Towards Practical Homomorphic Encryption in Cloud Computing

Adil Bouti
FernUniversität in Hagen
Faculty of Mathematics and Computer Science
58084 Hagen,Germany
adil.bouti@gmail.com

Jörg Keller
FernUniversität in Hagen
Faculty of Mathematics and Computer Science
58084 Hagen,Germany
joerg.keller@fernuni-hagen.de

***Abstract*—Secure computing in clouds faces many challenges related to data confidentiality and integrity. Classical security models focus on securing data from outside attacks, e.g. from other cloud users. Yet, breach of data confidentiality by the cloud provider has received far less attention. In previous work, we presented a protocol to delegate computations into clouds, providing security against other cloud users and cloud providers through encrypted data. The protocol is based on homomorphic properties of encryption algorithms. However, that protocol was only practical in certain circumstances. In the present paper we introduce some practical extensions to our algorithm to improve its efficiency. Additionally we extend the algorithm to support multiparty computation while preserving its homomorphic properties. We then show how these optimization blocks can be used for applying the scheme to efficient face recognition using Eigenface recognition algorithm.**

***Keywords*-Cloud Computing; Homomorphic Encryption; Privacy-preserving Face Recognition; Secure Delegation;**

I. INTRODUCTION

Cloud computing is one of the most distinctive advances in the IT world during the last decade. IT companies provide flexible cloud services and infrastructures almost for every purpose, allowing customers to quickly scale their services using pay-by-use provisioning models.

Concerns about potential risks involved with the loss of privacy, integrity and availability of the user data on cloud networks, due to technical failures or even targeted attacks from both outside and inside the cloud provider's network, have been discussed in several scientific works [13], [14]. Cloud providers generally use common security protection models that require to trust the cloud provider, and classical user transparent security measures to protect their networks and to satisfy the security goals of their customers. These measures are still in an early stage and are prone to common security threats, such as cloud providers forced by local (political) pressure to reveal cloud user data.

To fill the need for encrypted computing in cloud networks we presented an approach for secure delegated computation [6]. Computations on encrypted data are based on two independent homomorphic encryption algorithms, providing homomorphy for additions and multiplications. The model consists of an agent acting as the delegator and at least one worker in the cloud who performs the computation. However, the need to re-encrypt data to switch between addition and multiplication operations limited applicability. In the present work, we introduce measures to increase the efficiency of our protocol, and to increase security by multiparty computation. Furthermore we demonstrate the applicability of our approach with the implementation of a privacy-preserving face recognition system by means of homomorphic encryption schemes. The typical scenario here is a delegator-worker(s) application where the client needs to know whether a specific face image is contained in the encrypted face repertoire of a server. Our application requires that the delegator trusts the worker(s) to correctly perform the requested operations for the face recognition but without revealing any information to the server about the requested image as well as about the outcome of the matching algorithm.

The remainder of this article is organized as follows. In Section III, we review and improve the concept of delegated computation based on multiple, partly homomorphic public key cryptosystems. In Section IV, we discuss the delegation of the encryption and decryption overhead to multiple cloud providers. In Section V we present a privacy-preserving face recognition algorithm using our protocol. In Section VI, we give a conclusion.

II. RELATED WORK

The increasing demand for flexible and secure cloud solutions for private and commercial use has attracted a lot of attention to alternative schemes allowing to evaluate circuits over encrypted data. In [1] a "somewhat homomorphic scheme" able to perform view multiplications was presented. The first working fully homomorphic encryption (FHE) schemes were published in [3], [4], [5]. The complexity of FHE schemes to encrypt and decrypt one bit using FHE schemes and the not proportionate ciphertext sizes have mobilized many researchers and institutes to contribute improving FHE performance to make fully homomorphic encryption and secure multi-party computations more practical. First attempts [2], [26] to integrate FHE schemes in cloud-ready systems show that the computational costs for a single fully homomorphic operation are dramatically larger than the results presented in this work. Previous papers focusing on the implementation of privacy preserving face recognition and feature extraction used different approaches to reach this goal. Erkin et al. [22] protocol is based on the homomorphic properties of Paillier to achieve face recognition using encrypted face images. Sadeghi and Schneider [21] combined Yao garbled circuits and homomorphic algorithms to implement practicable face recognition. Qin et al. [25] combined different cryptographic primitives to implement privacy preserving image feature extraction.

III. Precomputed encrypted computation

In previous work [6] we presented a method to delegate computations on confidential data into a cloud, with the assumption that the majority of computations would be additions or multiplications, or could be transformed into them. Our approach uses two homomorphic encryption algorithms A and M, one for addition and one for multiplication. The delegator first encrypts confidential data with A, transmits them to the cloud, and starts the application. The application works on the encrypted data. If a multiplication occurs, the encrypted data is sent back to the delegator, decrypted with A, encrypted with M, and sent again to the cloud. In the present work we use RSA [9] for homomorphic multiplication operations and Paillier [7] for additions.

Even when the computational performance of the cloud is very high, the method only relieves the delegator if the effort to encrypt and decrypt data is smaller than for the computation itself. We thus also discussed means to reduce the frequency of encryptions and decryptions, e.g. by reordering of computational steps, and bundling several transmissions to reduce communication overhead. Still, the effort remained high so that only some applications could profit from our approach.

Precomputation is a widely used low level optimization technique used e.g. by processors to reduce the operation latency while precomputing input independent operations or performing speculative memory accesses. In cryptography, precomputation is a classic and a well-studied technique adopted e.g. in a fixed base exponentiation method in order to precompute modular exponentiations in cryptographic operations.

Let $X = \sum_{i=0}^{n-1} z_i \cdot 2^i$ with $n \in \mathbb{N}$ and $z_i \in \{0,1\}$, i.e. $z_{n-1}, ..., z_0$ is a binary representation of a non-negative integer X. In order to reduce the encryption overhead for encrypted addition operations using additive homomorphic encryption algorithms such as Paillier [7], the delegator computes the encryption of all powers of two within a specified number range. The precomputed encrypted powers of two can be combined in order to encrypt an arbitrary number using its binary representation:

$$E_k(X) = \prod_{z_i=1} E_k(2^i) \mod n^2$$

For a processor with a bit width of n bits, we should precompute n encryptions (each 2^i value) within the number range of the specified bit width. The proposed usage of precomputation for encrypted arithmetic additions eliminates the overhead for modular exponentiations in favor of modular multiplications.

The usage of precomputation for encrypted addition operations will speed up our algorithm due to the fact that additions represent the majority of the operations processed within numerical algorithms. The optimization of our algorithm with precomputation has many advantages, on the one hand the precomputation overhead is low compared to the encryption overhead. On the other hand the memory consumption through precomputation is manageable.

Furthermore the precomputed values and the respective plaintext values can be stored in an efficient data structure by the delegator in order to compare the encrypted results with the values stored in the data structure, like a result cache, in order to decrease the decryption overhead for arithmetic additions over encrypted data.

We can also compute and store encryptions of all t-bit combinations with $2^t \cdot l$ storage consumption, with l the length of encrypted number bits.

Let $X = (x_{n-1}, \ldots, x_0)$ a binary representation of an integer and $l := \sum_{i=0}^{n-1} x_i$ the number of relevant bits.

The encryption $X = (x_{n-1}, \ldots, x_0)$ (assume n is a multiple of t) can be computed as fallows:

$$E(X) = \prod_{i=0}^{\frac{n}{t}-1} E_k(x_{i \cdot t+t-1}, \cdots, x_{i \cdot t})$$

The power of 2 is a straightforward special case of $t = 1$. If $t = 10$ and $n = 64$, then only 7 multiplications are necessary in contrast to 32 on average or 64 in worst case for $t = 1$.

Using precomputation for addition operations the delegator performs $l - 1$ multiplications with the complexity of $\theta(n)$. Compared to our initial implementation we avoid the overhead of the modular exponentiations needed for each encryption using $l-1$ multiplications. In the present work we waive using precomputation for RSA-encrypted values,due to the huge complexity for the delegator to compute the prime factors for every multiplication operand. Nevertheless we show in IV how to use this approach to delegate the RSA-encryption of chosen values.

Figure 1 depicts the advantages of precomputation for variables of different bitwidth in a sample application.

IV. Distributed Homomorphic Encryption

In this section we present how the deployed encryption algorithms can be used to support distributed computing using homomorphic encryption. In the following use case the delegator wants to outsource the encryption and decryption of the input data and results to $k + k'$ cloud workers. This

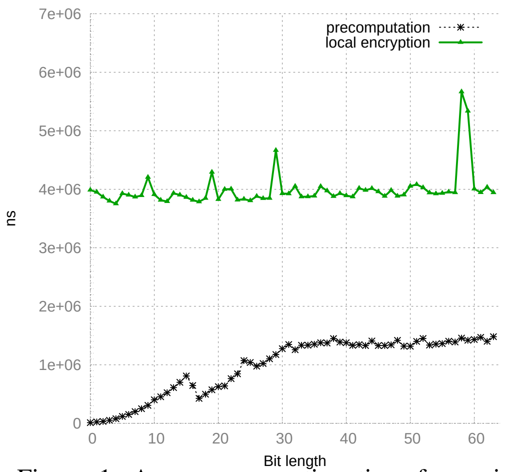

Figure 1: Average encryption time for variables with and without precomputation

is a critical point, as on the one hand, the data should not be disclosed in plaintext to any external party, to protect confidentiality, but on the other hand, if the effort for encryption and decryption on the delegator side is higher than the computation itself, then the delegator would be better off to perform the computation locally.

In the following we will present our extensions using the example of two encryption algorithms, RSA [9] (multiplicative homomorphic) and Paillier [7] (additive homomorphic) to support distributed homomorphic encryption.

A. RSA

Let us suppose the delegator and the workers have already exchanged an RSA modulus N e.g. using the shared RSA key generation protocol by Damgård et al. [10], [11].

Let e be an RSA exponent co-prime to $\varphi(N)$ and d the corresponding RSA decryption exponent such that

$$\begin{aligned} d &= \{d_1\} + \{d_2 + ... + d_{k'}\} \\ &= d' + d'' \end{aligned}$$

where d_i are positive integers. The representations of e and d have no impact on the correctness of the encryption and decryption computations and thus must fulfill only the requirement that the number of operands d_i is equal to the number of the workers performing the encryption respectively the decryption. Notice the number k of workers performing the encryption is not necessarily equal to the number k' of workers to decrypt messages.

The delegator distributes the partial exponent d_i to the involved workers. Let $m \in \mathbb{Z}_N^*$ be a plaintext value with $m = m_1 \cdots m_n = \prod_{k=1}^{n} m_k$. the prime factors of m, the delegator has the possibility to delegate the encryption computation to a number of clients $k \leq N$. Each partial ciphertext $c_i = {m_i}^e \bmod N$ is computed by a worker. The partial ciphertexts c_i can be multiplied by the delegator or any other cloud worker in order to compute the final ciphertext c:

$$\begin{aligned} c &= c_1 \cdot c_2 \cdots c_k \bmod N \\ &= m_1^e \cdot m_2^e \cdots m_k^e \bmod N \\ &= (m_1 \cdots m_2 ... \cdots m_k)^e \bmod N \\ &= m^e \bmod N\,. \end{aligned}$$

The encryption can be securely outsourced to the workers even if the modulus e is known. In order to prevent that rogue cloud-workers can reconstruct m, the delegator can e.g. use random blind factors in order to obfuscate m for the workers. The blind factors can be removed after the decryption dependent on the performed multiplications on the initial ciphertext. By the use of e.g. two independent cloud providers, the delegator chooses carefully m_1 and m_2 with $m = m_1 \cdot m_2$ and each of the cloud workers computes the encryption of one of the ciphertext parts $(E_e(m_1), E_e(m_2))$. The delegator can combine the ciphertexts to reconstruct $E_e(m)$ without that the cloud workers are able to gain knowledge about m. Alternatively the delegator can generate a random number r and computes $r' = r^d$ and $m' = m \cdot r' \bmod N$ and sends m' to the worker. The worker computes:

$$\begin{aligned} c' &= m'^e \mod N \\ &= (m \cdot r')^e \mod N \\ &= m^e \cdot r^{d \cdot e} \mod N \\ &= m^e \cdot r \mod N \end{aligned}$$

The delegator recovers c by multiplying c' by $r^{-1} \cdot r \equiv 1 \pmod N$ The random values guarantees that the workers aren't able to reconstruct neither m from m' nor r from r', enabling the delegator to achieve semantic security for RSA achieving a similar effect as using special padding RSA padding schemes (e.g. RSA-OAEP+ [17], [18]), without the loss of the homomorphic property.

Decrypting a message m can be carried out as described by Shoup [12], every decryption client computes $c^{d_i} \bmod N$. The delegator computes $m = \prod_i c^{d_i} \bmod N$.

$$m = c^{d'} \cdot c^{d''} \bmod N$$

$m' = c^{d''}$ can be computed by the workers, every decryption client computes $c^{d_i} \bmod N$ with $i \in \{2, ..., k'\}$. This part of the computation can be easily parallelized and considerably accelerate the decryption. The delegator finally computes $m = c^{d'} * m' \bmod N$ in order to recover the message plaintext. The secret key part d' kept secret by the delegator has few bits and can be used to compute m locally in a efficient and resource-friendly manner.

We take in consideration that the cloud provider could reconstruct the key if all subkeys d_i are available on the cloud. Alternatively, different cloud providers can be used for different parts of the input data, so that no provider ever sees all inputs in ciphertext. A more elaborate scheme could split the decryption on two independent cloud providers, preventing that a malicious cloud provider receives enough subkeys allowing in the worst case to rebuild the secret key or at least to brute-force the remaining parts of it. The delegator has also the possibility to keep a defined part of the secret subkeys, in order to prevent attacks on the encryption secret keys by a malicious cloud provider.

So far our base assumption is that the encryption cloud workers receive no input data in plaintext in order to compute the partial encryptions and the message can be split up into several parts of which each is not confidential enough to not be trusted to one encryption provider. Alternatively, different cloud providers can be used for parts of the input data, allowing no provider to proceed all inputs. A more elaborated scheme could split a message m into two factors m_1 and m_2 such that $m = m_1 \cdot m_2$, have the two factors encrypted separately, and generate the encrypted message by multiplying the partial encryptions of both factors. Note that the input data typically is from a much smaller range (32-bit or 64-bit values), so that factorization is not resource costly for the delegator. Furthermore the delegator can use encrypted values present in the prime factors of the message m in order to reduce encryptions and compensate the factorization overhead. A malicious cloud provider shouldn't be able to distinguish between a full input message and factors of it. thus the worker receives encrypted values sequences including whole messages and prime factors of messages without any possibility to distinguish between the context in which the encrypted values are used in.

Note the use of RSA as multiplicative homomorphic encryption scheme is only appropriate in use cases where semantic security is not required and one-way security is sufficient.

B. Paillier

Let N be an RSA modulus, (n, g) a Paillier public key and (λ, μ) the corresponding private key.

Let $m \in \mathbb{Z}_n^*$ be a plaintext message with $m = m_1 + \cdots + m_k$ with $m >> k$ and $r_i \in \mathbb{Z}_n^*$ a random number. The partial encryptions c_i of m can be computed by the distributed clients as follows:

$$c_i = g^{m_i} \cdot r_i^n \mod n^2 \, .$$

The complete encryption of m can be computed based on the k partial encryptions c_i by the delegator or any other worker as follows:

$$\begin{aligned} c &= g^{m_1} \cdot g^{m_2} \cdots g^{m_k} \cdot r_1^n \cdot r_2^n \cdots r_k^n \mod n^2 \\ &= g^{(m_1+m_2 \ldots +m_k)} \cdot r^n \mod n^2 \\ &= g^m \cdot r^n \mod n^2 \, . \end{aligned}$$

For λ the least common multiple of $(p-1, q-1)$, $\lambda = \lambda_1 + \ldots + \lambda_k$ a Paillier private key, the decryption of an encrypted message c can also be computed by the distributed workers if the partial encrypted messages c_is are available.

$$m = L(c^\lambda \mod n^2) \cdot \mu \mod n$$

with

$$\mu = \frac{1}{L(g^\lambda \mod n^2)} \mod n$$

The decryption can be moved to k workers, where every worker has to compute:

$$m_i = c^{\lambda_i} \mod n^2$$

The delegator receives the respective values of c_i from the workers and can compute the plaintext message m as follows:

$$\begin{aligned} m &= L(\prod_{i=1}^{k} c^{\lambda_i} \mod n^2) \cdot \mu \mod n \\ &= L(c^{\Sigma_{i=1}^{k} \lambda_i} \mod n^2) \cdot \mu \mod n \\ &= L(c^\lambda \mod n^2) \cdot \mu \mod n \, . \end{aligned}$$

The security of the decryption relies on outsourcing merely parts of the secret key λ_i not allowing to decrypt the ciphertext or parts of it. In the two party setting of the semi-honest model, The cloud provider shouldn't possess enough λ_i allowing to decrypt the cipher text or to brute-force the remaining parts. We notice that our protocol doesn't leak any partial information about the number k of private key parts nor are the number of involved workers revealed. In contrast to factorization-based key parts, the addition based key parts aren't distinct and thus able to include sufficient entropy in order to complicate the prediction of key parts. Paillier is semantically secure under a strong decisional assumption as shown by Paillier [7] and provides indistinguishability under chosen-plaintext attacks. Nevertheless the distributed encryption functionality theoretically allows performing chosen plaintext attacks through combining several message parts.

One disadvantage of sharing private key parts is that the storage and transmission of the parts of keys requires an amount of storage and bandwidth resources equivalent to the number of used key parts between the delegator and the involved workers.

C. Protocol overhead

Our initial protocol presented in [6] encrypts every variable with a specific encryption algorithm in order to exploit its homomorphic property to compute arithmetic operations on the encrypted values. Using algorithms from computer arithmetic [15], we implement operations on long numbers via operations on shorter numbers. For typical sizes of 1024-bit key lengths we get:

$$\begin{aligned}MOD_E(d,n) &= 1.5 \cdot l(d)[M(l(n)) + 2Mod(l(n)) + 1]\\ M(w) &= 3M(w/2) + 5A(w) + 2S\\ A(w) &= w/32\\ Mod(w) &= Mod(w/2) + 4M(w/2) + 1.5A(w) + 3S\end{aligned}$$

Where $l(x)$ denotes the bit length of an argument. We assume that the operations $M(64)$, $A(64)$, $Mod(64)$ and the shift operation S need each one CPU clock cycle. Chinese remaindering as well as random number generation is considered to be negligible.

$$\begin{aligned}M(1024) &= 3M(w/2) + 5A(w) + 2S\\ &= 3M(512) + 160 + 2\\ &= 3[3M(256) + 5A(512) + 2S] + 162\\ &\dots\\ &= 81M(64) + 1380\\ &= 1461\end{aligned}$$

$$\begin{aligned}Mod(1024) &= Mod(w/2) + 4M(w/2) + 1.5A(w) + 3S\\ &= Mod(512) + 4M(512) + 1.5A(1024) + 3\\ &\dots\\ &= Mod(64) + 4M(64) + 1.5A(128) + 3S + 2393\\ &= 2407\end{aligned}$$

The cpu clock cycles for a modular exponentiation using a 1024 Bit key can be computed as follows:

$$\begin{aligned}MOD_E(d,n) &= 1.5.l(d)[M(l(n)) + 2Mod(l(n)) + 1]\\ &= 1.5.1024[M(1024) + 2Mod(1024) + 1]\\ &= 1536(1461 + 4814 + 1)\\ &= 9639936\end{aligned}$$

using the optimized exponentiation techniques e.g. based on the chinese remainder theorem (CRT) as described in [16], the cpu clock cycles need can be reduced to:

$$\begin{aligned}MOD_E(d,n) &= 2MOD_E(d/2,n/2) + A(512) + 2M(512)\\ &\quad + Mod(512)\\ &= 2(1.5 \cdot l(d/2)[M(l(n/2)) +\\ &\quad 2Mod(l(n/2)) + 1]) + 16 + 866 + 624\\ &= 1.5.512[M(512) + 2Mod(512) + 1] + 1506\\ &= 2585058\end{aligned}$$

The number of cpu clocks needed for each encryption operation using RSA is generally less than the decryption due to the smaller value of the public key e compared to d. In our computations we used the common exponent $e = 2^{16}+1$.

$$\begin{aligned}MOD_E(e,n) &= 1.5 \cdot l(e)[M(l(n)) + 2Mod(l(n)) + 1]\\ &= 141210\end{aligned}$$

The Paillier encryption is performed using two modular exponentiations modulo n^2:

$$c = g^m \cdot r^n \mod n^2$$

The total number of arithmetic operations needed by the encryption is dependent on the plaintext message m. The number of operations needed by the exponentiation $r^n \mod n^2$:

$$\begin{aligned}MOD_E(n,n^2) &= 1.5 \cdot l(n)[M(l(n^2)) + 2Mod(l(n^2)) + 1]\\ &= 1.5 \cdot l(n)(M(2048) + 2Mod(2048) + 1)\\ &= 32879616\end{aligned}$$

This computation is the most complex part of the encryption and can be easily precomputed by the delegator. For g^m the computation can be significantly accelerated by using a small value for g, provided it fulfills the requirement $g \in \mathbb{Z}^*_{n^2}$. In our example we choose $g = 2$. The total amount of operations needed for

$$\begin{aligned}MOD_E(m,n^2) &= 1.5 \cdot l(m)[M(l(n^2)) + 2Mod(l(n^2)) + 1]\\ &= 1.5 \cdot l(m)(M(2048) + 2Mod(2048) + 1)\\ &= 32109 \cdot l(m)\end{aligned}$$

The total amount of arithmetic operations needed for each Paillier encryption is $32109 \cdot l(m)$. The Paillier decryption $m = \frac{c^\lambda \mod n^2 - 1}{n}$ is practically one modular exponentiation modulo n^2. In our example λ has a bit length of 1024 bits.

$$\begin{aligned}MOD_E(\lambda,n^2) &= 1.5 \cdot l(\lambda)(M(l(n^2)) + 2Mod(l(n^2)) + 1)\\ &= 1.5 \cdot 1024(M(2048) + 2Mod(2048) + 1)\\ &= 32879616\end{aligned}$$

The presented distributed extension reduce the huge encryption and decryption overhead for encryption algorithms based on modular exponentiations to the overhead of comparably fast arithmetic multiplications. The computed complexities of modular exponentiations with common moduli show the huge complexity of our initial approach in [6]. The distributed computation approach brings a significant improvement to our algorithm as shown in the benchmark results in Figure 2 and Figure 3 measured on a 64 Bit system with Intel Core i5-2540M 2.60GHz CPU and a local cloud infrastructure connected via 100 MBit/s local area network. The benchmark is based on distributed computation of homomorphic multiplications and additions of integer values of different sizes. Table II shows the average encryption and decryption time needed by the delegator when using the distributed algorithm compared to local computation.

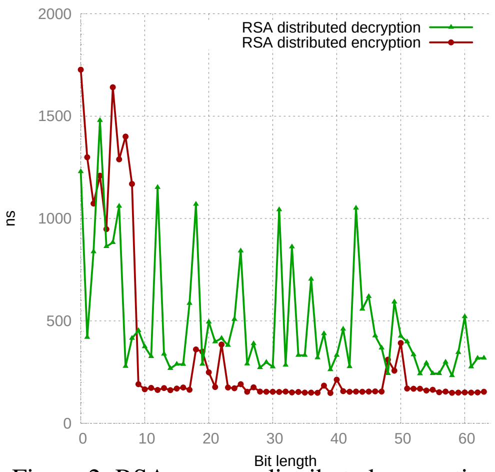

Figure 2: RSA average distributed encryption and decryption time

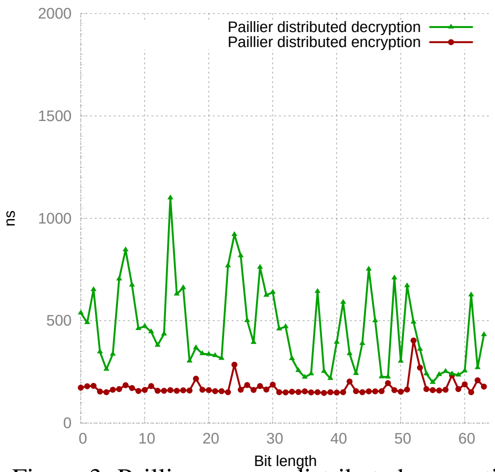

Figure 3: Paillier average distributed encryption and decryption time

Algorithm	Nanoseconds/Operation	
	local	distributed
RSA $_{\text{1024 Bit}}$ Encryption	301075	341
RSA $_{\text{1024 Bit}}$ Decryption	15213828	492
Paillier $_{\text{1024 Bit}}$ Encryption	2625474	173
Paillier $_{\text{1024 Bit}}$ Decryption	2403878	455

Table I: Encryption and decryption time average for local and distributed computations

Due to the underlying basic arithmetic operations and the parallelization potential of the construction. The complexity of each distributed encryption and decryption is less complex than performing the encryption operations by the delegator locally. The management and network overhead for the delegator grow linear to the number of variables used in the outsourced computations and the number of necessary encryption updates in order to switch between different arithmetic operations. Furthermore the delegator may use Shamir secret sharing algorithm in order to distribute the plaintext variables securely to trustworthy workers.

V. Eigenface

Sirovich and Kirby [23] have developed a face classification algorithm, transforming faces into a set of eigenfaces forming a low-dimensional vector space (also known as face space). The eigenfaces are generated through computing principal components analysis on the set of training faces to achieve a vector representation of every face image to the face space spanned by the eigenfaces. The eigenvectors are derived from the covariance matrix of the probability distribution over the high-dimensional vector space of face images.

Matthew Turk et al. have developed [19], [20] in 1991 a dimension reduction method of calculating the eigenvectors of a covariance matrix on a large number of face images based on size independent matrices sized only by the number of images within the training set. This improvement allowed to apply the face classification algorithm of Sirovich and Kirby to achieve practical computer based face recognition. Recognition of a face is done by first projecting the query face image to the face space and subsequently locating the closest matching vector. The preparation phase of the recognition consists of the following steps:

1) The average of the training face images Ψ is computed as follows:

$$\Psi = \frac{1}{M}\sum_{i=0}^{M}\theta_i$$

The average face is then subtracted from each test image within the training set (composed of M training faces $\{\theta_1, \cdots, \theta_M\}$) to compute the difference vectors $\phi_i = \theta_i - \Psi$. The difference vectors describe how much each image differs from the average image Psi.

2) The covariance matrix of the difference vectors is generated as follows:

$$S = \frac{1}{M}\sum_{i=0}^{M}\phi_i\phi_i^T$$

3) To obtain orthonormal eigenvectors and the corresponding eigenvalues we need to apply principal components analysis to the covariance matrix S. The eigenvectors of the covariance matrix have then the same dimension as the underlying test images and give

the direction in which the eigenvector differs from the average image.

4) We choose the principal components $K \ll M$ arbitrary. From the set of M training images $\theta_1, ..., \theta_M$ to determine an appropriate K-dimensional face space, in which face images are projected in and represented using the eigenfaces $u_1, u_2, ..., u_K$ (corresponding to the K largest eigenvalues) as points. The result is the projected face images $\Omega_1, \Omega_2, ..., \Omega_K$.

The recognition algorithm consists of three phases:

1) Given a test image Γ we compute the projection of Γ in to the face space by calculating $\overline{\omega}_1 = u_i^T(\Gamma - \Psi)$ to obtain the projected face $\overline{\Omega} = (\overline{\omega}_1, \cdots, \overline{\omega}_K)$
2) For all images within the test set the distances between the vector $\overline{\Omega}$ and the feature vectors $\Omega_1, \cdots, \Omega_M$ are computed:
$$D_i = \|\overline{\Omega} - \Omega_i\|^2$$
3) The matching face image is calculated through comparing $D_{min} = min(D_1, \cdots, D_m)$ with a specific threshold value ε. A match is given when the smallest distance D_{min} is smaller than ε otherwise a match isn't given.

A. Homomorphic Eigenface

In this section we present a privacy preserving Eigenface implementation operating on encrypted face images. In our proof of concept we wanted to test the practicability of our approach using a real life scenario (video surveillance, border control, etc.) where one requires privacy-preserving online face recognition. Our protocol is based on a delegator owning a face image he wants to recognize against a database previously delegated to one or many cloud workers.

Notation: We denote an encrypted value x with an additive homomorphic encryption algorithm by $[\![x]\!]^+$. An encrypted value with a multiplicative homomorphic property is denoted by $[\![x]\!]^\times$. With $\ominus$ and $\otimes$ we denote the homomorphic arithmetic operations performed on encrypted values.

1) Offline phase: We use in the offline step the preparation phase as shown above to generate the The average of the training face images Ψ, the eigenfaces $u_1, \cdots, u_K$ and the feature vectors $\Omega_1, \cdots, \Omega_K$. We assume that the delegator can calculate the encrypted values $[\![\Psi_i]\!]^+$, $[\![u_i]\!]^\times$ with $\{i \in \mathbb{N} : i \leq K\}$ and $\{[\![\Omega_1]\!]^+, \cdots, [\![\Omega_K]\!]^+\}$ during the processor idle times. The delegator transfers the one-off precomputed encrypted values to the cloud workers to perform the recognition steps later.

2) Online phase: The projection of an encrypted image face $[\![\Gamma]\!]$ is calculated as follows:

$$[\![\overline{\omega}_i]\!] = [\![u_i]\!]^\times \otimes ([\![\Gamma]\!]^+ \ominus [\![\Psi]\!]^+)$$

for each $i \in \{1, \cdots, K\}$. The feature vector $[\![\overline{\Omega}]\!] = [\![(\overline{\omega}_1, \cdots, \overline{\omega}_K)^T]\!]$. The projection operations are primarily performed by the workers using the homomorphic properties of the used encryption algorithms, the delegator task is limited to perform dispatching tasks and recrypting operations when needed.

After computing the feature Vector $[\![\overline{\Omega}]\!]$, the Euclidean distance between $\overline{\Omega}$ and all the feature vectors within the database $D_i(\Omega_i, \overline{\Omega}) = \|\Omega_i - \overline{\Omega}\|^2$ with $i \in \{1, \cdots, M\}$ is calculated by the workers. The distances $[\![D_i]\!]$ are returned to the delegator where he selects the index with the distance smaller than the threshold value ε otherwise the recognition will fail.

B. Implementation

We used the "ORL Database of Faces" from AT&T Laboratories Cambridge [24], which contains 10 images of 40 distinct subjects. All images within the database have a dark background with the subject in upright, frontal position. The size of each image is 92×112 pixels with 8-Bit grey levels per pixel.

In our basic set the delegator communicates with a dynamic number of cloud workers up to 32 (the peak value to achieve efficient cpu load on the delegator side). The Programs were developed in ANSI C using the GNU MP Library for arbitrary precision computations. For efficiency reasons, floating point values were converted to fixed points values. Tests were performed on a system with Intel®Core™i5-2540M CPU @ 2.60GHz and 8 GB RAM. The network communication was performed within a 100 MBit LAN network.

To achieve the measurements results presented in [22] we need round about 32 independent cloud workers. The overhead is due to the recryption operations needed to switch between homomorphic abstraction and multiplication operations not needed in [22] (under the assumption that eigenface vectors can be made public if the underlying face image database isn't classified as a secret).

M	sec./recognition		
	w=8	w=16	w=32
40	138	73	37
80	157	81	40
160	182	89	46
320	203	103	51
400	211	113	62

Table II: Computation complexity for different Database sizes M and distributed workers w

VI. Conclusion and future work

We presented improvements to a protocol for computation on encrypted data in clouds. Our analyses indicate that the overhead is low enough to make the protocol practical. The distribution of computations among several cloud providers increases security at the cost of additional communication.

The algorithms presented in Chapters III and IV can be combined as needed in order to optimize the tradeoff between local and remote encryption processing. The support of precomputation and distributed encryption and decryption is a significant progress towards applicable homomorphic applications. Future work will focus on implementing a prototype application supporting more encryption algorithms with homomorphic properties and efficient and privacy-friendly face recognition and feature extraction algorithms to provide evidence of its feasibility for productive use. A full length comparison against available privacy preserving face recognition algorithms will be also considered.

References

[1] M. Naehrig and K. Lauter and V. Vaikuntanathan. Can Homomorphic Encryption Be Practical. *In Proc. 3rd ACM Workshop on Cloud Computing Security Workshop (CCSW '11)*, pages 113-124, 2011.

[2] M. Brenner, H. Perl and Matthew Smith. Practical Applications of Homomorphic Encryption. *In Proc. 7th Int. Conference on Security and Cryptography (SECRYPT 2012)*, pages 5–14, 2012.

[3] C. Gentry. Homomorphic encryption using ideal lattices. *In ACM Symposium on Theory of Computing (STOC'09)* , pages 169–178, 2009.

[4] N. P. Smart and F. Vercauteren. Homomorphic encryption with relatively small key and ciphertext sizes. *In Public Key Cryptography (PKC'10), volume 6056 of LNCS*, pages 420–443, 2010.

[5] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. *In Advances in Cryptology (EUROCRYPT'10), volume 6110 of LNCS* , pages 24–43, 2010.

[6] A. Bouti and J. Keller. Securing cloud-based computations against malicious providers. *In SIGOPS Oper. Syst. Rev., Vol. 46, No. 2*,pages 38–42, 2012.

[7] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *In Proc. 17th Int.l Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT '99)*, pages 223–238, 1999.

[8] A. Shamir. How to share a secret. *In Commun. ACM, Vol. 22 No.11*, pages 612–613, 1979.

[9] R. L. Rivest and A. Shamir and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *In Commun. ACM, Vol. 26*, pages 96–99, 1983.

[10] I. Damgård and K. Dupont. Efficient threshold RSA signatures with general moduli and no extra assumptions. *In Proc. 8th Int.l Conference on Theory and Practice in Public Key Cryptography (PKC '05)*, pages 346–361, 2005.

[11] I. Damgård and M. Koprowski. Practical threshold RSA signatures without a trusted dealer. *In Proc. Int.l Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '01)*, pages 152–165, 2001.

[12] V. Shoup. Practical threshold signatures. *In Proc. 19th Int.l Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT '00)*, pages 207–220, 2000.

[13] T. Ristenpart and E. Tromer and H. Shacham and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. *In Proc. 16th ACM conference on Computer and communications security (CCS '09)*, pages 199–212, 2009.

[14] D. Osvik and A. Shamir and E. Tromer. Cache attacks and countermeasures: the case of AES. *In Proc. 2006 The Cryptographers' Track at the RSA conference on Topics in Cryptology (CT-RSA'06)*, pages 1–20, 2006.

[15] R. Hwang and F. Su and Y. Yeh and C. Chen. An Efficient Decryption Method for RSA Cryptosystem. *In Proc. 19th Int. Conference on Advanced Information Networking and Applications (AINA '05), Vol. 1*, pages 585–590, 2005.

[16] H. Kamarulhaili and N. Basir. RSA Decryption Techniques and the underlying Mathematical concepts. *In Int. J. of Cryptology Research, Vol. 1, No. 2*, pages 165-177, 2009.

[17] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption How to Encrypt with RSA. *In Lecture Notes in Computer Science. vol. 950 (EUROCRYPT 94)*, pages 92-111, 1994.

[18] V. Shoup. OAEP Reconsidered. *In Proc. 21st Annual Int. Cryptology Conference on Advances in Cryptology (CRYPTO 01), Report 2000/060*, pages 239-259, 2001.

[19] M. Turk and A. Pentland. Eigenfaces for Recognition. *In J. of Cognitive Neuroscience, Vol. 3, No. 1* , pages 71-86, 1991.

[20] M. Turk and A. Pentland. Face recognition using eigenfaces. *In IEEE Computer Vision and Pattern Recognition (CVPR91)*, pages 586-591, 1991.

[21] A. Sadeghi and T. Schneider and I. Wehrenberg. Efficient privacy-preserving face recognition. *In Proc. 12th Int. Conference on Information Security and Cryptology (ICISC'09)*, pages 229-244, 2010.

[22] Z. Erkin and M. Franz and J. Guajardo and S. Katzenbeisser and I. Lagendijkand and T. Toft. Privacy-Preserving Face Recognition. *In Proc. 9th Int. Symposium on Privacy Enhancing Technologies (PETS '09)*, pages 235-253, 2009.

[23] L. Sirovich and M. Kirby. Low-Dimensional procedure for the characterization of human faces. *In J. of the Optical Society of America, Vol. 4*, pages 519524, 1987.

[24] The ORL Database of Faces. Available at. *http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html*, AT&T Laboratories Cambridge.

[25] Z. Qin, J. Yan, K. Ren, C.Chen, and C. Wang. Towards Efficient Privacy-preserving Image Feature Extraction in Cloud Computing. *In Proc. ACM Int. Conference on Multimedia (MM '14)*, Pages 497–506, 2014.

[26] M. Brenner, H. Perl and Matthew Smith. How Practical is Homomorphically Encrypted Program Execution? An Implementation and Performance Evaluation. *In 11th IEEE Int. Conference on Trust, Security and Privacy in Computing and Communications (TrustCom-12)*, pages 375–382, 2012.