Abstract: As stated by NIST in "Guidelines on Security and Privacy in Public Cloud Computing", lowering costs for computing resources made Cloud Computing attractive for different organizations and users. Modern software applications that run on desktops, Cloud or mobile device ecosystems are exposed to many security threats. For instance, there are security issues that affect the processing of confidential data in the Cloud because the providers of Cloud services have full access to the hardware, storage, software and as a result, to confidential data. Furthermore malicious software can be ran in the Cloud or and attack other Cloud services to retrieve confidential information. Mainly the mobile device ecosystem is vulnerable because the digital assets and confidential data they contain represent an attractive target for the attackers. Assuming fully homomorphic encryption schemes, which allow computing any function over encrypted data, will become fast enough to be considered practical, privacy concerns with respect to desktop, Cloud or mobile services could be solved.

Key-Words: cloud computing, homomorphic encryption, partially homomorphic encryption, fully homomorphic encryption, homomorphic operation, voting scheme, private information retrieval.

1. Introduction

Modern technologies help users and companies to connect and share things with other users and co-workers no matter the location. But these connections can bring huge costs if the users aren’t careful enough. During learning how to protect personal identity information, creating strong passwords is crucial for the safety and security of the digital devices users interact with, and so is the information that those devices store. Otherwise, users can expose themselves to digital threats like computer viruses, data and identity theft, and hacking.

To create an analogy, consider the following statement: "Encrypting a document is like placing it inside of a locked safe". The locked safe is a good example because cryptography and physical safes have the same purpose: they ensure the confidentiality of sensitive document information.

In practice, they also share many drawbacks. Users tend to remove documents they need from the safe storage at first chance. This exposes them to all the usual threats and explains why so few cases of document theft involve safecracking. The same principle holds for encryption: users decrypt their data in order to be able to use it.

The good part is that there exists a kind of encryption that allows us to bypass some of the limitations presented before. We refer to this as homomorphic encryption. Evolving from a concept called privacy homomorphism [30] (1978, Rivest, Adleman and Dertouzos), homomorphic encryption gained attention by offering the possibility of manipulating ciphertexts without the need of decrypting them. The terminology is closely related to algebraic homomorphisms. In fact, the encryption function of a cryptosystem
with the above mentioned property is a homomorphism. We give a short survey of homomorphic encryption schemes within Section 2, discuss possible applications of Partially and Fully Homomorphic Encryption (PHE and FHE) in Section 3 and conclude in Section 4. As cloud computing could become the main application of homomorphic encryption schemes, within Section 3 we will mostly tackle this area. We will also give short descriptions of electronic voting schemes and private information retrieval.

2. PHE and FHE

Any homomorphic encryption scheme is malleable by default. First mentioned in 1982 by Goldwasser and Micali, the definition of semantic security changed as time passed and became equivalent to ciphertext indistinguishability (IND-CPA). This form of security is said to be the weakest for a cryptosystem. A less weak form is IND-CCA and even a stronger one is IND-CCA2. Some malleable cryptosystems may be IND-CCA, but it was stated that the property of IND-CCA2 is equivalent to non-malleability (in 2011, the notion “CCA-embeddable homomorphic encryption” was introduced by Loftus, May, Smart and Vercauteren [27]). The next described partially and fully homomorphic encryption schemes are at least IND-CPA secure (only unpadded RSA is not semantically secure).

2.1 Partially Homomorphic Encryption Schemes

Except Boneh-Goh-Nissim cryptosystem (which allows a random number of additions and a single multiplication), a PHE scheme usually allows only one type of homomorphic operation. We will refer to (unpadded) RSA and El Gamal only from the homomorphic encryption point of view. We stress that Goldwasser-Micali [10] was the first provably-secure probabilistic public-key encryption scheme.

2.1.1 Unpadded RSA and El Gamal

The homomorphic operation provided by RSA is the multiplication of two messages modulo n: given two messages $m_1$ and $m_2$ and their encryptions $\varepsilon(m_i)_{i=1,2}$, it is very easy to prove that $\varepsilon(m_1)\varepsilon(m_2) = \varepsilon(m_1m_2)$. Rarely used for its homomorphic properties (as it is not IND-CPA secure for unpadded messages), this cryptosystem can be made semantically secure using random encryption padding schemes like OAEP [14].

El Gamal cryptosystem mainly provides the multiplication of two messages. Let $m_1$ and $m_2$ two messages and $\varepsilon(m_i)_{i=1,2}$ be their encryptions using El Gamal. Then, $\varepsilon(m_1)\varepsilon(m_2) = \varepsilon(m_1m_2)$.

2.1.2 Naccache-Stern (NS)

Seen as a corrected generalization of Benaloh scheme [10], the semantic security of Naccache-Stern cryptosystem relies on the higher residuosity problem. NS has been first defined as a deterministic cryptosystem, but it was shown that with a small number of modifications it can be made probabilistic. We chose the presentation of the probabilistic version. We first describe the key generation and message encryption. Let $p_1, \ldots, p_3$ be a family of distinct primes. Let $\prod_{i}^{k/2} p_i$ and $\prod_{i}^{k} p_i$. Set $\sigma = uv = \prod_{i}^{k} p_i$.

Choose large primes $a$ and $b$ s.t. both $p = 2au + 1$ and $q = 2bv + 1$ are prime. Set $n = pq$. Choose a random $g$ mod $n$ s.t. $g$ has $\varphi(n)$ order 4. The public key is $(\sigma, n, g)$ and the private key $(p, q)$.

The encryption of a message $m$ in $\mathbb{Z}_\sigma$:

Pick a random $x \in \mathbb{Z}_n$.

Compute $\varepsilon(m) = x^\sigma g^m$ mod $n$. 
We further present the way decryption works. Given a ciphertext \( c \) we compute
\[
C_i \equiv c^{m_i^{-1}} \mod n,
\]
where \( m_i \equiv m \mod p_i \). Comparing \( C_i \) to \( g^{j \cdot \varphi(n)} \) for \( j = 1, p_i - 1 \), \( m_i \) can be recovered. With a direct application of CRT, \( m \) can be found. 

The NS cryptosystem provides the homomorphic operation of addition (and the multiplication of a ciphertext by a constant).

Let \( m_1 \) and \( m_2 \) be two messages and \( \varepsilon(m_1) \) and \( \varepsilon(m_2) \) their encryptions. Then, \( \varepsilon(m_1) \cdot \varepsilon(m_2) = x_1^x_2 g^{m_1} x_2^x_2 g^{m_2} \mod n = (x_1 x_2)^x g^{m_1 + m_2} \mod n \). The obtained result is a valid encryption for the message \( m_1 + m_2 \).

### 2.1.3 Paillier

Seen as an extension of the Okamoto-Uchiyama cryptosystem [6], Paillier scheme is a research result regarding trapdoor discrete logarithms-based cryptosystems. The IND-CPA security of Paillier relies on computing composite residuosity problem. Basic ideas of the key generation and message encryption algorithms are given next.

Choose two large prime numbers \( p \) and \( q \) (randomly and independently of each other) s.t. \( \gcd(pq, (p - 1)(q - 1)) = 1 \).

Compute \( n = pq \) and \( \lambda = \text{LCM}(p - 1, q - 1) \).

Select a random integer \( g \in \mathbb{Z}_{n^2}^* \).

Let \( L \) be a function defined as \( L(u) = \frac{u - 1}{n} \).

Ensure that \( n \) divides the order of \( g \) by checking the existence of \( \mu = (L(g^2 \mod n^2))^{-1} \mod n \).

The public key is \( (n, g, h, G, G_1, e) \) and the private key \( (\lambda, \mu) \).

If the message to be sent is \( m \in \mathbb{Z}_n \), then:

Select a random \( r \in \mathbb{Z}_{n^2}^* \).

Compute \( \varepsilon_r(m) = g^{m r^2} \mod n^2 \).

In order to decrypt a message, compute \( m = L((\varepsilon_r(m))^2 \mod n^2) u \mod n \).

The Paillier cryptosystem provides the homomorphic operations of message addition and multiplication by a constant.

Given two messages \( m_1 \) and \( m_2 \) and their encryptions \( \varepsilon_r(m_1) \) and \( \varepsilon_r(m_2) \), we have:

\[
(1) \varepsilon_r(m_1) \cdot \varepsilon_r(m_2) = g^{m_1 r^2} g^{m_2 r^2} \mod n^2.
\]

### 2.1.4 Damgard-Jurik (DJ)

Paillier’s scheme can be considered as a special case of the Damgard-Jurik cryptosystem. Like Paillier, the security of DJ can be proven under the decisional composite residuosity assumption. DJ cryptosystem uses computations modulo \( n^s+1 \) (instead of \( n^2 \) for Paillier).

The homomorphic operation provided by DJ is the same as in Paillier cryptosystem: addition. The only difference is that here we make computations modulo \( n^{s+1} \).

### 2.1.5 Boneh-Goh-Nissim (BGN)

BGN is the first scheme that can handle (in terms of homomorphic encryption) an arbitrary number of additions and one multiplication. A very important ingredient of the BGN cryptosystem is the use of bilinear pairings [15].

We give a short description with respect to key generation and message encryption.

Let \( G \) and \( G_1 \) be two multiplicative groups of order \( n \), with a bilinear pairing \( e: G \times G \rightarrow G_1 \).

Choose two distinct primes \( p \) and \( q \), set \( n = pq \) and select a positive integer \( T < q \).

Choose \( g \) and \( u \) two random generators of \( G \) and set \( h = u^q \) is a generator of the order \( p \) subgroup.

The public key is \( (n, g, h, G, G_1, e) \) and the private key \( p \).

Given a message \( m \in \mathbb{Z}_T \) and a random \( r \in \mathbb{Z}_n \), the encryption of \( m \) is as follows:

\[
\hat{1}_r(m) = g^m h^r.
\]
For decryption, compute  
\[ c = \left( e \cdot \frac{I(m)}{m} \right)^p \bmod n \] and use Pollard’s lambda method \[19\] to take the discrete logarithm of \( c \) in base \( g^p \).

BGN inherits the same homomorphic properties as Paillier and Okamoto-Uchiyama cryptosystems. Besides additive operations, BGN allows one multiplication.

2.1.6 PHE Schemes Efficiency
It is already known that NS cryptosystem implies a smaller message expansion than the Benaloh cryptosystem. The message expansion rate of NS is \( N/Q \), where \( N \) is the bit-length of and the bit-length of (using the same notations as in section 2.1.2). For the cryptosystem to remain secure, the lower bound of this rate must be 4. On the other hand, NS hasn’t been considered as attractive as Okamoto-Uchiyama (developed at the same time). The last mentioned cryptosystem is easier to implement and has a fixed expansion rate of 3.

Researchers where interested in lowering the expansion rate (but without minimizing security properties). We can observe that Paillier cryptosystem allows encrypting many bits during a single operation with a constant expansion factor of 2. Moreover, Paillier allows efficient decryption.

DJ cryptosystem has been proven to be as secure as Paillier’s original scheme. But, this generalization of Paillier allows reducing the expansion factor to almost 1.

Assuming the same security parameter \( k \), a comparison of DJ, Paillier, RSA (with public exponent \( 2^{16}+1 \)) and El Gamal can be found in \[5\].

The message expansion rate of BGN cryptosystem is \( N/R \), where \( N \) is the bit-length of \( n \) and \( R \) the bit-length of \( r \) (using the same notations as in section 2.1.5).

2.2 Fully Homomorphic Encryption Schemes
As Craig Gentry mentioned in \[5\], the appearance of FHE schemes solve a central problem in cryptography.

Such a scheme should allow one to compute arbitrary functions over encrypted data without the decryption key.

A FHE scheme consists of four algorithms: KeyGen (key generation), Encrypt (message encryption), Decrypt (decryption) and Evaluate (homomorphic evaluation).

Consider a boolean circuit \( f: \{0,1\}^h \rightarrow \{0,1\} \), \( h \) inputs \( m_1, ..., m_h \in \{0,1\} \), a pair of keys \( (pk, sk) \) and ciphertexts \( c_i = \text{Encrypt}_{pk}(m_i) \), \( i = 1, h \).

If \( \text{Decrypt}_{sk}(\text{Evaluate}(f,c_1, ..., c_h)) \) = \( f(m_1, ..., m_h) \), the scheme is said to be fully homomorphic.

2.2.1 Evolution of FHE Schemes
In 2009, Gentry published a first FHE scheme. His system was able of evaluating an arbitrary number of additions and multiplications on encrypted data (therefore, computing “any” function). Using lattice-based cryptography, his first proposal was rather theoretical than implementable.

Thus, in December 2009, he described within a paper written together with van Dijk, Halevi and Vaikuntanathan, a simpler scheme (DGHV) than his first one, using integers instead of lattices. In 2010, new versions working implementations of FHE schemes appeared. A new scheme based on RLWE (ring learning with errors) problem was presented (by Brakerski and Vaikuntanathan) and implemented in 2011 (by Lauter, Naehrig and Vaikuntanathan). Also, in 2011, Coron, Naccache and Tibouchi described a compression technique for reducing the public key size of DGHV scheme and Gentry, Halevi and Smart came up with an improvement of Gentry’s bootstrapping technique.

We further present the basic ideas of DGHV scheme.
First, a useful Somewhat Homomorphic Encryption Scheme is constructed. After that, the scheme is modified to become bootstrappable, therefore, a FHE one. The somewhat homomorphic encryption scheme is presented next.

The secret key $\langle sk \rangle$ consists of an odd $p$. The public key $\langle pk \rangle$ consists of “many encryptions of 0”: $x_i = [q, p + 2r]_{x_0}, i = 1, \ldots, n$. 

$Enc_{pk}(m) = [\text{subset sum}(x_i) + m + 2r]_{x_0}$, for a message $m$. 

$Dec_{sk}(c) = (c \mod p) \mod 2$, for a ciphertext $c$.

An idea used in server-aided cryptography is used for making the above scheme bootstrappable.

### 3. Applied PHE and FHE

Cloud computing’s ways of usage in which both company and individual users run their own or a rent application on shared data centers seem limitless. But reality shows they are not. One of cloud computing’s constraints is that encrypted information can’t be processed within the cloud because currently there is no way to handle the data in a safe way once it is opened for computations. This is the area where HE should enter and fulfill the need for security and privacy.

Homomorphic cryptosystems may be used for creating secure voting schemes, CHRF’s (collision-resistant hash functions) and PIR (private information retrieval) schemes.

For more than 30 years the existence and practicality of FHE schemes were strongly questioned (together with the practicality of PHE schemes). The basis for practical applications is already shown and we have the ability to build upon it. Users’ reactions are somehow divided. On one hand the standard reaction is to have “something” that will keep data safe, but on the other hand the idea that movies, music or e-books can be encrypted, by-pass existing content filters and end on P2P networks like Bittorrent is not so appealing.

Also virtual environments forensics must be designed to deal with HE because it is harder to prove that something bad or illegal happened when everything is encrypted.

As already specified within this paper’s abstract, cloud computing is another fitting area for homomorphic encryption. Voting schemes, PIR and, especially, cloud computing will be described next.

### 3.1 Cloud Computing

In the context of cloud computing, data must be sent to a certain cloud provider. For example, if the users send data to an online storage service, the data can be encrypted before sending it. In this case, data privacy is assured and the provider can neither use nor analyze the data.

However, analyzing current implementations of cloud computing services like Software-as-a-Service or even Infrastructure-as-a-Service, data can be encrypted during the transfer phase but it must arrive at destination as plaintext. More precisely, in order for a virtual machine to properly work the user must make sure that the code is not encrypted so it can be executed.

For now, both users and companies that use cloud computing services have to trust the service provider they choose. Techniques for encrypting a virtual machine and their attached disk volumes like [20] and [22] exist, but they only represent partial solutions.

The main disadvantage of the solutions mentioned before is that they require the decryption key to be transmitted at a given moment, typically when booting up the virtual machine. For short, we can say that the data is not encrypted. This is a problem because it slows down the wide adoption of cloud services because of the lack of security for the most sensitive data.

As presented in our paper, HE is a process by which complex calculations can be performed on data, without knowing where it is stored and that it is encrypted. This addresses the need to have “everything
encrypted all the time”, both in data transfer and data usage.

On the 8th of August 2011, Microsoft’s research laboratory for cloud cryptography [18] announced a significant advance towards this direction in their work [23] by implementing a prototype storage system for cloud platforms.

Nevertheless, we are far from being able to run a virtual machine in a cloud environment using fully homomorphic encryption, but we think that the first practical applications will be implemented on Software-as-a-Service or Platform-as-a-Service for handling certain specific data types.

A question naturally rises: “Why should we care about this?” The answers are long and we can find a lot of them in different places, but the main idea is the same in all. Data leaks and breaches made by malicious people with the intent of stealing sensitive information are and will always be a great concern for users. With the help of HE this information can always be encrypted and it is useless for anybody else except the legit owner of it.

Of course, the normal evolution of this led to the banking area. Does it make sense to use HE with cloud payment-card transactions? This is not such a good case because the cloud system does no actual data processing on credit card transaction data.

A more realistic scenario sounds like this. Consider the existence of a software-as-a-service application provided by some company that runs a tax preparer algorithm. Basically the users give out personal information, credit card information and the service provider runs some kind of algorithm to optimize the tax and finance strategy.

But can we really store the bank account number and other balances to the cloud in a reliable and secure way? What if users could have the possibility to give the application a key by which it can download homomorphically encrypted data from the bank and run the proprietary algorithms over it and finally give to the users a cipher text that only the initial user can decrypt and analyse?

Cloud services are now used for different kinds of computing and business software.

A private cloud medical records storage system was proposed in 2009 by Chase, Lauter, Benaloh and Horvitz [3]. The goals of their PCE (patient controlled encryption) proposal are the guarantee of strong security and the maintained functionality (private key and also symmetric key schemes are presented).

In their proof of concept, a somewhat HE scheme is used. They stated that the data can only "escape" in encrypted form and for attackers is nearly impossible to decode it without having the patient user key. This means a medical service that calculates predictions or issues warnings based on encrypted data received from medical equipment is constructible.

Other applications that require a somewhat homomorphic encryption scheme belong to financial and advertising domains.

Newer projects like CryptDB [11][28] are trying to implement HE on part of their system. CryptDB is designed to offer confidentiality and reliability in the face of SQL databases oriented attacks.

It works by executing SQL queries over encrypted data using a set of SQL-aware encryption schemes.

Current FHE schemes are too slow to be considered practical, but their fast development seems promising. We can even find open-source FHE/PHE like the hCrypt Project [20] and Coron [15].

### 3.2 Voting Schemes

The main problem of the current electronic voting schemes is their requirement: a large amount of trust from the election officials and the people that actually vote. The solution found in this case was the use of specific cryptographic schemes. The problem of this approach is that cryptography is added as an additional layer, rather than an integral part of the voting system.

The modern electronic voting schemes are an application of cryptography. Online voting (over the Internet) can be
very profitable because the users have the possibility to vote independently of their location. As a result, the number of participant voters will increase and this will have a great impact on our society. Furthermore, elections can be held more often in order to permit users express their opinion for certain problems. Electronic voting schemes have been studied over the last twenty years. Secure electronic voting systems need malleability (ensuring voters’ privacy is very important). It is already possible with end-to-end voting systems like Helios [17] to publicly store vote ballots in the cloud in an encrypted way, so that the public users can add them up to confirm the totals. Also they can check that their own vote was indeed included, without having the doubt that a user can sell his vote. This is a great deal for low-risk private elections, but it is risky for government elections. Since only vote addition is required a PHE scheme is suitable.

For example, in additive homomorphic encryption the product of two ciphertexts is a third ciphertext that encrypts the sum of the two original plaintexts. Voting applications can use additive homomorphisms to allow counting to be done before decryption. With other forms of encryption all the data ballots are separated from their identifying pieces of information and then decrypted and counted. If HE is used, the counting can be done while the votes are still encrypted and the final total can then be decrypted. This process hides the content of the original ballots. It is risky not because of any weakness in the HE scheme. The problem in this case is that the desktop client used to connect to the voting system can be compromised by malware, viruses or Trojans, thereby changing the vote before it is encrypted.

Some of the partially homomorphic encryption schemes may be used in this area. One of them is Paillier cryptosystem (Paillier and blind signatures are mentioned by Rivest [19] for creating a voting scheme).

We describe the basic steps of an electronic voting model using Paillier. Consider voters. Assume that a voter can give only a binary vote (of either 1 or 0). Encrypt votes before casting them. Decrypt the product of the encrypted votes to the value \( n \), the sum of all votes (\( n \) people voted for and \( m-n \) voted against).

Other PHE scheme that could be used for electronic voting are El Gamal and Damgard-Jurik. A homomorphic voter-verifiable election scheme is described in [3]. In [5] Damgard and Jurik have shown how (generalized) Paillier can be efficiently used as an electronic voting scheme. They pointed out that when it comes to large scale elections, Paillier is a better scheme than El Gamal. Also, they provided the reader with an efficient proof of validity of a vote and an efficient threshold version of the scheme.

### 3.3 Private Information Retrieval

As stated in [32], an important application of homomorphic encryption comes from the work of Kushilevitz and Ostrovsky [25] who showed how to construct single server PIR protocols with sub-linear communication, from any additively HE scheme.

In cryptography, PIR permits users to get certain data from a database stored on the network or Internet without revealing what he actually needs. The receiver wants the \( i^{th} \) value from the database, but the sender must learn nothing about \( i \).

For example, a common situation that can be found nowadays is the one in which a certain user is interested in watching a movie via pay-per-view from a certain supplier. The problem that arises is the user refusing to purchase the video stream because of its conviction that the supplier will sell his information to groups that are paying for such contact information of all the people who purchased that movie. Moreover, the association of the user’s identity to that
purchase can increase the time he spends cleaning spam messages out of his mailbox. In this particular case, PIR can allow the user to purchase the video stream without the supplier knowing which feed he actually got.

We have a couple of solutions for this. A naive solution to this problem is to send the entire database to the user, but this approach is not computational and communicational efficient because we must send $n$ communication bits. Another solution can be obtained by masking the $i^{th}$ value with additional random indices. This has the drawback of giving a lot of information about $i$.

The construction of such a PIR scheme starts by simply writing out the truth-table of a function, for example $n$-bit input and 1 bit output, as a database. The ciphertext of a message $m \in \{0,1\}^n$ is computed as the query of the PIR user for the $m^{th}$ entry of the truth table, and homomorphic evaluation is simply running the PIR protocol to retrieve $m^{th}$ entry. Since the query of the PIR user hides $m$, the scheme is semantically secure. Furthermore, the length of the ciphertext after homomorphic evaluation is exactly the communication complexity of the PIR protocol which is sub-linear or even logarithmic [1], [2], [8], [9].

As Gahi et al state in their work [7] cloud computing emerges as a solution for storing and processing confidential data. However, a cloud user cannot have the guarantee that the data is processed in a legal way. Also, Chow et al [4] discussed this issue and presented it from the final user’s perspective. Even this technology is used, personal and sensitive data is not uploaded and certain guarantees must be given.

4. Conclusion

Our survey consisting of homomorphic encryption schemes offers the reader some ideas regarding the development of cryptographic requirements with respect to this topic. While cloud computing revives researchers’ interest in creating practical FHE schemes (for computing any function on encrypted data), somewhat homomorphic encryption schemes are being used instead.

Right now, offloading computations to the cloud is pure fantasy. The most efficient FHE scheme currently known is still very expensive and the “arbitrary computation” part involves representing the computation as a circuit where each logic gate is emulated through its own HE. We are not talking about a 10 times slowdown within this process, we are talking about the whole Amazon EC2 cloud not being able, in a day, to perform homomorphically a computation which would take one second on a regular computer.

Performing computations using FHE nowadays takes quite a long time, but as techniques evolve things will quickly change.

Researchers believe in the possibility of advancing in FHE area and bringing new related technologies to the wide market. In such a context, HE will start having more and more applications far beyond the cloud area: it can be used whenever the need of doing computations on pieces of un-owned information appears.

References
[5] Damgard I. B. and Jurik M. J., A Generalisation, a Simplification and


[16] https://github.com/coron/fhe


[22] http://www.saout.de/misc/dm-crypt/


