Securing Cloud-based Computations against malicious providers

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
Security in clouds often focuses on preventing clients from gaining information about other clients' computations. However, cloud providers might also be a source for loss of confidentiality. We present a protocol to delegate computations into clouds with encrypted data. The protocol is based on homomorphic properties of encryption algorithms. The protocol can also be used to amend existing applications by software patches of binaries. We evaluate the protocol by a proof-of-concept implementation to investigate practicability, and discuss variants and extensions to increase the prototype's efficiency.

Categories and Subject Descriptors
K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms
Security, Management

Keywords
Cloud computing, homomorphic encryption, privacy, secure delegation

1. INTRODUCTION
Cloud Computing is counted among the new advances in the IT world with the potential to make the access to large amounts of computing power easier and cheaper. The idea of near-real time provisioning of services and infrastructures comes from the increasing need of dynamically scalable resources. Customers have the possibility to purchase the resources they require when they require it. This service oriented billing model makes the decision to adopt such services very attractive. Although the advantages of this model are unarguable, there are potential risks involved with the loss of privacy, integrity and availability of the user data due to technical failures or even targeted attacks from both outside and inside the cloud provider's network. The use of these new services could bring security problems to computer networks extended or entirely composed of cloud computing services. Users offload the processing of their critical data to networks that may have different security levels and protection requirements. Hybrid Clouds that include private and public clouds intensify the security concerns of cloud computing customers regarding security guarantees in heterogeneous environments consisting of multiple service providers.

The security measures provided by the cloud providers to achieve the protection goals for customer data processed on their infrastructure are still in an early stage. To address this problem, a number of cryptographic schemes have been developed during the last years. Those schemes typically strive to encapsulate data of different users to provide confidentiality against other users of the cloud. It seems that the field of securing data or code against the cloud provider gets less attention, although it seems quite realistic that intellectual property theft is partly done by or supported by governments that have the power to force a provider to break privacy rules.

We present an approach for secure delegated computation. While the code to be executed in the cloud is unencrypted, data is encrypted before sent to the cloud. Computations on encrypted data are based on two independent homomorphic encryption algorithms. The model consists of an agent acting as the delegator and at least one worker in the cloud who performs the computation. The algorithm can be adopted to different application scenarios.

Although the computation and communication overhead are high for the worker, it still seems worthwhile to investigate whether delegating a complex computation to an untrusted third party might still be of advantage to the delegator itself. We therefore study which application properties influence the overhead of our approach, and how far the overhead must be reduced to allow wider use. Also, we expect that advances in homomorphic encryption will directly aid our protocol.

The remaineder of this article is organised as follows. In Section 2, we introduce the concept of delegated computation based on multiple partly homomorphic public-key cryptosystems. In Section 3, we discuss preliminary runtime measurements and their impact on the usability of the algo- rithm. In Section 4, we give a conclusion.

Table 1: Homomorphic operations

Algorithm	Ciphertext operation	Plaintext operation
RSA	$\cdot$	$\cdot$
Paillier	$\cdot$	$+ \mod m$

2. OUR CONTRIBUTION

The concept of homomorphic encryption was introduced by Rivest et. al. [18] and has been adopted in many cryptographic protocols e.g. zero knowledge protocols [14] and oblivious transfers [11]. Many encryption algorithms supporting either addition or multiplication have been developed e.g. RSA [19], El-Gamal [10] and Paillier [17]. So-called fully-homomorphic-encryption was for a long time "the holy grail" of cryptographers until Craig Gentry announced in 2009 the first fully-homomorphic encryption scheme based on ideal lattice [13]. New developements in fully homomorphic encryptions [4, 5, 6] improve on performance and reduce complexity. Fully homomorphic encryption has been also used in a non interactive verifiable computation scheme [12] enabling a computationally weak client to outsource the computation of functions to one or more untrusted workers employing fully homomorphic encryption combined with verifiability techniques. In order to allow secure outsourcing, many protocols have been developed in the field of secure computation. Atallah et al. present secure outsourcing schemes for specific algebraic computations using Paillier's homomorphic property [2] or secret sharing [1, 21]. Hohenberger et al. [15] provide protocols for secure outsourcing of modular exponentiations used in most public key encryption operations. Wang et al. [22] have recently presented an approach to outsource linear programming using decomposition of the linear programming computation into public solvers running on the cloud and private parameters owned by the customer. These aproaches are well studied in terms of security and complexity and provide special solutions that can be adopted for cloud computing. We present an approach based on asymmetric encryption algorithms to securely outsource integer computations to untrusted cloud providers.

Let x and y be two bit sequences representing the initial plaintexts and $\circ$ a binary operation that can be performed on x and y to obtain $z = x \circ y$. An encryption algorithm is called homomorphic for a defined operation $\circ$, if $(Enc_k(x) \circ Enc_k(y)) = Enc_k(x \circ y)$.
In other words, with a homomorphic encryption, arithmetic operations can be performed on ciphertexts and the respective result can be retrieved after the decryption. Our approach is based on the use of two different asymmetric encryption algorithms, one for addition and one for multiplication, the two basic arithmetic operations.

Let α be an encryption algorithm with a homomorphic property related to addition of integers (denoted by $+$) and β an encryption algorithm with a homomorphic property related to multiplication of integers (denoted by $\cdot$). In the following we use the algorithms from Table 1.

The Delegator (Alice) wants to perform some mathematical computations on a set of confidential data but unfortunately has not enough computation power to perform the complex computation by herself. So Alice decides to delegate the computation to an external party (Bob) with unlimited computation power, who is not trustworthy with respect to confidentiality. To delegate the computation and mitigate her security concerns, Alice wants to provide Bob only with encrypted data. For this reason, she uses the homomorphic properties of the public-key encryption algorithms α and β to achieve this goal.

For x_1 and x_2 two plaintext elements, $r_1, r_2 \in \mathbb{Z}_n^*$ random numbers and $\mathcal{E}$ the encryption function of Paillier with a valid public key (g, n). The additive homomorphic property of Paillier is computed as follows:

$$\begin{aligned}\alpha(x_1) \cdot \alpha(x_2) &= (g^{x_1} r_1^n)(g^{x_2} r_2^n) \mod n^2 \\ &= g^{x_1+x_2}(r_1 r_2)^n \mod n^2 \\ &= \alpha(x_1 + x_2 \mod n)\,.\end{aligned}$$

Given an RSA public key (n, e) and two plaintext elements x_1 and x_2. the multiplicative homomorphic property of RSA is:

$$\begin{aligned}\beta(x_1) \cdot \beta(x_2) &= x_1^e x_2^e \mod n \\ &= (x_1 x_2)^e \mod n \\ &= \beta(x_1 \cdot x_2)\,.\end{aligned}$$

Alice encrypts the initial data according to the operations that will be later performed by the worker and generates a program describing the arithmetic operations. The data and the program are sent to the worker. In the best case the worker can perform the computation without additional interaction with Alice. In other cases, encrypted data can't be processed without a further "transformation" of the encrypted data.

As an example, we consider that Alice wants to compute $c \cdot (a + b)$. Alice computes $A = \alpha(a)$, $B = \alpha(b)$, $C = \beta(c)$ and sends the data and the following computation scheme to the worker:

1. $D = A \cdot B$
2. $D' = update(D)$
3. $E = C \cdot D'$

First, the worker computes $D = A \cdot B$ and sends the result back to Alice in order to update the value. Alice decrypts the result with $\alpha^{-1}(D)$ to get the new value of $d \mod n$ and encrypts it again with $D' = \beta(D)$ and sends D' again to the worker. The worker uses the updated value of D to compute $E = C \cdot D'$ and sends E to Alice. Finally, Alice decrypts E with $\beta^{-1}(E)$ and gets the final result of the computation.

We see from this example that the practical use of our scheme depends on interaction with the delegator in order to switch between addition and multiplication operations on the ciphertext (using re-encryption). The update-operations in our protocol are to a certain extent similar to Gentry's

bootstraping step [13] to refresh "noisy" ciphertexts. The amount of interaction depends largely on the format of the computation scheme and thus strongly affects the complexity of the whole protocol.

Updates partition the computation into instruction sequences preceeded by an encryption and followed by a decryption. A sequence s should be outsourced to the worker if the time to execute s locally is longer than the time for encryption and decryption. Otherwise, the sequence could be executed by the delegator, given that all necessary data are available locally. Similarly, operations on data other than addition or multiplication can be treated locally. It might be advantageous not to treat each sequence in isolation, but consider the overall benefit, even if some sequences are outsourced although they could be executed faster locally.

Thus, program transformations should be applied to push updates as far apart as possible. Computation sequences that are not dependent on further values can be precomputed by the worker and thus reduce the total computation time needed. These measures might be aided by encrypting only part of the data. Variables used only for control flow, such as loop variables, need not be encrypted as the code is not encrypted as well. For other variables, a subset might be chosen for encryption such that a sufficient level of privacy is still achieved while reducing the update frequency as far as possible. The delegator can perform an initial analysis on the computation scheme to optimize its structure according to the proposals above. So far, we focus on reducing the amount of work for the delegator, and ignore increased computational effort in the cloud as well as overhead from network communication.

The security of the proposed protocol depends on the security of the employed cryptosystems, in our case El-Gamal and Paillier, that are "semantically secure" under chosen plaintext attacks and thus ensure the confidentiality of the data during transmission and computation against passive attacks. Nevertheless, we inherit also security weaknesses of those algorithms that can possibly be exploited by malicious attackers in order to gain access to computation data and be able to manipulate them. We consider that many cryptographic algorithms used in our protocol do not preserve their homomorphic properties when padding is used during the encryption phase. This can lead to security weaknesses related to adaptive chosen ciphertext attacks [7, 8, 20]. In our model the worker does not receive the plaintext to encrypted messages hence he would have restricted abilities to perform this kind of attack. Nevertheless, the worker can perform chosen plaintext attacks against the algorithms in use if the respective public keys are known or the attacker is able to request the encryption of chosen plaintext.

An attacker may also try to inject or manipulate packets sent to or from Alice. We recommend adding digital signatures for every computation payload to ensure the authenticity of the transmitted encrypted values. Digital signatures should be generated with dedicated keys in order to avoid security weaknesses that may occur for instance when using RSA and may lead to key attacks allowing to decrypt messages or generate valid digital signatures for arbitrary input. In practice, Transport Layer Security (TLS)[9] might be used

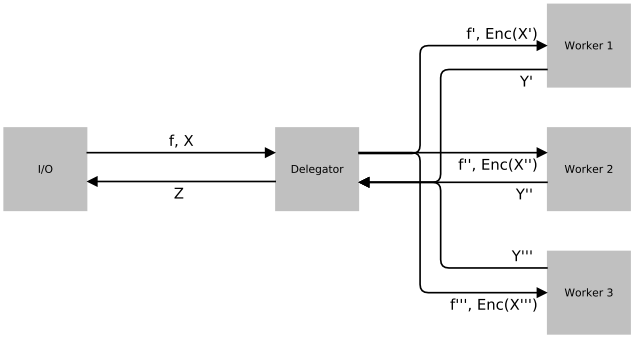

Figure 1: A sample delegation scenario involving three workers. The initial function f is split in to three functions f',f'' and f''' computed by the workers. The results are decrypted and merged.

in order to enhance the integrity and confidentiality of the transmitted data.

Another side effect of the algorithms used that may influence the security of the presented protocol is the so called malleability property of Paillier cryptosystem. Let's consider the case that Alice wants to transfer a message m_1 encrypted to Bob. She computes $Enc(m_1)$ and transfers this value to Bob. If Mallory can wiretap the connection and retrieve the value of $Enc(m_1)$ he will be able to compose valid ciphertext using known variables from Alice's public key:

$$Dec(Enc(m_1).g^{m_2} \mod n) = m_1 + m_2 \mod n$$

The Malleability may be exploited by an attacker in order to inject valid data in computation streams if the public key used is known. In our protocol architecture the worker isn't intended to generate encrypted communication using the delegator's public keys. To avoid this kind of theoretical attack, Alice can keep both parts of Paillier's asymmetric keys secret.

3. PROTOCOL IMPLEMENTATION ON X86 CPU

Our prototype implementation of the protocol is written in C/C++. The protocol is implemented in user mode and does not require any changes to the operation system. The implementation performs operations on integers of arbitrary size based on big number operations. The worker implementation is based on a network server accepting raw data connections including variables in encrypted form. The server doesn't require client authentication.

The delegator is implemented using an interpreter that translates the source code with arithmetic operations into a homomorphic operation on encrypted operands. The translation can be achieved in different ways. New applications can adopt our protocol in their source code to perform secure operations on cloud computing networks. Existing applications might be extended by the functionality of our

protocol through patching of binaries, see below. In our proof of concept the translation is achieved using generic type template functions `Htype homomorphic_add (Htype, Htype)` and `Htype homomorphic_multiply(Htype, Htype)` that perform homomorphic operations on different numeric data types. The implementation allows deterministic arithmetic computation enabling execution state saving. The template functions for addition and multiplication replace the operators for addition and multiplication with function- The functions allocate memory for the argument variables that will be respectively encrypted and transmitted to the workers. The values received are classified and decrypted according to the computation scheme.

In the assembly implementation we use dynamically `.ascii` fixed length strings and `.string` variable length strings to store big number values of arbitrary size. An existing application is read in binary mode and disassembled to assembler code which can be patched with assembly functions that perform the translations between variable registers and their corresponding encrypted values.

As an example for the delegated computation we use a program that delegates the polynomial evaluation for big integer numbers. The test programs used are written in C / C++ and use a native implementation of the protocol on an x86-64 architecture. After compiling and building the programs, the delegator transfers the client program to the cloud in order to perform the computation. The delegator program initially performs a basic analysis of the computations that should be outsourced to the cloud. Based on this, the program generates one or more computation schemes that can be externally computed by the worker. The corresponding variables are encrypted and sent together with the computation scheme to one or more workers. The results of the different computations are consolidated in order to get the final result or to flow as input for another depending computation scheme. We note that in this operation mode the operations performed on the encrypted data must be disclosed to the worker. A rogue worker may be able to identify critical data based on the operations performed on it.

Besides the operation mode described above we propose another operation mode called "anonymous" that enables the delegator to perform homomorphic operations without disclosing the arithmetic operations performed on the data. We exploit the fact that the Paillier and RSA homomorphic properties are achieved with the multiplication of ciphertext values. In this mode, the worker has no possibility to distinguish between addition and multiplication. This operation mode has the negative side effect that operation results must be sent back to the delegator after each operation. This leads to increased network traffic and time overhead.

Figure 3 presents the time needed by a CPU in order to perform a homomorphic operation or a translation between two homomorphic operations on an operand of size n. The graph figures the average time for the operations simulated on two systems with Intel Core2 Duo P8700 CPU with 2.53GHz and Intel Core i5 M520 CPU with 2.40GHz connected via local network (100-Mbit/s-Ethernet).

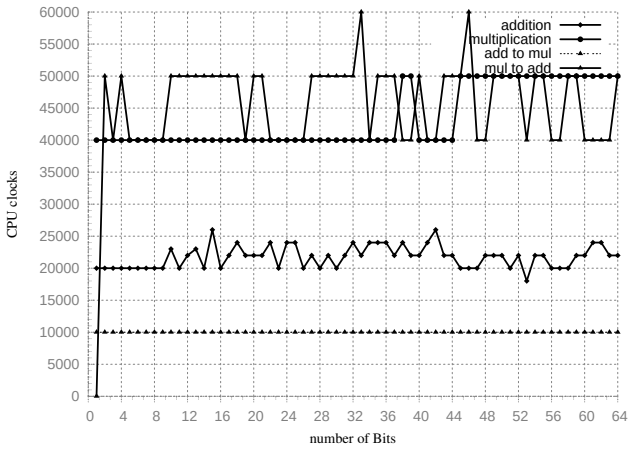

Figure 2: Benchmark of the individual basic operation's runtime.

We note that the individual operations involving RSA encryption lead generally to increased computation overhead compared to addition or update operations from addition to multiplication.

In Figure 3 we present a visualisation of our polynomial evaluation example using random 64 bits values. The implementation doesnâĂŹt use any optimisations for Paillier or RSA such as Montgomery arithmetic [3] or caching of ciphertext independent factors. We observe that the runtime is linear in the number of terms and thus in the underlying number of operations.

While we recognize that outsourcing leads to more work for the delegator in our toy example, Figure 2 indicates that updates must only be apart by 10^4 instructions to achieve a reduction, which might for example occur in computations where large numbers of additions are executed on a set of data, such as computing different averages of measured data. Also, the amount of work necessary to make the scheme practical can be reduced as the delegator could implement efficient data structures (e.g. rainbow chains [16]) to save precomputed values and their appropriate encryption strings with the algorithms in use. Furthermore, although we presented our protocol only for one worker, this protocol can be easily extended to any suitable number of workers given sufficient independent computation on the delegator side. The computation might be split, and the results reassembled by the delegator, cf. Figure 3.

4. CONCLUSION

We presented an approach to exploit homomorphic properties of asymmetric encryption algorithms to increase the security and the trust level in today's cloud based services. The experimental results indicate that the computation overhead can be reduced so far that the scheme is applicable for practical use. The presented protocol can be implemented in application source code. Existing applications with no possibility to modify the source code might be extended by the functionality through software patches of the binary executables, although the possibility to reduce the number of

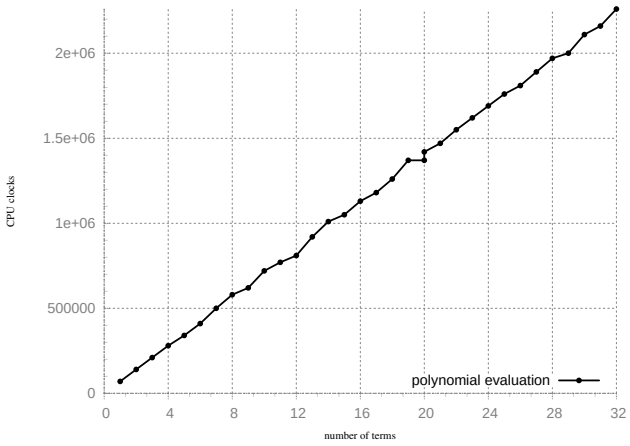

Figure 3: CPU time needed to perform polynomial evaluation for polynomials of different degrees and random values.

transfers is strongly reduced.

5. REFERENCES

[1] M. J. Atallah and K. B. Frikken. Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 48–59, New York, NY, USA, 2010. ACM.

[2] D. Benjamin and M. J. Atallah. Private and cheating-free outsourcing of algebraic computations. In *Proceedings of the 2008 Sixth Annual Conference on Privacy, Security and Trust*, PST '08, pages 240–245, Washington, DC, USA, 2008. IEEE Computer Society.

[3] K. Bentahar and N. P. Smart. Efficient 15,360-bit rsa using woop-optimised montgomery arithmetic. In *Proceedings of the 11th IMA international conference on Cryptography and coding*, Cryptography and Coding'07, pages 346–363, 2007.

[4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[5] Z. Brakerski and V. 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] J.-S. Coron, D. Naccache, and M. Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 446–464, 2012.

[7] G. Davida. Chosen signature cryptanalysis of the rsa public key cryptosystem. Technical Report TR-CS-82-2, Dept of EECS, University of Wisconsin, Milwaukee, 1982.

[8] Y. Desmedt and A. M. Odlyzko. A chosen text attack on the rsa cryptosystem and some discrete logarithm schemes. In *Advances in Cryptology*, CRYPTO '85, pages 516–522, 1986.

[9] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), 2008.

[10] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, 1985.

[11] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.

[12] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 465–482, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, 2009.

[14] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, Feb. 1989.

[15] S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *TCC*, pages 264–282, 2005.

[16] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 617–630, 2003.

[17] P. 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, 1999.

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

[19] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.

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

[21] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

[22] C. Wang, K. Ren, and J. Wang. Secure and practical outsourcing of linear programming in cloud computing. *Computer Engineering*, pages 820–828, 2011.