An Implementable Scheme for Secure Delegation of Computing and Data

Josep Domingo-Ferrer[1] and Ricardo X. Sánchez del Castillo[2]

[1] Universitat Rovira i Virgili, Escola Tècnica Superior d'Enginyeria, Autovia de Salou s/n., E-43006 Tarragona, Catalonia, e-mail jdomingo@etse.urv.es
[2] Universitat de Barcelona, Servei d'Informàtica, Trav. Corts 131-159, E-08028 Barcelona, e-mail ricardo@dalila.ird.ub.es ⋆⋆⋆

Abstract. The need for delegating information arises when the data owner wants to have her 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. Uses of delegation range from public administration to smart cards. In this paper, correctness and security requirements as well as protocols are specified for delegation of computing and data. A cryptographic solution to the secure delegation problem is described which provides data confidentiality and computation verifiability. Finally, an implementation allowing secure delegation of information over the Internet is briefly discussed.

1 Introduction

In many scenarios, 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 implementable solution 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 A data owner (client) sends to a handler (server) a data set, some basic operations and an expression to be evaluated on the data. The handler subsequently returns to the owner the result of the evaluation of the expression on the set of data. The data owner is the client interested in the computation.

⋆⋆⋆ 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.

Data delegation A data owner (server) sends a data set to a handler (client), who thereafter performs some computation with those data. The handler is the client interested in the computation.

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 a few practical applications:

- A computing delegation problem happens whenever a (small) company wants to use external computing facilities to do some calculations on corporate confidential data. Think of 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.
- Data delegation problems appear when several lower-level organizations (municipalities, member states, etc.) cooperate with a higher-level organization (national agency, European Union, etc.) in data collection (census data, etc.). In return, the former organizations would like to analyze the whole collected data set, but only the latter organization is legally entitled to do so. Without secure data delegation, the higher-level organization will have to spend resources in (uninteresting) analyses requested by the cooperating organizations.
- In [4], availability of secure data delegation is relied on for increasing the multi-application capacity of smart cards. The basic idea is that if a very resource-demanding application is to be run on card-stored data then the card exports these data in encrypted form and the application is run on an external computing server.

Requirements and protocols for computing and data delegation are specified in section 2. Solving both problems relies on data confidentiality and computation verifiability. Encrypted data processing is sketched in section 3 as a way of preserving data confidentiality. A method achieving computation verifiability is described in section 4. Finally, section 5 briefly discusses the implementation of a prototype allowing secure delegation.

2 Requirements and Protocols for Delegation

The basic requirements of both computing and data delegation are *security* and *correctness*. In both types of delegation, security essentially translates to preservation of *data confidentiality*. This means that the handler should not know the computation input data.

In delegation of computing, correctness essentially translates to *verifiability of the handler computation* by the data owner. The handler may deviate from the claimed computation accidentally (overflow) or intentionally. In any verification procedure, there is a tradeoff between the confidence attained and the

resources spent. If total confidence is desired, the owner must entirely repeat all computation (which makes delegation of computing useless); on the other hand, if the owner does not worry about the correctness of the computation, no check is needed. Intermediate solutions will be proposed in this paper which provide a reasonable confidence at a reasonable cost.

In data delegation, there is no correctness requirement from the data owner's viewpoint, since computation is done by the handler for his own use. From the handler's viewpoint, overflow detection is a serious problem when computing on encrypted data. The owner's verification can help as follows: if the owner detects fraud, she tells the handler. If the handler committed no fraud, he knows that overflow has occurred.

To meet the above requirements, the following protocols are proposed which rely on encryption

Protocol 1 (Computing delegation) *1. Prior to sending data to the handler, the owner encrypts them using an encryption transformation which allows some operations to be performed directly by the handler on encrypted data (homomorphical encryption).*
2. The handler returns a result which is still encrypted and must be decrypted by the data owner to recover the clear result.

Protocol 2 (Data delegation) *1. Data sent to the handler are encrypted by the owner using a homomorphical encryption transformation. The owner also supplies the handler with the operations on encrypted data supported by the encryption transformation used.*
2. Upon completing his computation, the handler sends the (encrypted) result to the owner for decryption. The handler also provides the expression used to obtain such a result, in order to allow some verification.
3. The owner verifies *the handler's computation. If no fraud can be detected and the claimed expression does not lead to evident disclosure of input data (inference controls should be applied here, but this is beyond the scope of this paper, see [2]), then the owner returns the decrypted result to the handler. Fraud occurs when the result does not correspond to the claimed expression.*

In summary, to solve both the secure computing and data delegation problems, we need to solve two more basic problems: data confidentiality and computation verifiability. We next propose solutions to such problems.

3 Data Confidentiality

As mentioned above, in both delegation problems the handler must be able to compute without being revealed the input data. This means that 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 [8]).

Basically, such homomorphisms are encryption functions $E_k : T \longrightarrow T'$ allowing a set F' of operations on encrypted data without knowledge of the decryption function D_k. Knowledge of D_k allows the result of the corresponding set F of cleartext operations to be retrieved. As it has been shown above, the availability of secure PHs is central to the delegation of computation. RSA [9] is a well-known example of PH which allows multiplication and test for equality to be carried out on encrypted data. A summary of the state-of-the-art on PHs can be found in [4].

The choice of a particular PH depends on the type of the data to be dealt with. For qualitative (non-numerical) data, RSA resists chosen-plaintext attacks and provides a way for the handler to perform checks for equality on encrypted data. For numerical data, the PHs in [3] and [4] resist known-plaintext attacks and allow full arithmetic to be carried out by the handler on encrypted data.

4 Computation Verifiability

As noted above, absolute confidence can only be attained by the owner if she repeats the whole handler's claimed computation. But this makes delegation useless. Repetition of the claimed computation by the owner with low probability q, say 10%, only detects fraud or overflow with probability q. *Parity checking* is a more interesting alternative explained below, which can yield fraud or overflow detection probabilities as high as 50% (or more if intermediate results are verified).

Let the parity of an integer x be $Z(x) = x \bmod 2$. The parity of the addition or subtraction of two integers is obtained by XORing the parities of both integers. The parity of the product of two integers is obtained by ANDing the parities of both integers. If divisions are reduced to multiplications through use of fractions (as in the PHs of [3] and [4]), then any arithmetical expression that the handler claims to have computed on encrypted data can be mapped to a Boolean expression with XOR and AND gates yielding the parity of the claimed result.

Now, the following subprotocol can be embedded in protocols 1 and 2 to allow verification of the claimed expression:

Subprotocol 1 (Parity checking) *1. The handler supplies the Boolean parity formula associated to the claimed expression in SOP form (by a well-known theorem of switching theory, any Boolean expression can be rewritten in "sum of products" or SOP form,* i. e. *as the OR of several AND terms, see [5]).*

2. The data owner uses the supplied Boolean formula and her knowledge of clear input data to compute the claimed parity of the result (which is very fast). Then she decrypts the encrypted result computed by the handler and compares its actual parity against the claimed parity. If parities differ, fraud or overflow have been detected.

Notice that the handler never knows the parities of the data he is handling, because data are encrypted. The parity of the encrypted result is also unknown to the handler unless the computed expression is always even. Always-even expressions are those having the form $(d+d) \times e$, where d and e are subexpressions or input data (e is optional); as a side remark, notice that the handler cannot build always-odd expressions by just adding and multiplying encrypted data. If the claimed and the computed expressions differ but both are always even, they are indistinguishable using the parity checking subprotocol above. Therefore, always-even expressions should be forbidden by the data owner. This is not as restrictive as it may seem. If computing $(d + d) \times e$ is *really* needed, then compute instead $d \times e$. Once the clear outcome of $d \times e$ is known, just multiply by 2 in the clear. 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 for a Boolean formula in 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.

The following properties of the parity checking method are easy to prove:

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

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

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

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.*

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.*

Note. 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.

We conclude that the data owner can be assured of a nonzero detection probability but is *unable to compute $P(\mathit{detect})$ 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.

5 Dikē: A Prototype for Secure Delegation of Computation

A prototype christened Dikē (blind Greek goddess of justice and social order, and also Delegation of Information without Knowledge Exposure) is about to be completed which allows secure delegation [1]. As a conclusion, the design rationale of Dikē is briefly discussed in this section.

It was argued in section 1 that delegation problems can be viewed as client-server problems. For computing delegation, the client is the data owner. For data delegation, the client is the handler. In this context, it becomes clear that Dikē must be a distributed application.

On the other hand, when operating on encrypted data, the way a given operation is implemented depends on the particular privacy homomorphism used for encryption. Polymorphism provided by object-oriented technology is very useful here, because it allows coding handler applications that are independent of the privacy homomorphisms used by the data owner. The code for an operation may depend on the data type in a transparent way. For instance, "multiply $E_k(a)$ and $E_k(b)$" will call a different code if $E_k()$ is RSA encryption or encryption using the PH of [4] (the code to be used is provided by the data owner, who chooses either privacy homomorphism).

So, we conclude that Dikē should have a distributed object-oriented architecture. Ideally, Dikē's architecture should also be language-independent and even platform-independent. CORBA [7][10] is a standard for distributed computing offering such independence and has been adopted for Dikē. More specifically:

- CORBA provides a universal notation for the software interfaces (Interface Definition Language or IDL), with standard maps for many languages.
- The CORBA abstraction simplifies the construction of distributed computing applications and this allows the developer to concentrate in the core of the problem.
- Reusability, interoperability and scalability are concepts central to CORBA, and are inherited by CORBA-based applications.

Even if the resulting application is platform and language-independent, a platform and a language were needed for development. Unix and C++ have been

chosen. CORBA is available for Unix and translators from IDL to C++ exist. Moreover, C++ has the advantage of being object-oriented and widely used.

Finally, an arithmetical library had to be chosen to implement encryption, decryption and operations on encrypted data. Our requirements were: speed, wide range of number-theoretic primitives, easy multiprecision handling and an object-oriented interface. LiDIA [6] was our choice. Although no specific cryptographic primitives are included in LiDIA, they are very easy to derive from the available primitives.

References

1. J. Castilla, J. Domingo-Ferrer and R. X. Sánchez, *Dikē: Delegation of Information without Knowledge Exposure*, internal reports #1 (Nov. 1996) and #2 (Feb. 1997).
2. D. E. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982 (chapter 6 on inference controls).
3. J. Domingo-Ferrer, "A new privacy homomorphism and applications", *Information Processing Letters*, vol. 60, no. 5, pp. 277-282, Dec. 1996.
4. J. Domingo-Ferrer, "Multi-application smart cards and encrypted data processing", *Future Generation Computer Systems*, vol. 13, no. 1, pp. 65-74, July 1997.
5. F. J. Hill and G. R. Peterson, *Introduction to Switching Theory and Logical Design*. New York: Wiley, 1981.
6. 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
7. Object Management Group, *OMG Common Request Broker Architecture: Architecture and Specification (CORBA)*, Revision 2.0. OMG Document Number 96.03.04, March 1996.
8. 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.
9. 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.
10. J. Siegel, *CORBA Fundamentals and Programming*. New York: Wiley, 1996.