Computation on Randomized Data

Florian Kerschbaum
SAP Research Karlsruhe, Germany
florian.kerschbaum@sap.com

Abstract

Cryptographic tools, such as secure computation or homomorphic encryption, are very computationally expensive. This makes their use for confidentiality protection of client's data against an untrusted service provider uneconomical in most applications of cloud computing. In this paper we present techniques for randomizing data using light-weight operations and then securely outsourcing the computation to a server. We discuss how to formally assess the security of our approach and present linear programming as a case study.

1 Introduction

Whit Diffie is commonly quoted with "cryptography will not be the solution for security of cloud computing" due to economic reasons [9]. Cryptographic tools, such as secure computation [6, 11, 23] or homomorphic encryption [10, 17] are computationally expensive. Recent results [5, 12, 14, 18] indicate that while they are feasible for small computations they are orders of magnitude slower than non-secure computations. The general argument is then that non-secure, local computation – even on computational weak devices such as mobile phones – is more economical than cryptographically protected cloud computing.

In this paper we present an alternative approach based on randomization of input data. The clients compute a lightweight function $y = f(x, r)$ on their input x using random coin tosses r. The output y is sent to the cloud service provider who computes – the usually unmodified – function $z = g(y)$ and returns it to the client. The client then de-randomizes using the function $f'(z, r)$. The idea is that for correctness

$$f'(g(f(x, r)), r) = g(x)$$

Two performance objectives should be observed. First, the function f should be easy to compute. We propose as a design principle to only use lightweight randomization operations. For scalar data only the following operations should be used:

- addition of random numbers
- multiplication with random numbers

For vector data only the following operations should be used:

- permutation
- creation of new elements
- splitting of elements

All these operations can be performed much faster on current hardware than operations commonly used in cryptographic protection, such as modular exponentiation.

The second performance objective is that computation of the function g should be remain efficient. Ideally the function and consequently its complexity should remain unchanged to non-secure computation. Even a constant increase in computational cost should be modest.

We observe a fundamental difference to cryptographic protection, such as secure computation and homomorphic encryption. In the cryptographic methods the computation is expressed as a – Boolean or arithmetic – circuit and then executed. The circuit nevertheless needs to be oblivious, i.e. its execution needs to be independent of its input. Each gate of the circuit needs to be executed once and only once. This entails that the complexity of the circuit is at least the worst-case complexity of the encoded algorithm. Nevertheless, there are many algorithms which have very good average case complexity, but bad worst case complexity. An instance is our use case of Simplex, which is exponential in the worst case, but $O(n^3)$ on average [19]. There are many more instances for the class of NP-hard problems.

Even if the cryptographic algorithms were implemented in hardware and executed at clock speed, without overcoming this design many algorithms will remain uneconomical under cryptographic protection. Our method of unchanged computation may, of course, take advantage of average case complexity.

Ultimately this approach should yield a performance increase of several orders of magnitude. With the method we present in Section 2 the performance increase is more than 100.000. This performance increase comes at the expense of security. We discuss how to assess security in Section 3. Related work is reviewed in Section 4. We present the research challenges we foresee for our approach in Section 5 concluding the paper.

2 Example: Linear Programming

As a use case we present the example of Linear Programming. Linear Programming is a standard tool in business optimization. A Linear Program (LP) consists of a set of unknown variables x, a linear target function $c(x)$ representing the costs which shall be minimized (or equivalently the gain which has to be maximized) and a set of constraints (linear equalities or inequalities). As we will eventually transform all inequalities into equalities and as in most business applications many of the input constraints are actually equalities, we will treat them separately. This reduces the size of the problem as we need less slack-variables. A slack-variable is used to express inequality constraints as equality constraints. The idea is to introduce an additional variable for each constraint which takes up the "remainder" or "slack". So let the input problem - without loss of generality - be

$$\begin{array}{rcl} \min & c^T x & \\ \text{s.t. } M_1 x & = & b_1 \\ M_2 x & \leq & b_2 \\ x & \geq & 0 \end{array}$$

We use a positive monomial matrix Q to hide c (as proposed by [22, 4]). A monomial matrix contains exactly one non-zero entry per row and column. This corresponds to our randomization operations of multiplication with a random number and permutation.

$$\begin{array}{rcl} \min c^T Q Q^{-1} x & & \\ M_1 Q Q^{-1} x & = & b_1 \\ M_2 Q Q^{-1} x & \leq & b_2 \\ Q^{-1} x & \geq & 0 \end{array}$$

We also use a positive vector r to hide x. This corresponds to our randomization operation of adding a random number.

$$\begin{array}{rcl} \min c^T Q (Q^{-1} x + r) & & \\ M_1 Q (Q^{-1} x + r) & = & b_1 + M_1 Q r \\ M_2 Q (Q^{-1} x + r) & \leq & b_2 + M_2 Q r \\ (Q^{-1} x + r) & \geq & r \end{array}$$

For $z = Q^{-1} x + r$ and a strictly positive diagonal matrix S we have

$$\begin{array}{rcl} \min c^T Q z & & \\ M_1 Q z & = & b_1 + M_1 Q r \\ M_2 Q z & \leq & b_2 + M_2 Q r \\ S z & \geq & S r \end{array}$$

Then we define $c'^T = c^T Q$,

$$M' = \begin{pmatrix} M_1Q & 0 \\ M_2Q & \mathrm{A} \\ -S & \end{pmatrix} \text{ and } b' = \begin{pmatrix} b_1 + M_1Qr \\ b_2 + M_2Qr \\ -Sr \end{pmatrix}$$

where A is a permutation matrix representing slack-variables. This allows us to rewrite the program as follows:

$$\begin{array}{rcl} \min & c_s'^T z_s & \\ \text{s.t. } M'z_s & = & b' \\ z_s & \geq & 0 \end{array}$$

where c'_s is c' with added zeros for the slack-variables and z_s is the variable vector (z with added slack-variables). To hide the contents of M' and b' we use a nonsingular (invertible) matrix P and with $M'' = P{*}M'$ and $b'' = P' * b'$ we have

$$\begin{array}{rcl} \min & c_s'^T z & \\ \text{s.t. } M''z_s & = & b'' \\ z_s & \geq & 0 \end{array}$$

This transformed LP can then be sent to the cloud service provider. It can still be solved using the same algorithms as any LP, but it hides the input LP and its optimized result.

The cloud service provider returns the result z to the transformed LP. As $z = Q^{-1}x + r$, the result x of the original LP can be obtained from z by calculating $x = Q(z - r)$.

3 Security

Before evaluating the security of our approach, we highlight its enormous performance improvement potential. When running a secure computation for Linear Programming by Toft [21] we estimate the running time for 282 variable textbook problem to be roughly 7 years for 3 servers. When running it using our protocol the running is roughly 6 seconds for a single client. A more than 10^7 time increase in performance. Clearly this brings new types of problems into the practically feasible range and makes secure cloud computing a viable alternative.

In secure computation we distinguish between the ideal model where all inputs are sent to a trusted server and the real model of executing the protocol. The ideal model in cloud computing would be a client and a cloud service provider where the client submits data to the trusted server and receives the output. The cloud service provider remains oblivious despite playing an important role in the real model. This could be implemented using secure two-party computation. Also multi-party computation could be used by splitting the cloud service provider into mutually distrustful entities.

The basic security model of secure computation is semi-honest security. In semi-honest security the view $VIEW_f^{IDEAL}$ for a function f in the ideal model is compared to the view $VIEW_p^{REAL}$ for a protocol p in the real model. We call a protocol secure in the semi-honest model if those two views are indistinguishable.

Cryptographers distinguish between different types of security. In the information-theoretic setting with perfect security [6] the views are statistically indistinguishable without security parameter.

$$VIEW_f^{IDEAL} \stackrel{s}{=} VIEW_p^{REAL}$$

In the computational setting [11] the views are computationally indistinguishable negligible in a security parameter k.

$$VIEW_f^{IDEAL} \stackrel{c}{=} VIEW_p^{REAL} + \frac{1}{poly(k)}$$

In our setting the views are information-theoretically comparable, but not indistinguishable in the traditional sense. The transformed problem allows a partial inference about the input depending on the amount

of randomness and its security parameter k. Furthermore, this inference is no longer negligible in k, but polynomial in k.

$$VIEW_f^{IDEAL} \stackrel{s}{=} \frac{1}{poly(k)} \cdot VIEW_p^{REAL}$$

Different from security negligible in the security parameter, determining the polynomial $poly(k)$ now becomes quite important. We propose to use the method of leakage quantification [7, 20]. The adversary knows the transform and the distribution of the input variable X. He is observing the transformed variable $Y = T(X)$ and then tries to guess X. Intuitively, the leakage $\mathcal{L}_X(T)$ of T with respect to X measures how much observing the output of T increases the ability of the adversary to guess the input. It is defined as the ratio between the ability of the adversary to guess the input before observing the transformed input and after observing the transformed input. The probability that the adversary can guess the right input without observing the transformed input (the a priori probability of a right guess) is

$$\mathrm{PR}_{priori}(X) = \max_{x \in \mathcal{X}} p(X = x)$$

The probability that the adversary can guess the right input after observing the transformed input (the a posteriori probability of a right guess) is

$$\mathrm{PR}_{posteriori}(X) = \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} (p(X = x)p(y|x))$$

Then the leakage of T with respect to X is defined as

$$\mathcal{L}_X(T) = \frac{\mathrm{PR}_{posteriori}(X)}{\mathrm{PR}_{priori}(X)}.$$

This is referred to as the multiplicative leakage in [7].

We define the leakage $\mathcal{L}(T)$ of a transform T as the supremum of the leakages with respect to X, where $X \in \mathrm{Var}(\mathcal{X})$ runs over all random variables independent from the transform.

$$\mathcal{L}(T) = \sup_X \mathcal{L}_X(T).$$

As it was already observed in [7] one has

$$\mathcal{L}(T) = \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} p(y|x)$$

for uniformly distributed X.

For transform for Linear Programming in Section 2 we establish the following bounds:

$$L(c) \leq \min \left(\left(\frac{n+l}{l+1} + 1 \right)^k, \frac{(A(n,l)+k)!}{k! * (A(n,l))!} \right)$$

$$L(z) \leq \min \left(2\log(x_m + 2), \frac{(x_m + k)!}{k! * x_m!} \right)$$

$$L(M) \leq \frac{\prod_{i=1}^{k} ((n+1)^{k+l} - i)}{k!}$$

4 Related Work

Similar ideas to ours have been proposed a long time before cloud computing. In [3] several methods for outsourcing scientific computations are proposed. Atallah et al. already propose similar transformations (including non-linear ones for non-linear problems) and describe a number of algorithms. They call their transforms "disguise" in order to distinguish them from encryption and consider their same parameters – performance and security. We differ by our method of evaluation. First, we implement and measure the running time of the transforms, such that comparisons between different operations of the same complexity are possible. And second, we use the more formal framework of leakage quantification in order to assess the security of our transforms.

Later additional secure outsourcing techniques for different problems have been added [1, 2]. In [1] a method for sequence comparison using two mutually distrustful servers has been proposed. In [2] a provably secure technique outsourcing matrix multiplication is described.

A transform for secure linear programming was long sought after [4, 22, 15, 16]. The proposal in [22] has been proven incorrect in [4]. Similar approaches, but less general than ours, have been made in [15, 16].

Secure linear programming has also been considered in the cryptographic setting [8, 13, 21]. A multi-party solution has been proposed in [21] and a two-party solution in [13]. An efficiency improvement for smaller multi-party problems has been made in [8]. As our implementation results show these proposals are not economical for cloud deployment.

5 Conclusions and Future Work

We have shown that using simple randomizations it is possible to transform an input, such that it is still possible to execute the same algorithm for obtaining the transformed result. Our implementation results furthermore underpin the enormous performance improvement potential compared to cryptographic techniques. We also made the first step towards a formal assessment of security of such transforms.

This paper is supposed to initiate a discussion and is far from complete work. We have made some initial steps and hope to gather some feedback before proceeding.

Two major challenges remain: The first challenge is an automatic derivation algorithm for a transform given a function f. All approaches so far – including this paper – are somewhat ad-hoc, in the sense that they manually analyze the function f and then propose a transform T. Ideally, as in secure computation, there should be an algorithm that given f outputs T. For secure computation, such algorithm have been implemented, e.g. FairPlay [14] and FairPlayMP [5]. There f is specified in a programming language and then translated into a circuit.

It is not quite clear whether a specification in a traditional programming language is sufficient for devising a secure transform as proposed in this paper. It rather seems that these transforms are designed around invariants of the function that are hard to deduce from the program text. E.g. all randomizations applied in our linear programming transform have the property that they do not change the minimum solution except by a known constant.

The second challenge is an automatic security assessment for a given T. The leakage bounds we established in Section 2 are all derived and proven manually. Ideally there should be an algorithm – a logic – that given T outputs or at least verifies these bounds. There are several approaches for combining formal methods with cryptographic assumptions and these produce wonderful results. A similar approach should exists for randomized transforms.

Given a performance model of the transforms it would then be possible to tune a derivation algorithm, such that it optimizes the performance vs. security trade-off. The cloud consumer could choose its appropriate level and a resulting transform would be derived. This gives the cloud consumer even more choices and the opportunity to match the business and security needs.

References

[1] M. Atallah, and K. Frikken. Securely Outsourcing Linear Algebra Computations. *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, 2010.

[2] M. Atallah, and J. Li. Secure outsourcing of sequence comparisons *International Journal of Information Security 4(4)*, 2005.

[3] M. Atallah, K. Pantazopoulos, J. Rice, and E. Spafford. Secure Outsourcing of Scientific Computations. *Advances in Computers*, 2002.

[4] A. Bednarz, N. Bean, and M. Roughan. Hiccups on the Road to Privacy-Preserving Linear Programming. *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, 2009.

[5] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: A System for Secure Multi-Party Computation. *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2008.

[6] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. *Proceedings of the 20th ACM Symposium on Theory of Computing*, 1988.

[7] C. Braun, K. Chatzikokoladis, and C. Palamidessi. Quantitative Notions of Leakage for One-Try Attacks. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 2009.

[8] O. Catrina, and S. de Hoogh. Secure Multiparty Linear Programming Using Fixed-Point Arithmetic. *Proceedings of the 15th European Symposium on Research in Computer Security*, 2010.

[9] Y. Chen, and R. Sion. On Securing Untrusted Clouds with Cryptography. *Proceedings of the 9th ACM Workshop on Privacy in the Electronic Society*, 2010.

[10] C. Gentry. Fully Homomorphic Encryption using Ideal Lattices. *Proceedings of the 41st ACM Symposium on Theory of Computing*, 2009.

[11] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game. *Proceedings of the 19th ACM Symposium on Theory of Computing*, 1987.

[12] F. Kerschbaum, D. Dahlmeier, A. Schröpfer, and D. Biswas. On the Practical Importance of Communication Complexity for Secure Multi-Party Computation Protocols. *Proceedings of the 24th ACM Symposium on Applied Computing*, 2009.

[13] J. Li, and M. Atallah. Secure and Private Collaborative Linear Programming. *Proceedings of the 2nd IEEE Conference on Collaborative Computing*, 2006.

[14] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - A Secure Two-party Computation System. *Proceedings of the 13th USENIX Security Symposium*, 2004.

[15] O. Mangasarian. Privacy-Preserving Linear Programming. *Data Mining Institute Technical Report 10-01*, Available at *ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/10-01.pdf*, 2010.

[16] O. Mangasarian. Privacy-Preserving Horizontally-Partitioned Linear Programs. *Data Mining Institute Technical Report 10-02*, Available at *ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/10-02.pdf*, 2010.

[17] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Proceedings of EUROCRYPT*, 1999.

[18] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure Two-Party Computation is Practical. *Proceedings of ASIACRYPT*, 2009.

[19] G. Sierksma. Linear and Integer Programming: Theory and Practice. *Marcel Dekker, Inc.*, 1996.

[20] G. Smith. On the Foundations of Quantitative Information Flow. *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures*, 2009.

[21] T. Toft. Solving Linear Programs using Multiparty Computation. *Proceedings of the 13th International Conference on Financial Cryptography and Data Security*, 2009.

[22] J. Vaidya. Privacy-Preserving Linear Programming. *Proceedings of the 24th ACM Symposium on Applied Computing*, 2009.

[23] A. Yao. Protocols for Secure Computations. *Proceedings of the 14th IEEE Symposium on Foundations of Computer Science*, 1982.