KV Web Security: Applications of Homomorphic Encryption

Gerhard Pötzelsberger, B.Eng.

May 23, 2013

Contents

1 Introduction

Homomorphic encryption is a concept where specific computations can be performed on the ciphertext of a message. The result of these computations is the same as if the operations were performed on the plaintext first and encrypted afterwards. So homomorphic encryption allows parties who do not have an decryption key and thus don't know the plaintext value, still perform computation on this value.

Mathematically spoken is a *homomorphic cryptosystem* a cryptosystem whose encryption function is a homomorphism, and thus preserves group operations performed on ciphertexts. The two group operations are the arithmetic addition and multiplication. A homomorphic encryption is additive if:

$$\mathcal{E}(x + y) = \mathcal{E}(x) \otimes \mathcal{E}(y)$$

where $\mathcal{E}$ denotes an encryption function, $\otimes$ denotes an operation depending on the used cipher and x and y are plaintext messages. A homomorphic encryption is multiplicative if:

$$\mathcal{E}(x \cdot y) = \mathcal{E}(x) \otimes \mathcal{E}(y)$$

where again $\mathcal{E}$ denotes an encryption function, $\otimes$ denotes an operation depending on the used cipher and x and y are plaintext messages.

There are two different types of homomorphic cryptosystems, *fully homomorphic* cryptosystems and *partially homomorphic* cryptosystems. Partially homomorphic cryptosystems are defined over a group and therefore support just a single operation on the ciphertext. This operation can either be an addition or a multiplication. Fully homomorphic cryptosystems are defined over a ring and therefore support both operations, addition and multiplication.

The idea of computing with encrypted data is as old as public key cryptography. It was first proposed in 1978 by Ron Rivest, Len Adleman and Michael L. Dertouzos [15], who all worked at the MIT at this time. Just a few months earlier, Rivest, Adleman and Adi Shamir had introduced the first implementation of a public-key cryptosystem, which is today known as RSA.

Rivest, Adleman and Dertouzos referred to computations on ciphertexts as privacy homomorphisms. The original motivation for privacy homomorphisms was to allow for an encrypted database to be stored by an untrusted third party, while still allowing the owner to perform simple modifications such as updates and queries. Nothing about the database contents is revealed to the third party. In the conclusion of their paper, the authors outlined their optimism about finding useful privacy homomorphisms which offer enough operations so the above task could be accomplished.

The first homomorphic encryption schemes that were discovered were all partially homomorphic. As mentioned above RSA was the first one and it is multiplicative homomorphic when it is applied without padding. An example for an additive homomorphic cryptosystem is the GoldwasserMicali cryptosystem [11] which was proposed in 1982 or the Paillier cryptosystem [14] which

was proposed in 1999. A few years later, progress towards a fully homomorphic cryptosystem was made. In 2005 Dan Boneh, Eu-Jin Goh and Kobbi Nissim [4] devised a homomorphic cryptosystem that allowed an arbitrary number of additions and a single multiplication on the ciphertext. Finally in 2009 IBM researcher Craig Gentry [9] has published the first fully homomorphic encryption scheme which allows an arbitrary number of additions and multiplications on the ciphertext. So with the scheme Gentry proposed any type of function can be calculated on encrypted data.

In this work I will describe in which application scenarios researchers are trying to make use of homomorphic encryption and what is already used in practice.

2 Usage Scenarios

The demand for privacy of digital data is getting stronger as more and more devices are using services that are offered over the Internet. Mostly this involves communicating with a untrusted third party. Cryptosystems with the ability to compute on encrypted data can help to implement services where the privacy of a user is ensured despite the fact that private data is used.

This section will discuss possible applications of the homomorphic encryption in the fields of cloud services, e-voting, data mining and biometric authentication.

2.1 Cloud Services

Since cloud storage and cloud computing platforms were developed, users have the ability to outsource storage and computations on their data. Further it allows businesses to offload the task of maintaining a data-center. However, consumers and businesses are concerned about a possible loss of privacy which leads to an slow adoption of cloud services. The privacy concerns can be mitigated if users encrypt the data they send to the cloud. If the used encryption scheme is homomorphic, the cloud can still perform meaningful computations on the encrypted data. As learned in Section 1 a fully homomorphic encryption scheme is needed to perform arbitrary computation.

The main issue in this context is the question if fully homomorphic encryption schemes are efficient enough to be practical for cloud computing. Craig Gentry estimated in an article [7] that performing a Google search with encrypted keywords would multiply the necessary computing time by around 1 trillion. A more scientific analysis of Gentry's fully homomorphic encryption system was done for example in [10], but Gentry's estimation should make clear that the performance penalty of this scheme is way to big to use it in practice.

Kristin Lauter, Michael Naehrig and Vinod Vaikuntanathan argued in their paper [13] that there are many functions which could be useful for privacy preserving cloud services, which can be computed by many additions and a small number of multiplications on ciphertexts. For example, averages require no

multiplications, standard deviation requires one multiplication, and predictive analysis such as logistical regression requires a few multiplications.

There is a class of homomorphic cryptosystems that satisfy exactly that need, it is called *somewhat homomorphic cryptosystems*. Somewhat homomorphic cryptosystems, which support a limited number of homomorphic operations, are building blocks for fully homomorphic cryptosystems and provide much more efficiency and shorter ciphertexts than their fully homomorphic counterparts. A recent solution for such a scheme was introduced for example by Brakerski and Vaikuntanathan [5].

Lauter, Baehrig and Vaikuntanathan [13] provided a few concrete applications where customer privacy is preserved while various kinds of computation can be outsourced to the cloud. Namely, they detailed the medical, financial, and advertising sectors which I will repeat here. There were two aspects of the computation to consider: the data itself, and the function to be computed on this data. They considered cases where one or both of these are private or proprietary and should not be shared with the cloud. All data that is encrypted and sent to the cloud is public-key encrypted with the content-owner's public key, using a semantically secure somewhat homomorphic encryption scheme. Lauter, Baehrig and Vaikuntanathan described the three sectors as follows:

2.1.1 Medical Applications: Private data and Public functions

In [3], a private cloud medical records storage system (Patient Controlled Encryption) was proposed, in which all data for a patient's medical record is encrypted by the healthcare providers before being uploaded to the patient's record in the cloud storage system. The patient controls sharing and access to the record by sharing secret keys with specific providers (features include a hierarchical structure of the record, ability to search the encrypted data, and various choices for how to handle key distribution). However this system does not provide for the cloud to do any computation other than search (exact keyword match, or possibly conjunctive searches). With with homomorphic encryption, we can add the ability for the cloud to do computation on the encrypted data on behalf of the patient. Imagine a future where monitors or other devices may be constantly streaming data on behalf of the patient to the cloud. With homomorphic encryption, the cloud can compute functions on the encrypted data and send the patient updates, alerts, or recommendations based on the received data.

The functions to be computed in this scenario may include averages, standard deviations or other statistical functions such as logistical regression which can help predict the likelihood of certain dangerous health episodes. Encrypted input to the functions could include blood pressure or heart monitor or blood sugar readings, for example, along with information about the patient such as age, weight, gender, and other risk factors. The functions computed may not need to be private in this case since they may be a matter of public health and thus public.

2.1.2 Financial Applications: Private data and Private functions

In the financial industry there is a potential application scenario in which both the data and the function to be computed on the data is private and proprietary. As an example, data about corporations, their stock price or their performance or inventory is often relevant to making investment decisions. Data may even be streamed on a continuous basis reflecting the most up-to-date information necessary for making decisions for trading purposes. Functions which do computations on this data may be proprietary, based on new predictive models for stock price performance and these models may be the product of costly research done by financial analysts, so a company may want to keep these models private to preserve their advantage and their investment.

With homomorphic encryption, some functions can be evaluated privately as follows. The customer uploads an encrypted version of the function to the cloud, for example a program where some of the evaluations involve encrypted inputs which are specified. The streaming data is encrypted to the customers public key and uploaded to the cloud. The cloud service evaluates the private function by applying the encrypted description of the program to the encrypted inputs it receives. After processing, the cloud returns the encrypted output to the customer.

2.1.3 Advertising and Pricing

Imagine an advertiser, for example a cosmetics company, who wants to use contextual information to target advertising to potential customers. The consumer uses a mobile phone as a computing device, and the device constantly uploads contextual information about the consumer, including location, the time of day, information from email or browsing activity such as keywords from email or browser searches. In the future, imagine that information is uploaded potentially constantly from video devices: either pictures of objects of interest such as brands or faces which are automatically identified, or from a video stream from a camera on the body which is identifying context in the room (objects, people, workplace vs. home vs. store). When contextual information is uploaded to the cloud server and made accessible to the cosmetics company, the company computes some function of the contextual data and determine which targeted advertisement to send back to the consumer's phone.

Some examples of where context is important for advertising or providing targeted coupons: beer commercials during sports events, or, you are near a Starbucks in the morning and a coffee discount coupon for the Starbucks nearby is sent to your phone, or, cosmetics companies market different products for different times of day (e.g. Friday night going out vs. Sunday morning hanging out with the family), ads or coupons for shows if you are in New York near Broadway in the evening. Other (private) contextual data might be: your income, your profession, your purchasing history, your travel history, your address, etc.

Encrypted version: The problem with these scenarios is the invasion of privacy resulting from giving that much detailed information about the consumer to the server or to the advertising company. Now, imagine an encrypted version of this entire picture. All the contextual data is encrypted and then uploaded to the server; the advertiser uploads encrypted ads to the server; the server computes a function on the encrypted inputs which determines which encrypted ad

to send to the consumer; this function could be either private/proprietary or not. All contextual data and all ads are encrypted to the consumer's public key. Then the cloud can operate and compute on this data, and the consumer can decrypt the received ad. As long as the cloud service provider does not collude with the advertisers, and semantically secure homomorphic encryption is employed, the cloud and the advertisers dont learn anything about the consumer's data.

2.2 Electronic Voting

Electronic voting systems which are provided over the Internet could increase the participants of an election as it eliminates the inconvenience of going to a designated voting place. It would allow an eligible voter to participate from any location that provides Internet access. However, guaranteeing security and privacy are essential for electronic voting systems. Homomorphic encryption schemes can help to hide the content of a ballot by calculating the tally without decrypting any of the ballots. To counteract a corrupt authority the encryption of the tally can be distributed to several authorities in such a way that only coalitions of a certain size can decrypt the tally.

To get a carefully designed voting protocol that can't be easily compromised certain security parameters need to be fulfilled. The following list of security parameters are quoted from [12]:

- **Accuracy (Correctness)** demands that the announced tally exactly matches the actual outcome of the election. This means that no one can change anyone elses vote (**inalterability**), all valid votes are included in the final tally (**completeness**) and no invalid vote is included in the final tally (**soundness**).
- **Democracy:** A system is considered to be "democratic" if only eligible voters are allowed to vote (**eligibility**) and if each eligible voter can only cast a single vote (**unreusability**). An additional characteristic is that no one should be allowed to duplicate anyone else's vote.
- **Privacy:** According to this requirement no-one should be able to link a voter's identity to his vote, after the latter has been cast. Computational privacy is a weak form of privacy ensuring that the relation between ballots and voters will remain secret for an extremely large period of time, assuming that computational power and techniques will continue to evolve in today's pace.
- **Robustness** guarantees that no reasonably sized coalition of voters or authorities (either benign or malicious) may disrupt the election. This includes allowing abstention of registered voters, without causing problems or allowing other entities to cast legitimate votes on their behalf, as well as preventing misbehaviour of voters and authorities from invalidating the election outcome by claiming that some other actor of the system failed to properly execute its part. Robustness implies that security should also be provided against external threats and attacks, e.g. denial of service attacks.

- **Verifiability** implies that there are mechanisms for auditing the election in order to ensure that it has been properly conducted.
- **Uncoercibility (receipt-freeness):** An uncoercible scheme does not allow the voters to convince any other participant (e.g. a coercer) on what they have voted. More specifically, in an uncoercible voting scheme a voter neither obtains, nor is able to construct, a receipt proving the content of his vote.
- **Fairness** ensures that no one can learn the outcome of the election before the announcement of the tally. Therefore acts like influencing the decision of late voters by announcing an estimate, or provide a significant but unequal advantage (being the first to know) to specific people or groups, are prevented.
- **Verifiable participation (declarability)** ensures that it is possible to find out whether a particular voter actually has participated in the election by casting a ballot or not. This requirement is necessary in cases where voter participation is compulsory by law (as in some countries, e.g. Australia, Belgium and Greece) or social context (e.g. small or medium scale elections for a distributed organisation board) where abstention is considered a contemptuous behaviour.

There are several schemes proposed that adopt homomorphic encryption in the domain of electronic voting. As an example I. Damgård and M. Jurik [8] propose an electronic voting scheme that preserves the above parameters except for the receipt-freeness. The following summary of the protocol is based on [12].

Their scheme involves M voters $V_1, ..., V_M$, and N authorities $A_1, ..., A_N$. The initial scheme only allows for "yes" or "no" voting, but it was later expanded to allow 1-out-of-L elections, essentially by holding L elections in parallel. A bulletin board is used for communication among the participants.

In the announcement phase a key for the threshold generalised Paillier encryption is generated and the public key is published on the bulletin board for all to see, while the shares for the private key are secretly given to the corresponding authorities $A_1, ..., A_N$.

In the voting phase each voter chooses a random r_i, encrypts his vote v_i as $E_i = \mathcal{E}(v_i, r_i) = g^{v_i} \cdot r_i^{n^s} \bmod n^{s+1}$ and attaches a proof that it encrypts either 0 or 1. The proof is generated using the Fiat-Shamir heuristic, incorporating voter's identity to prevent vote copying. The resulting vote consisting of the ciphertext and the proof is published on the message board.

In the tallying phase each authority reads the posts on the bulletin board submitted by the voters, and checks for each voter that he has only posted one ciphertext accompanied by a valid proof that is an encryption of either 0 or 1. It then calculates the product of the ciphertext in the valid votes. Because of the homomorphic property of the Paillier encryption scheme the votes get added. The number of voters as well as the resulting product of ciphertexts, $E = \prod_i \mathcal{E}_i(v_i, r_i) = \mathcal{E}(\sum_i v_i \bmod n, \prod_i r_i \bmod n^{s+1})$, is published on the message board. Each authority decrypts E with its key share and posts the result on the message board together with a proof that the decryption was correctly performed.

Having completed these steps, an appointed authority locates the first t authorities that have posted a decryption share together with a valid proof of it being legal. Using these shares the authority computes the product of the published ciphertext to get the result of the election and posts it on the bulletin board. The final tally consists of the number of valid votes and the number of "yes" votes.

Anyone can verify that the votes that have been taken into account are valid, by checking the proof of validity accompanying them. Correct partial decryption by each authority can be verified by the proof of correct decryption posted with the computed share. Finally, the actual result of the election can be computed by anyone, by multiplying the published shares there is no need for a key in order to do this.

2.3 Data Mining

Privacy concerns have become one of the major issues of data mining. Making use of homomorphic encryption could help to protect sensitive attributes of customers data. Current solutions that have been proposed to address the privacy problem are using randomisation techniques on customer data. However such solutions suffer from a trade-off of accuracy and privacy, the more privacy of each customer has, the more accuracy the miner loses in his result.

Yang, Zhong and Wright [16] proposed a solution based on homomorphic encryption that is both fully private and fully accurate. It allows a data miner to compute frequencies of values or tuples of values in the customers' data in a privacy preserving manner. Technically, the problem can be reduced to a problem of securely summing private inputs, which is a similar to the problem of e-voting (Section 2.2). The requirements for an efficient protocol in this domain are that each customer only sends one flow of communication to the miner and that no customer needs to communicate with other customers. Yang, Zhong and Wright present a privacy-preserving joint frequency algorithm as well as a naive Bayes classifier that uses the frequency algorithm as building block. For simplicity reasons I will only detail their frequency mining algorithm here.

In the privacy-preserving primitive for frequency mining they consider a very basic scenario where there are n customers $(U_1 ... U_n)$ and each customer U_i has a Boolean value d_i. The miner would like to find out how many d_i's are 1's and how many are 0's without revealing any of the d_i's.

The protocol Yang, Zhong and Wright propose is based on the additively homomorphic property of the ElGamal encryption scheme and works as follows. Suppose that each customer U_i has two pairs of keys, $(x_i, X_i = g^{x_i})$, $(y_i, Y_i = g^{y_i})$ where g is a generator of group G. The values x_i and y_i are private keys, X_i and Y_i are public keys. Further, all customers have to know the values $X = \prod_{i=1}^{n} X_i$, $Y = \prod_{i=1}^{n} Y_i$, as well as the generator g of group G. Each customer U_i sends the following to the miner:

$$m_i = g^{d_i} \cdot X^{y_i}$$

$$h_i = Y^{x_i}$$

Afterwards the miner calculates:

$$r = \prod_{i=1}^{n} \frac{m_i}{h_i}$$

The miner can then find d, the sum of all d_i's, by checking for all $d = 1...n$ if $g^d = r$. For the one d that holds the equation $g^d = r$, $d = \sum_{i=1}^{n} d_i$. A proof that when the miner finds $g^d = r$, the value d is the desired sum, can be found in [16].

The privacy-preserving frequency mining algorithm was implemented in C and a series of experiments was run by Yang, Zhong and Wright. They were performed on a PC with a 1GHz processor and 512MB RAM. The length of the cryptographic keys were 512 bit. In the protocol it takes each customer only 1 millisecond to prepare her message to the miner. The miner's computation is somewhat longer. Figure 1 shows the time the miner needs to compute one frequency for different numbers of customers. For example, for 10.000 customers, the miner's computation takes 146 milliseconds.

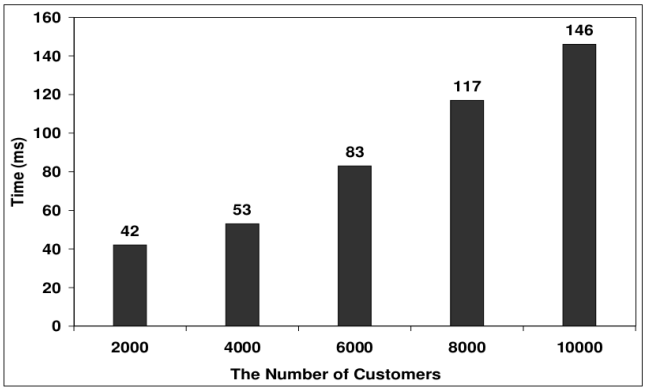

Figure 1: Servers Computation Time for a Single Frequency Calculation.

2.4 Biometric Authentication

With the emerge of biometric authentication systems and their ever growing popularity in recent years, protecting peoples privacy has become increasingly challenging. Biometric data is assumed to be public, as fingerprints, for instance, are left on every object a person touches, and pictures of a persons facial area can easily be taken without notice. It is therefore not the biometric data itself that needs protection, but essentially the relationship between the identity and the biometric feature of an individual.

I consider a generic environment, where an individual tries to authenticate herself to a database using her biometric feature. Typically a reference biometric template for an individual is stored in the database. When a person wants to authenticate herself she has to provide the system with a new sample of her biometric trait, which is normally captured by a sensor (e.g. by a fingerprint

scanner or a camera for iris data). This sample is then matched against the template in the database. As the relationship between the persons identity and her biometric trait must be hidden, comparison of the enrolled template and the newly captured sample has to take place in an encrypted domain. However, the newly captured sample will vary heavily from the one already stored in the database. Hence, using traditional cryptographic methods such as hash-values is not suitable in this scenario but homomorphic encryption my help us here.

Biometric templates are data which often is represented as a binary feature vector, such as iris codes, retina codes, or finger codes. Thus authentication comes down to comparing two binary strings. Several proposals in this domain, such as [6], have shown that one can use the Hamming distance between the two feature vectors and a predefined threshold to achieve the comparison. Homomorphic encryption can accomplish the privacy preserving binary template similarity assessment in the encrypted domain.

Basically the biometric authentication protocol consists of two parts, the enrollment phase and the identification phase. Enrollment is the process of storing an individuals biometric trait in a database, whereas identification is a one-to-many comparison in an attempt to identify an unknown individual. The identification phase itself can again be divided into two parts. At the client side a given template t is encrypted with the according public key and the resulting binary string is transferred to the server component. The server component consists of three independent parts, the authentication server AS, the database server DB and a matcher M.

The authentication server AS deals with the clients service request and randomly fetches all enrolled templates t'_i, $i = 1, ..., k$ where k denotes the number of templates stored in the database. Each enrolled template is then xor-ed against the encrypted template t provided by the client. The resulting binary string is randomly permuted using a permutation function π and sent to the matcher M. As the permutation does not change the number of zeros and ones in the binary string, the permutation has no effect on the resulting Hamming weight. The matcher decrypts the permuted string using the private key and calculates the Hamming weight h_i, which again is sent back to authentications server. If the minimum of all Hamming weights $h_1, \ldots, h_k$ falls within a predefined threshold θ access to the system is granted, otherwise access is denied. The whole identification process is depicted in Figure 2.

The privacy preserving property of this architecture is achieved by distributing the verification/identification process among several independent components and it is essential that no single component is able to track a user and/or can gain any sensitive relationship information between a persons identity and her biometric trait.

One step of the above description needs to be further detailed. How can the xor value of two binary templates be computed without revealing their contents. Of course it is done by making use of the homomorphic property of an encryption algorithm, but the exact way depends on the used encryption algorithm. For the Goldwasser-Micali encryption scheme we can simply exploit

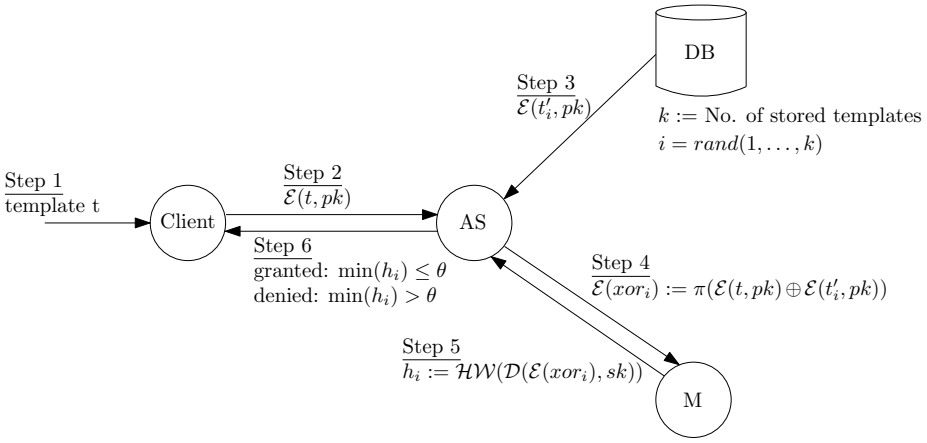

Figure 2: Identification Architecture.

its homomorphic property. For any $m_1, m_2 \in \{0, 1\}$

$$\mathcal{D}_{GM}(\mathcal{E}_{GM}(m_1) \ \cdot \ \mathcal{E}_{GM}(m_2)) = m_1 \oplus m_2$$

In contrast calculating the *xor* of two encrypted binary strings for the Paillier cryptosystem is not as trivial. Thus the process of *xor*-ing two bit-strings needs to be exermined more closely.

Definition 1. *Let $x = (x_1 \ldots x_k), y = (y_1 \ldots y_k)$ be two binary strings of length k. Then*

$$x \oplus y = x_i + y_i - 2x_i y_i, i = 1, \ldots, k$$

As a consequence of Definition 1 a bit by bit encryption of the Paillier scheme has to be applied and then the encrypted *xor* can finally be performed as follows:

$$\tilde{y}_i = -2y_i \bmod n$$

$$\mathcal{E}_P(x_i \oplus y_i) = \mathcal{E}_P(x_i) \cdot \mathcal{E}_P(y_i) \cdot (\mathcal{E}_P(x_i))^{\tilde{y}_i} \bmod n^2$$

where n is part of the public key for the Paillier cryptosystem and all encryption steps also use the public key.

However, encrypting only a single bit is very inefficient as the Paillier cryptosystem is designed to encrypt messages of length $m < n$. A different algorithm for *xor*-ing two messages m_1, m_2 in the encrypted domain can be used which exploits Paillier's property to encrypt messages of length $m < n$. For the remainder of this paper I refer to this algorithm as *Paillier Chunkwise*.

Before encrypting m_1 and m_2, these messages are up-sampled as follows:

$$m_{j,up}[i] = \begin{cases} m_j[\frac{i}{2}], & 2 \mid i \\ 0, & otherwise \end{cases}$$

$i = 1 \ldots 2 \cdot length(m)$, $j \in \{1, 2\}$

Then two encrypted messages can be *xor*-ed as shown below:

$$(m_1 \oplus m_2)[i] = (\mathcal{D}_P(\mathcal{E}_P(m_{1,up}) \cdot \mathcal{E}_P(m_{2,up}) \bmod n^2))[2i]$$
$$i = 1 \ldots length(m)$$

Finally I will look at the performance of the biometric identification architecture. The binary feature vectors got extracted from the *Chinese Academy of Science: CASIA V3 Interval Iris Database*. All of the following results were obtained using an Intel Core i7-2600 4x3400MHz PC with 4GB RAM. Moreover, for all en- and decryption operations a key length of 1024 bits was chosen.

The first experiment in this section measures the identification performance with only one person enrolled in the database. The results are summarized in Table 1.

Table 1: Performance: 1 person enrolled

Scheme	Time (sec)
Unencrypted	< 0.1
Goldwasser-Micali	22.8
Paillier	551.9
Paillier Chunkwise	3.0

We can see that Goldwasser-Micali encryption scheme is approximately 24 times faster than the bit-by-bit Paillier encryption scheme, but still approximately 7 to 8 times slower than Paillier Chunkwise. Although Paillier Chunkwise is the fastest of the examined encryption schemes it is still 30 times slower than using no encryption.

In the second experiment the identification performance with 30 individuals enrolled in the database was measured. The results are presented in Table 2.

Table 2: Performance: 30 persons enrolled

Scheme	Time (sec)
Unencrypted	2.5
Goldwasser-Micali	682.0
Paillier	16200.0
Paillier Chunkwise	90.0

We see that the identification performance of the encrypted schemes is now approximately 30 times slower than with only one individual enrolled. This means that the computing times of the encrypted schemes increase linearly for each additionally enrolled individual. Therefore for large databases the identification process is very time consuming. For instance, if bitwise Paillier encryption scheme is used a person has to wait 16200 seconds or 4.5 hours(!) for its identification result. On the other hand the Goldwasser-Micali encryption scheme reduces the waiting time to approximately 682.0 seconds (11.4 minutes). By far the best results are obtained using Paillier Chunkwise which cuts down identification time to 90 seconds. Still this is far too long to be applicable for a real system.

3 Homomorphic Encryption used in Practice

The only real-life application that is using homomorphic encryption I was able to find is the Helios e-voting system [1]. Helios offers a web-based, open-audit voting system where anyone can create and run an election, and any willing observer can audit the entire process. It is targeting online software communities, local clubs, student government, and other environments where trustworthy, secret-ballot elections are required but coercion is not a serious concern [2].

Helios uses the homomorphic property of the ElGamal cryptosystem to calculate the tally without decrypting any of the ballots. The basic principle of electronic voting described more detailed in Section 2.2.

The fact that it is very hard to find real-life applications that make use of homomorphic encryption either means that it is not used very often in practice or it is not mentioned in the application documentation. Knowing that operations on encrypted data are a huge computational overhead for an application, the first statement might apply.

4 Conclusion

In this paper we saw that homomorphic encryption systems are very rarely used in real-life applications. On the other hand it is an active research area where different usage scenarios and protocols are proposed. In all usage scenarios homomorphic encryption tries to ensure privacy of user data. It is remarkable that usage scenarios which utilize fully- or somewhat homomorphic encryption schemes are still quite abstract, without figures how they would perform in real life. In contrast, usage scenarios that utilize a partially homomorphic encryption scheme include detailed protocols, often together with experiments from implementations. The biggest drawback of applications that use homomorphic encryption is the huge computational overhead that it takes to perform operations on encrypted data.

References

[1] Helios voting. `http://http://heliosvoting.org/`. Accessed: 2013-05-14.

[2] Ben Adida. Helios: web-based open-audit voting. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association.

[3] Josh Benaloh, Melissa Chase, Eric Horvitz, and Kristin Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 103–114, New York, NY, USA, 2009. ACM.

[4] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of Cryptography*, pages 325–341. 2005.

[5] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st annual conference on Advances in cryptology*, CRYPTO'11, pages 505–524, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] Julien Bringer, Hervé Chabanne, Malika Izabachène, David Pointcheval, Qiang Tang, and Sébastien Zimmer. An application of the Goldwasser-Micali cryptosystem to biometric authentication. In *Proceedings of the 12th Australasian Conference on Information Security and Privacy*, ACISP'07, pages 96–106, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] Michael Cooney. Ibm touts encryption innovation. `http://www.computerworld.com/s/article/9134823/IBM_touts_encryption_innovation`, 2009. Accessed: 2013-04-08.

[8] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC '01, pages 119–136, London, UK, UK, 2001. Springer-Verlag.

[9] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[10] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Proceedings of the 30th Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology*, EUROCRYPT'11, pages 129–148, Berlin, Heidelberg, 2011. Springer-Verlag.

[11] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the fourteenth annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[12] Costas Lambrinoudakis, Dimitris Gritzalis, Vassilis Tsoumas, Maria Karyda, and Spyros Ikonomopoulos. Secure electronic voting: the current landscape. In DimitrisA. Gritzalis, editor, *Secure Electronic Voting*, volume 7 of *Advances in Information Security*, pages 101–122. Springer US, 2003.

[13] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, pages 113–124, New York, NY, USA, 2011. ACM.

[14] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[15] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. pages 169–177. Academic Press, 1978.

[16] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *Proceedings of the 5th SIAM International Conference on Data Mining*, pages 21–23, 2005.