategy has allowed us to perform a full 32-b 256-point fast Fourier transform in less than four hours on an AMD-based NUMA machine with 48 cores, a calculation which otherwise appeared to be undoable in "nonprohibitive" time.

However, the reader should be warned that these results have been obtained using cryptosystem parameter values, which are presumably too small to provide a nontrivial level of security. They should thus be considered giving more of an optimistic lower bound on the level of performance that can be achieved using the BGV system rather than a conservative upper bound. In our opinion, despite the fact that BGV-style cryposystems enjoy very strong theoretical security properties, practical parameter setting for the BGV system as well as for its siblings is a question that still needs additional theoretical investigations. These figures are representative as they have been obtained with one of the first implementations of a full-blown fully homomorphic cryptosystem.

In addition to these results, we managed to execute the sum of ten 4-b elements over the variant of the BGV scheme of [18] with larger parameters. With an approximative 40-b security level, the sum of encrypted elements took about 1 min (without parallelization). For information, a 64-b security level is considered suitable for small attackers, 80-b is the smallest general-purpose protection, and 128-b is considered a long-term protection. Testing our implementation with a higher security level on various algorithms and developing compilation tools will be the subject of future work.

Finally, the execution times of HCRYPT have been obtained with default parameters (which are also too small to provide a nontrivial level of security), as the underlying FHE scheme in our system. Something we were able to do seamlessly (as soon as an HCRYPT-based `CryptoBit` class was implemented). Although the performances obtained with our implementation of BGV appear to be much better, we should still emphasize that these results are hard to compare to those of Table 2 for two reasons. First, the HCRYPT library implements the bootstrapping-based Smart-Vercauteren FHE scheme that is by no means a potentially nonprohibitive scheme. Second, we only have a limited understanding of the extent to which parallelism is used in that library (as well as its numerous dependencies).

## CONCLUSIONS

In this article, we have discussed a number of steps toward bridging the gap between nontrivial algorithms and their practical, relatively seamless, execution on (somewhat) FHE schemes. We have also provided some preliminary experimental results indicating that there is hope, in the near future, to be able to homomorphically execute simple algorithms on BGV-style cryptosystems in reasonable time.

Still, we have shown that the level of performances achieved is still far from enabling the execution of more computationally involved algorithms in nonprohibitive time.

Despite this, there is hope in the sense that theoretical progress has been fast-paced since 2009 (with the theoretical overhead decreasing by an order of $\sqrt{\lambda}$ every year or so, $\lambda$ being the security parameter) and that research work on algorithm "FHE-friendlyness," on compilation (in the wide sense) as well as on ad hoc optimized (parallel) execution supports for these cryptosystems is only just beginning. These latter fields of research, as we have hinted in this article, can be expected to contribute significantly to the performance improvements required to make homomorphic encryption-based computations a practical reality, particularly in the field of signal processing.

**THERE IS HOPE, IN THE NEAR FUTURE, TO BE ABLE TO HOMOMORPHICALLY EXECUTE SIMPLE ALGORITHMS ON BGV-STYLE CRYPTOSYSTEMS IN REASONABLE TIME.**


## ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for their valuable remarks and suggestions that led to improvements in the article.


## AUTHORS


*Carlos Aguilar-Melchor* (carlos.aguilar@xlim.fr) is a former student of the Ecole Polytechnique de Paris, France, and obtained his Ph.D. degree in 2006 from the LAAS-CNRS (Toulouse) under the supervision of Yves Deswarte. His research interests are security, privacy, codes, and cryptography. In security and privacy, he has worked mainly on anonymous

communications and on trust and replication issues on new generation networks. In codes and cryptography, he has worked on various privacy primitives such as private information retrieval, homomorphic encryption, and ring signatures and on the enumeration of self-dual codes. He is currently an assistant professor at the University of Limoges.

***Simon Fau*** (simon.fau@cea.fr) received his master's degree in cryptology at Paris Diderot University and then joined the Commissariat à l'Energie Atomique as a Ph.D. student to work on cryptocomputing, focusing on homomorphic encryption. His thesis is done in partnership between the Commissariat à l'Energie Atomique and the University of Bretagne-Sud.

***Caroline Fontaine*** (caroline.fontaine@telecom-bretagne.eu) is a full-time researcher at CNRS (French National Research Institute). She has been working on content protection for more than 15 years. Her publications cover cryptography, steganography, digital watermarking, and active fingerprinting, with most of her articles tackling several of these domains at the same time. She has been and is involved in many research projects on these topics, and in the organization and program committees of many conferences and publications. She is with CNRS/Lab-STICC and Télécom Bretagne.

***Guy Gogniat*** (guy.gogniat@univ-ubs.fr) is a professor in electrical and computer engineering with the University of Bretagne-Sud, Lorient, France, where he has been since 1998. In 2005, he spent one year as an invited researcher with the University of Massachusetts, Amherst, where he worked on embedded system security using reconfigurable technologies. His work focuses on embedded systems design methodologies and tools. He also conducts research in the domain of reconfigurable and adaptive computing and embedded system security.

***Renaud Sirdey*** (renaud.sirdey@cea.fr) is a senior researcher at Commissariat à l'Energie Atomique, where he leads the Embedded Real-Time Systems Laboratory. His main research interests include parallelism, compilation, discrete optimization as well as applied cryptology. He has most notably led the research team that designed a complete industry-grade dataflow compiler for the 256 cores MPPA architecture as part of a joint CEA/KALRAY lab. His activities include fundamental research on both exact and approximate resolution of combinatorial optimization problems, optimization under uncertainty as well as parallel resolution of such problems. More recently, he has started to work on compilation and parallel execution supports for the currently emerging homomorphic encryption primitives.

## REFERENCES


[1] A. Adelsbach, M. Rohe, and A.-R. Sadeghi, "Non-interactive watermark detection for a correlation based watermarking scheme," in *Proc. 9th IFIP TC-6 TC-11 Int. Conf. CMS 2005 Communications and Multimedia Security*, LNCS 3677, pp. 129–139.

[2] C. Aguilar-Melchor, P. Gaborit, and J. Herranz, "Additively homomorphic encryption with *d*-operand multiplications," in *Proc. CRYPTO'10*, LNCS 6223, pp. 138–154.

[3] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Donida Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, A. Piva, and F. Scotti, "A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates," in *Proc. BTAS 2010 IEEE 4th Int. Conf. Biometrics: Theory, Applications and Systems*, 2010, pp. 15–21.

[4] M. Barni, P. Failla, R. Lazzeretti, A.-R. Sadeghi, and T. Scheider, "Privacy-preserving ECG classification with branching programs and neural networks," *IEEE Trans. Inform. Forensics Sec.*, vol. 6, no. 2, pp. 452–468, 2011.

[5] T. Bianchi, A. Piva, and M. Barni, "Encrypted domain DCT based on homomorphic cryptosystems," *EURASIP J. Inform. Sec.*, vol. 2009, Article ID 716357, 2009.

[6] T. Bianchi, A. Piva, and M. Barni, "Implementing the discrete Fourier transform in the encrypted domain," *IEEE Trans. Inform. Forensics Sec.*, vol. 4, no. 1, pp. 86–97, 2009.

[7] T. Bianchi, A. Piva, and M. Barni, "Composite signal representation for fast and storage-efficient processing of encrypted signals," *IEEE Trans. Inform. Forensics Sec.*, vol. 5, no. 1, pp. 180–187, 2010.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc. 3rd Innovations in Theoretical Computer Science Conf.*, 2012, pp. 309–325.

[9] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *Proc. 2011 IEEE 52nd Annu. Symp. on Foundations of Computer Science*, pp. 97–106. [Online]. Available: http://eprint.iacr.org/2011/344

[10] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Proc. Advances in Cryptology, (CRYPTO 2011),* vol. 6841, p. 501.

[11] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, R. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Proc. 9th Int. Symp. Privacy Enhancing Tecnologies*, 2009, pp. 235–253.

[12] Z. Erkin, A. Piva, S. Katzenbeisser, R. Lagendijk, J. Shokrollahi, G. Neven, and M. Barni, "Protection and retrieval of encrypted multimedia content: When cryptography meets signal processing," *EURASIP J. Inform. Sec.*, Article ID 0313402007.

[13] P. Failla and M. Barni, "Gram-Schmidt orthogonalization on encrypted vectors," in *Proc. 21st Int. Tyrrhenian Workshop Digital Communications, (ITWDC),* Ponza, Italy, 6–8 Sept. 2010.

[14] P. Failla, Y. Sutcu, and M. Barni, "eSketch: A privacy-preserving fuzzy commitment scheme for authentication using encrypted biometrics," in *Proc. 12th ACM Multimedia and Security Workshop,* Rome, Italy, 9–10 Sept. 2010.

[15] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," *EURASIP J. Inform. Sec.*, vol. 2007, no. 1, pp. 1–15, 2007.

[16] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. STOC'09*, pp. 169–178.

[17] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Proc. EUROCRYPT*, 2011, pp. 129–148.

[18] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," *CRYPTO 2012,* (LNCS) vol. 7417, 2012, p. 850–867, 2012.

[19] C. Gentry, S. Halevi, and V. Vaikuntanathan, "A simple BGN-type cryptosystem from LWE," in *Proc. EUROCRYPT'2010* (LNCS) vol. 6110, pp. 506–522.

[20] S. Katzenbeisser, A. Lemma, M. Celik, M. van der Veen, and M. Maas, "A buyer–seller watermarking protocol based on secure embedding," *IEEE Trans. Inform. Forensics Sec.*, vol. 3, no. 4, pp. 783–786, 2008.

[21] D. E. Knuth, *The Art of Computer Programming*, vol. III, *Sorting and Searching*. Reading, MA: Addison-Wesley, 1973.

[22] M. Kuribayashi and H. Tanaka, "Fingerprinting protocol for images based on additive homomorphic property," *IEEE Trans. Image Processing*, vol. 14, no. 12, pp. 2129–2139, 2005.

[23] R. Lagendijk, Z. Erkin, and M. Barni, "Encrypted signal processing for privacy protection," *IEEE Signal Processing Mag.,* vol. 30, no. 1, pp. 82–105, 2013.

[24] H. Perl, M. Brenner, and M. Smith, "Poster: An implementation of the fully homomorphic Smart-Vercauteren crypto-system," in *Proc. ACM Conf. Computer and Communications Security*, 2011, pp. 837–840.

[25] A. Piva, V. Cappellini, D. Corazzi, A. D. Rosa, C. Orlandi, and M. Barni, "Zero-knowledge ST-DM watermarking," in *Proc. IS&T/SPIE Int. Symp. Electronic Imaging 2006*, San Jose, CA, 16–19 Jan. 2006.

[26] J. P. Prins, Z. Erkin, and R. Lagendijk, "Anonymous fingerprinting with robust QIM watermarking techniques," *EURASIP J. Inform. Sec.*, 2007.

[27] O. Regev, "The learning with errors problem (invited survey)," in *Proc. IEEE Conf. Computational Complexity*, 10 Dec. 2010, pp. 191–204.

[28] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations of Secure Computation,* R. Demillo, D. Dobkin, A. Jones, and R. Lipton, Eds. New York: Academic, 1978, pp. 169–180.

[29] J. R. Troncoso-Pastoriza and F. Pérez-González, "Zero-knowledge watermark detector robust to sensitivity attacks," in *Proc. ACM Multimedia and Security (MM&SEC 2006),* pp. 97–107.

[30] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *EUROCRYPT'2010* (LNCS) vol. 6110, pp. 24–43.


[SP]