Dikē: A Prototype for Secure Delegation of Statistical Data

Javier Castilla
Institut d'Estadística de Catalunya
Josep Domingo-Ferrer
Universitat Rovira i Virgili
Ricardo X. Sánchez del Castillo
Universitat de Barcelona[*†]

Abstract

The need for delegating statistical data arises when the data owner (*e.g.* statistical office) wants to have its data handled by an external party. If the external party is untrusted and data are confidential, delegation should be performed in a way that preserves security. A cryptographic solution to the secure delegation problem is outlined which provides data secrecy and computation verifiability. Also, the design principles of Dikē —an implementation allowing secure delegation of information over the Internet— are discussed in some detail.

Keywords: Delegation of information; Encrypted data processing; Distributed computing; Statistical data protection.

1 Introduction

In many scenarios, statistical data cannot be processed where they originate or belong to. In such cases, the data owner must transfer data to a remote environment (the handler) for processing. If data are confidential, a security problem arises. A legal solution to this problem is to require that the handler sign a non-disclosure agreement. For example, this is the procedure followed when some government agencies release data to universities for research purposes.

The above legal solution has a serious drawback: the data owner must *believe* that the handler is fair, since there is no technical means to prevent data misuse. In this paper, we describe an implementation that allows the data owner to control and limit the kind of operations performed by the handler. Depending on who is interested in the results of the data processing, two types of delegation scenarios can be distinguished:

Computing delegation The party interested in the result of the data processing is the data owner.

Data delegation The party interested in the result of the data processing is the handler.

In the solution described below, *the work performed by the handler depends on the nature of the processing to be done, whereas the work done by the data owner is fairly independent of the processing* (in fact, the data owner functionality could be implemented in hardware). We illustrate next two practical applications.

[*]*Address for correspondence:* R. X. Sánchez del Castillo, Universitat de Barcelona, Servei d'Informàtica, Trav. Corts 131-159 (Pavelló Rosa), E-08028 Barcelona, Catalonia, Spain. E-mail ricardo@dalila.ird.ub.es

[†]This work is partly supported by the Spanish CICYT under grant no. TIC95-0903-C02-02 and by the Statistical Institute of Catalonia under contract no. FBG-2577

Example 1 A computing delegation problem happens whenever a (small) company wants to use external computing facilities to do some calculations on corporate confidential data. A very common variant of this situation is a medical research team using a (insecure) university mainframe for processing confidential healthcare records. The reason for using external facilities may be the complexity of the calculations but also the huge volume of the data set. □

Example 2 Data delegation problems appear in the interaction between public administrations at several levels. For example, municipalities cooperate with national statistical institutes in statistical data collection. In return, municipalities would like to be able to analyze the whole collected data set (pooled from all municipalities). But only national statistical institutes are usually authorized to hold nation-wide individual census data. A similar problem occurs in any federal-like structure (European Union, U.S.A., Germany, etc.). Member states cooperate with federal agencies in collecting data from individuals, companies, etc. In return, states would like to be able to analyze data at a federal level. A secure solution in both scenarios above is for the organization owning the whole data set to perform (probably for free) the analyses requested by the cooperating organizations. But then the data owning organization becomes a bottleneck and is forced to waste time and resources in uninteresting tasks. A better solution would be for the data owner to delegate data in a secure way and reduce its role in subsequent analyses to a minimum. □

Section 2 outlines a cryptographic solution to the secure delegation problem. Section 3 gives an overview of a prototype called Dikē (blind Greek goddess of justice and social order and also Delegation of Information without Knowledge Exposure, [1]). The architecture of Dikē is further specified in section 4. Section 5 contains a conclusion and some lines of future research.

2 A cryptographic solution to the secure delegation problem

In [4] a cryptographic solution to the secure delegation problem is specified. Its basic concepts will be briefly recalled and some of its properties will be proven here for the first time. Two requirements must be met by secure delegation: first, the data owner must be assured that data remain secret (*data secrecy*) and, second, the data owner must be able to verify that the computation carried out by the handler is correct (*computation verifiability*).

In order for the handler to compute without being revealed the input data, computation is to be carried out on encrypted data. Encryption transformations allowing some operations to be carried out directly on the encrypted data are known as privacy homomorphisms (PHs for short, see [7]). Two PHs are currently implemented in Dikē: RSA [8] and a PH described in [3], which be denoted by JD. RSA allows multiplication and test for equality to be carried out on encrypted data. JD allows full arithmetic (addition/subtraction, multiplication and "fraction division") on encrypted data.

Full computation verifiability would ultimately require the data owner to repeat the whole handler computation. Since this is impractical, only probabilistic verification is practical. The choice for Dikē is parity checking. The idea is that arithmetical operations can be mapped to Boolean operations on data parities; the parities of $a+b$ and ab can be computed as

$$Z(a+b) = Z(a) \oplus Z(b); \quad Z(ab) = Z(a) \cdot Z(b) \tag{1}$$

where $\oplus$ denotes exclusive OR and $\cdot$ denotes logical AND. Now, let the *claimed expression* the expression that the handler claims to have computed on the data; let the *computed expression* be the expression actually computed by the handler. Upon completing a computation on encrypted data, the handler returns the (encrypted) result together with a claimed expression to the data owner. The data owner can decrypt the result and obtain its parity; on the other hand, the data owner can also quickly compute the parity resulting from feeding the input data to the claimed expression (only quick Boolean operations are needed). If both parities differ, then handler fraud has been detected.

The following properties of the parity checking mechanism were stated in [4] without proof and are proven here:

Lemma 1 (Fraud detection probability) *If the claimed and the computed expressions differ, the probability of fraud detection by the owner is*

$$P(\mathit{detect}) = p(1-p') + p'(1-p) \tag{2}$$

where p and p' are, respectively, the probabilities of the claimed and computed expressions being odd.

Proof : For both the claimed expression and the computed expression do:

1. Consider the parity formula of the expression, which yields the parity of the result by using XOR and AND gates on the parities of input data. Perform Boolean reduction on the formula so that it can be represented as a tree where the root is the parity of the result, leaves are parities of input data, nodes are two-input logical gates and for each node, both inputs are independent from each other. Let the estimated parity distribution for an input datum d be $(1-p_d, p_d)$ where p_d is the probability of d being odd.

2. Starting from the leaves, compute the parity distribution of every node output o (intermediate result) in the tree from the distributions of node inputs a and b, using the following rules derived from equations (1)

   XOR nodes: $p_o = p_a(1-p_b) + (1-p_a)p_b$

   AND nodes: $p_o = p_a p_b$

   OR nodes (may appear in the tree as a result of Boolean reduction): $p_o = p_a + p_b - p_a p_b$

If parity distributions of the results of the claimed and computed expressions are $(1-p, p)$ and $(1-p', p')$, the probability of the handler being caught is

$$P(\text{detect}) = P(\text{odd claimed})P(\text{even computed})$$

$$+P(\text{even claimed})P(\text{odd computed}) = p(1-p') + p'(1-p)$$

□

Theorem 1 *For random input data, the probability of fraud detection by parity checking of the final result is tightly upper-bounded by 1/2.*

Proof : If input data are random, they have parity distribution $(1/2, 1/2)$. Additions introduce no parity bias but results of multiplications tend to be even (only odd times odd yields odd). This means $p \leq 1/2$ and $p' \leq 1/2$ in equation (2). Thus $P(\text{detect}) \leq 1/2$. If the claimed and the computed expressions contain only additions, then $p = p' = 1/2$ and $P(\text{detect}) = 1/2$. So the bound is tight. □

Theorem 2 *For random input data, the probability of fraud detection by parity checking of the final result is tightly lower-bounded by* $\max(p, p')$*, where p and p' are, respectively, the probabilities of the claimed and computed expressions being odd. If always-even claimed expressions are forbidden by the data owner, nonzero detection probability is guaranteed.*

Proof : The probability of detection is given by equation (2). We first show that it is greater or equal than p'. We have

$$P(\text{detect}) = p(1-p') + p'(1-p) = p' + p(1-2p')$$

So, for fixed p' the probability of detection can be regarded as a straight line which is a function of p. If input data are random, from the proof of theorem 1 one has $p' \leq 1/2$, so that the slope of the above straight line is $1-2p' \geq 0$. Since $p \geq 0$, it holds that $P(\text{detect}) \geq p'$. Now exchanging p and p' in the above argument, we obtain $P(\text{detect}) \geq p$, which proves the lower bound. Since the bounds on p and p' given in the proof of theorem 1 are tight, so is the bound on $P(\text{detect})$. Finally, if always-even claimed expressions are forbidden, then $p > 0$ and therefore $P(\text{detect}) > 0$. $\square$

Note 1 *Even if always-even claimed expressions are forbidden, in data delegation the handler can fabricate for every $\epsilon > 0$ a claimed expression such that $0 < p < \epsilon$. But, for fraud to make sense, the computed expression is expected to be chosen on the basis of its usefulness to the handler, which means that p' takes a fixed value. Therefore, the lower bound* $\max(p, p')$ *—and consequently the detection probability— cannot be made arbitrarily small by the handler if the computed expression is to remain useful.*

To recognize a (forbidden) always-even expression, the data owner must check whether the associated Boolean parity formula is equivalent to 0. This check is very easy if the claimed expression is required to be in sum-of-products (SOP) form, where it amounts to checking that each AND term contains some input variable d and its complemented $\bar{d}$. Note that it is trivial for the owner to make sure that the Boolean formula provided by the handler is in SOP form.

We conclude that the data owner can be assured of a nonzero detection probability but is *unable to compute $P(\text{detect})$ nor the lower bound of theorem 2* because she does not know p'. On the other hand, theorem 1 gives an upper bound on the confidence attainable if verification of the final result passes. A way for the owner to increase the probability of detection (maybe above 1/2) is to verify several independent intermediate results. For example, if the claimed expression is regarded as a binary tree having input data as leaves and the result as root, then the owner may request from the handler the two encrypted intermediate results preceding the final result in the tree. If detection probabilities for such results are p_l and p_r, then the probability of fraud detection is $p_l + p_r - p_l p_r$. The procedure can be iterated by backtracking up the tree: if the owner requests also the four encrypted intermediate results preceding the previous two ones, then the probability of detection increases further. A trade-off is obvious: increasing the detection probability entails more verification work for the data owner.

3 Overview of Dikē

From section 1, it turns out that delegation can be viewed as a client-server problem. The data owner plays the role of server (since it has the data) and the data handler plays the role of client. In this context, it becomes clear that Dikē must be a distributed application, possibly over the Internet. More specifically, three basic components are considered, where each component may correspond to a different machine or to a different process:

Server It contains the statistical data, the users, the log files of computations and the security information related to each user and each table.

Client It contains the instances of statistical data authorized by the server for a given user. The client also contains the public security parameters which enable it to carry out operations on encrypted data.

Administrator It is responsible for managing the association between users, data and security parameters.

The statistical data in Dikē are treated as tables. A microdata set can be viewed as a table where rows correspond to individuals and columns correspond to variables. A macrodata set or contingency table of variables A and B, where A takes values $a_1, \cdots, a_m$ and B takes values $b_1, \cdots, b_n$, can be viewed a a set of rows (a_i, b_j, f_{ij}), where f_{ij} is the joint frequency of values a_i and b_j. This set of rows is actually a table with $m \times n$ rows and 3 columns. Columns in a table can be encrypted under the RSA homomorphism (typically when the column corresponds to a qualitative variable), under the JD homomorphism (for columns corresponding to quantitative variables) or can be left in the clear.

For a client to access a server, the client must be associated to a user. The server associates each user with several *security sets*. Each security set is linked to a statistical table: in fact, the security set consists of a key for each privacy homomorphism (RSA, JD), a reference to an encrypted version of the table (some columns may be encrypted under RSA and some under JD) and a reference to the clear table.

Dikē uses the CORBA [6][9] distributed computing standard to describe and implement its main components. The implementation consists of two levels:

- The first level is the library for handling encrypted data, *i. e.* the library implementing the privacy homomorphisms RSA and JD. This library is written in C++, is based on the arithmetical library LiDIA [5]. A PH needs a method for encryption, a method for decryption and a method for each operation that can be performed on encrypted data.
- The second level is the visible part of CORBA, *i. e.* the interface of components. CORBA IDL (Interface Definition Language) is used to write interfaces.

4 System architecture

Figure 1 represents the scenario of interaction between the three top-level components of Dikē. Initially, the Client, the Administrator and the Server contain, respectively, an object of class StatClient, an object of class StatAdmin and an object of class StatServer.

For each StatClient that logs into StatServer, StatServer allocates an object StatServerAccess; all subsequent interaction takes place between StatClient and StatServerAccess. Each time StatAdmin accesses StatServer, StatServer allocates an object StatServerAdmin; all subsequent interaction takes place between StatAdmin and StatServerAdmin.

4.1 Structure of StatServer

The structure of StatServer is depicted in figure 2 using the notation described in [2]. Boxes represent object types. A line between two boxes represents a relationship between two object types. A line ending with a $\diamond$ means that the object type at the beginning of the line is part of the object type at the end of the line. When a line represents a one-to-many relationship, a $\bullet$ indicates the “many” end of the line.

Initially, StatServer consists of a UserServer, a SecurityServer, a TableServer and a LogServer. Each time a Client logs into StatServer, an instance of StatServerAccess is created. When the Administrator logs into StatServer, an instance of StatServerAdmin is created; note that there may be only one instance of StatServerAdmin at the same time, *i. e.* only one Administrator may be manipulating the Server.

UserServer is responsible for the management of users of Dikē. Its main functions are:

- Control that only one client per user identity can access the StatServer at the same time.
- Implement the user identification and table access authorization by maintaining an access matrix that specifies which data tables can be accessed by each user.

Figure 1: Client-server-administrator interaction

Figure 2: StatServer structure

Figure 3: StatClient structure

SecurityServer manages the security sets, *i. e.* the association between users and tables. Such management involves creating and deleting security sets as well as extracting a key or another parameter from a security set.

TableServer is responsible for the management of tables in the server. It contains tables and its interface consists of operations to: add and remove tables; add and remove columns from a table; retrieve a table.

LogServer logs all requests to decrypt encrypted results obtained from encrypted data. Each log entry contains a reference to the user requesting the decryption, the encrypted result, a reference to the encrypted table used as input data to the user computation, and the expression describing the computation performed by the user on the encrypted table. To implement computation verifiability, a function exists which returns all log entries corresponding to a given user-table pair; parity checking is used on each log entry to detect potential user fraud (see section 2). The parity checking function can be invoked each time the user asks for decryption of a result or it can be invoked after a certain number of decryptions have been logged.

4.2 Structure of StatClient

The structure of StatClient is depicted in figure 3. StatClient consists of a CalcServer, a TableServer and a reference to a StatServerAccess.

CalcServer is used to perform to operations on an encrypted table. It maintains a stack on which operations are performed. Operands are pushed onto the stack; arithmetical and logical operations (add, multiply, compare) take as operands the top two elements popped from the stack; the result is pushed onto the stack. Note that operations are carried out on encrypted data; thanks to polymorphism, an operation such as multiply is performed in a way which depends on the PH used for encryption.

TableServer is analogous to the TableServer in the StatServer, but it only contains tables associated to the current user.

4.3 Structure of StatAdmin

StatAdmin is in charge of the administration of the users, the tables and the security sets. When StatAdmin invokes the Admin method in StatServer, then StatServer creates an instance of StatServerAdmin. It is through the interface of StatServerAdmin that StatAdmin does its job.

5 Conclusion and future research

Due to its very nature, the solution to the delegation problem must be a distributed one. We have proposed cryptographic techniques to enable an untrusted handler to deal with sensitive data without compromising statistical confidentiality. Also, the structure of a prototype based on the CORBA technology has been described.

In its current state, our implementation still suffers from a lack of definition of the security policy to be enforced. A precise set of rules should be worked out to decide when decryption of a result should be denied to the data handler. Such a set of rules should depend on:

- The security limitations inherent to the privacy homomorphisms used. For example, it may be dangerous to release too many ciphertext-cleartext pairs.
- The lowest probability of handler fraud detection that can be tolerated by the owner. Some expressions claimed by the handler may be nearly always even (see section 2) and should probably be rejected.

A second line of research is the creation of a set of tools to analyze the logs of the operations on encrypted data. This analysis would take place off-line and could help detect "unfaithful" handlers, who could be subsequently denied authorization to access sensitive data.

References

[1] J. Castilla, J. Domingo-Ferrer and R. X. Sánchez, *Dikē: Delegation of Information without Knowledge Exposure*, internal reports #1 (Nov. 1996), #2 (Feb. 1997), #3 (May 1997) and #4 (in preparation).

[2] S. Cook and J. Daniels, *Designing Object Systems*, London: Prentice-Hall International, 1994.

[3] J. Domingo-Ferrer, "Multi-application smart cards and encrypted data processing", *Future Generation Computer Systems*, vol. 13, pp. 65-74, Jun. 1997.

[4] J. Domingo-Ferrer and R. X. Sánchez, "An implementable scheme for secure delegation of computing and data", in *International Conference on Information and Communications Security - ICICS'97*, Lect. Notes in Comp. Sci., Berlin: Springer-Verlag (to appear).

[5] The LiDIA Group, *LiDIA Manual. A Library for Computational Number Theory.* Ver. 1.3, Feb. 1997. TH Darmstadt/Universität des Saarlandes. ftp://ftp.informatik.th-darmstadt.de/pub/TI/systems/LiDIA

[6] Object Management Group, *OMG Common Request Broker Architecture: Architecture and Specification (CORBA)*, Revision 2.0. OMG Document Number 96.03.04, March 1996.

[7] R. L. Rivest, L. Adleman and M. L. Dertouzos, "On data banks and privacy homomorphisms", in *Foundations of Secure Computation*, R. A. DeMillo *et al.*, Eds. New-York: Academic Press, 1978, pp. 169-179.

[8] R. L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, pp. 120-126, Feb. 1978.

[9] J. Siegel, *CORBA Fundamentals and Programming.* New York: Wiley, 1996.