2003

Using Online Services in Untrusted Environments: A Privacy Preserving Architecture

Claus Boyens
Humboldt-University, Berlin, boyens@wiwi.hu-berlin.de

Oliver Guenther
Humboldt-University, Berlin, guenther@wiwi.hu-berlin.de

Using Online Services in Untrusted Environments: A Privacy-Preserving Architecture

Claus Boyens[*]
Institute of Information Systems, Humboldt University Berlin
Spandauer Str. 1, 10178 Berlin, Germany
Phone: +49-30-2093-5742, Fax: +49-30-2093-5741
boyens@wiwi.hu-berlin.de

Oliver Günther
Institute of Information Systems, Humboldt University Berlin
Spandauer Str. 1, 10178 Berlin, Germany
Phone: +49-30-2093-5742, Fax: +49-30-2093-5741
guenther@wiwi.hu-berlin.de

Abstract

While online service providers are sometimes accused of forwarding identifying customer information as name and address to untrusted third parties, comparatively little attention is paid to the input data that customers provide explicitly to the service. If the input data is sensitive but the service provider is not completely trustworthy, this constitutes a serious privacy problem. From a privacy-defending point of view, the most desirable situation would be for the service not to require any kind of sensitive information at any time, while still yielding useful results for the customer. This paper presents a service architecture that allows for the use of a restricted number of services without requiring the transmission of unencrypted customer data to the service provider. The supported services include the execution of basic database and arithmetic operations that can be combined in numerous useful ways. The basic idea of this architecture is to transform sensitive data on the client side before transferring it to the service provider. The latter processes the transformed data without being able to draw any further conclusions from it. The pseudo-result obtained is returned to the customer who applies a special retransformation to obtain the actual result.

Keywords

Electronic Commerce, Privacy Protection, Service Architectures, Encryption

1 Introduction

When it comes to the protection of sensitive user data in online services, the danger most often described is the forwarding of identifying information as user name, address, e-mail or telephone number to an untrusted third party. This mainly happens for marketing purposes in related branches of a similar industry, e.g. when hotel chains acquire airline customer lists. Comparatively little attention is drawn to the information that is needed to perform the service, i.e. its input data. However in some particular areas as financial and human resource services, the data to be processed contains important details. One often observes a mismatch between the actual aim of the service provision and potential privacy violations. For example,

[*] This research was supported by the Deutsche Forschungsgemeinschaft, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316/3)

a company's wage accounting system may be designed to issue the monthly payments or to calculate aggregate figures, but it could easily give out the number of hours that a particular person was absent in a given period of time.

The problem is that the services mentioned above require (unencrypted) client data. If the data is sensitive and the service provider is not completely trustworthy, this may be an argument for not using the service at all. Note that service features as "secure encrypted data transfer" just relate to the communication channel that connects the machines of client and server. Still, data is stored in plaintext format in the service provider's database and is of course not encrypted for processing purposes. The threats to these assets are manifold:

- External attack of the provider's database.
- Malicious or incompetent staff on the provider's part (esp. database and system administrators).
- Bankruptcy of the provider with consequential insecurity about data property.

As most of the leading data centres have installed robust protection means against attacks from outsiders, more importance should be assigned to the two latter threats. As the CSI/FBI Computer and Crime Survey (Power 2001) shows, the financial damage caused by unauthorized insider access is an order of magnitude higher than that caused by external system penetration ($1.01M vs. $0.17M). Furthermore, the study shows that disgruntled employees are the most likely source of attacks, even more likely than independent hackers or competitors (81% vs. 77% / 44%, resp.). Database and system administrators with extensive rights are potentially able to acquire almost all the information in the customer database if they should so choose. The other important threat assumes an incident which is all too common among new economy firms: Customer data is sold once the service provider has run out of money. Who could guarantee customers that their most sensitive data does not end up at their direct competitor?

It seems reasonable, therefore, to attempt to protect sensitive client data in the described service environment. Whereas organizational and legal measures may alleviate some of the potential threats, the question for technical measures then arises. The strongest claim possible is not to give away *any* kind of sensitive information to the service provider at *any* time, while still allowing for the requested service to be carried out. This paper presents a service architecture that allows for the outsourcing of a restricted number of simple services without revealing plaintext data. It includes the execution of basic database and arithmetic operations without revealing *any* kind of information to the service provider. The principle of this architecture is to transform sensitive data on the client side before transferring it to the service provider. The provider can process the transformed data but not draw any further conclusions from it. The result of the service request, which still carries no meaning to the service provider, would then be returned to the client who re-transforms it into the actual term in plaintext.

The triple threat described above is emphasized by the current privacy policy of Amazon.com (2002), the world's biggest online retailer:

> "…we might sell or buy stores, subsidiaries, or business units. In such transactions, customer information generally is one of the transferred business assets... Also, in the unlikely event that Amazon.com, Inc., or substantially all of its assets are acquired, customer information will of course be one of the transferred assets…"

In spite of numerous protests since the introduction of this policy (Darrow 2000), the only effective measure that protects user data from being sold is not to use the service. Nevertheless, some technical countermeasures are currently being developed.

Private Information Retrieval (PIR) was first presented by Chor et al. (1995). It allows users to query a database server, revealing neither the query nor the result of the query to the server. This is facilitated via a secure hardware device, a *secure coprocessor* (Smith and Weingart 1999), which is the only place where the query is decrypted and processed. Asonov and Freytag (2002) adopted PIR for practical deployment.

Hacigumus et al. (2002) present a client-server approach that allows the elementary database operators to be executed over encrypted data. They fragment the data on the client side, where also some of the processing such as sorting data sets must be done.

Data Disclosure Protection deals with (mostly statistical) tables that are anonymized before their publication. Denning (1982) first showed that from aggregate data, conclusions can often be drawn for single values. Recently, Sweeney (2001) revealed potential privacy breaches in published health care tables and proposed more efficient anonymization techniques.

This paper takes advantage of theoretical foundations in cryptography and derives a new privacy-preserving service architecture. A specific algorithm is adopted to allow for more functionality in practically relevant cases. We map important database and arithmetic operations from the untransformed to the transformed data and we present sample services that can be carried out within the framework. We also evaluate the approach with regard to memory and performance requirements and we propose different implementation methods.

Section 2 presents the new architecture. Section 3 is dedicated to the corresponding data transformation algorithm. The extent of the feasible services is described in section 4. In section 5, we present some sample services and discuss performance and implementation issues. We conclude with a discussion of opportunities and limitations of the approach as well as with further research ideas.

2 A privacy-preserving approach to protect sensitive client data

In this section, we present a service architecture that allows for processing data with ultimate privacy, i.e. sensitive data is not only withheld with respect to non-trusted third parties, but also to the service provider itself. The service provider will not dispose of *any* unencrypted client data at *any* time. Contrary to the concept of Private Information Retrieval, no hardware equipment is involved. Our approach requires the service provider to work directly with encrypted data.

Following the approach of *public key infrastructure* first proposed by Rivest et al (1978b), the basic idea is to transform the sensitive data with the help of a secret key only known to the client. The service provider uses the corresponding public key in order to process the encrypted data. Without the private key, the server cannot see any sensitive information in plaintext (as is intended by the client). Without the public key, it cannot even compute the data.

From an infrastructural point of view, the architecture requires:

- The creation of a private key and its safe-keeping.
- The creation of a public key and distribution of it to the service provider.
- The equipment of client software with the transformation algorithm.
- The adoption of the server's business logic such that encrypted data can be processed.

How these requirements are dealt with in practice is discussed in section 5.3. The volume of infrastructure requirements implies that the approach is more suitable for Application Service Provider (ASP)-like solutions which allow at least for some customizing than for fine-grained, standardized web services.

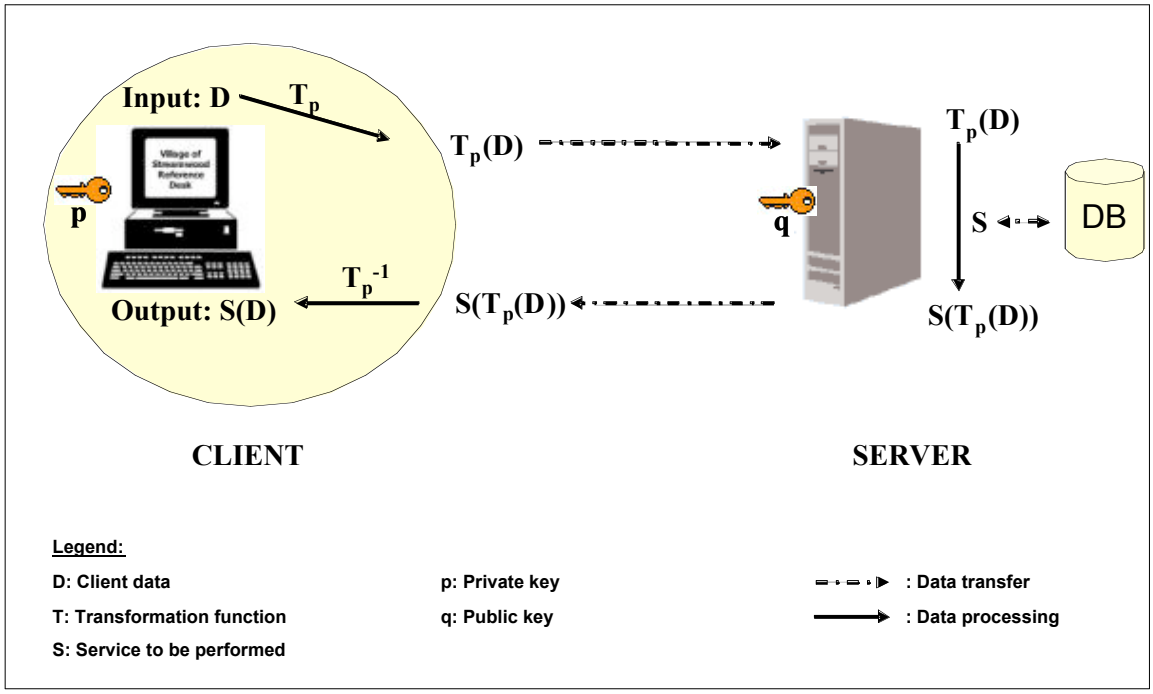

Figure 1: Sketch of the proposed service architecture

Figure 1 describes the general service procedure: The client C wants the service provider to perform some service S on the confidential data D she provides. After installing the key infrastructure, critical data is marked up as "sensitive" and the application running on the client machine encrypts it using the provided transformation scheme T and the private key p. The server, who only sees encrypted data, now uses the public key q to perform the requested service S. Once the server has performed its service, the encrypted pseudo-solution S(T(D)) is retransferred to the client who applies a re-transformation (usually T^{-1}) to obtain the desired result S(D).

The whole procedure is summed up in Figure 2.

1. Install key infrastructure
   i) C creates two large secret primes p (=: private key) and p´ (e.g. 128 bit numbers)
   ii) 2. C computes q= p*p´ (=: public key, e.g. 256 bit number)
   iii). C keeps the private key p and transfers the public key q to the Server
   iv). The Plaintext domain now is Zp, the Ciphertext domain is Zq
2. C encrypts the sensitive data with T_p
3. C transmits the transformed data $T_p(D)$ to the service provider
4. The service provider carries out the service S on the transformed data $T_p(D)$
5. C receives the encrypted result $S(T_p(D))$ from the server
6. C decrypts the result to S(D) with the inverse transformation function T^{-1} and eventually performs some post-processing

Figure 2: The proposed service procedure

An example:

Let us give a short example. Suppose C is a company that outsources its human resource (HR) management system to a service provider. C requests: "Give me the total number of absence hours in the treasury department for October 2002!" This would fit into the proposed terminology as follows:

C: The company that outsources its HR Management System

D: The transferred employee data that is stored in the encrypted database at the server.

S: "Calculate the average over the absence hours in October in the Finance Department!"

T: The transformation scheme that maps employee data to encrypted data

3 Encryption and decryption (How data is transformed)

3.1 Cryptographic background: Privacy homomorphisms

The "transformation schemes" referred to in the previous section actually are *encryption functions* in the cryptographic terminology. These functions map *plaintext*, the readable sensitive data, to *ciphertext*, its encoded counterpart. Cryptanalysts determine the security of an encryption scheme in terms of its resistance against five attacks of increasing scale. In order to simplify the framework, we will denominate the most unsafe encryption scheme *level-1-secure* and the safest one *level-5-secure*. For a detailed discussion of the actual scale from "ciphertext-only-resistance" to "chosen-text-resistance", see Stallings (1999).

The architecture presented in this paper is based on a particular class of encryption functions, so-called *privacy homomorphisms (PHs)*. Rivest et al. (1978a) introduce them as "encryption functions that permit encrypted data to be worked with without preliminary decryption of the operands". The sample PH they describe yields that the multiplicative product of two encrypted numbers is equal to the encryption of the corresponding cleartext product.

$$T(d_1)*T(d_2)= T(d_1*d_2)$$

Applying the inverse function gives

$$T^{-1}(T(d_1)*T(d_2))= T^{-1}(T(d_1*d_2))= d_1*d_2$$

To employ PHs in service environments, the basic idea would be to transfer the encrypted data $T(d_1)$ and $T(d_2)$ to the service provider instead of the cleartext pair d_1 and d_2. The provider then computes the pseudo-solution $T(d_1)*T(d_2)$ which is, due to T's homomorphic property, equal to $T(d_1*d_2)$. Applying the inverse function T^{-1} then decrypts the pseudo-solution and yields the desired result d_1*d_2. Figure 3 shows this basic idea:

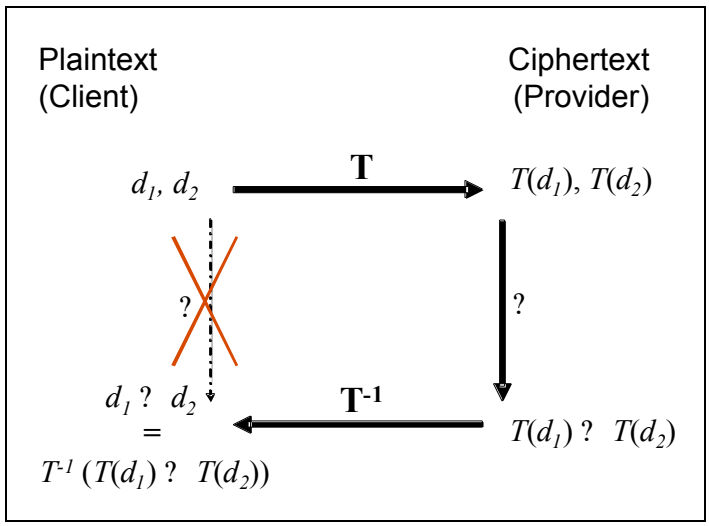

Figure 3: Scheme of a privacy homomorphism T

If one considers "multiplication" as a simple kind of service, T thus guarantees ultimate privacy because the customer may use the service while revealing neither the factors nor the result to the service provider.

Whereas this multiplicative PH is a very secure one (level-4), performing addition on encrypted data turns out to be a more complicated issue. Ahituv et al. (1987) show that an additive PH may reach at most level-2 security. Brickell and Yacobi (1988) are the first to present an R-additive PH that permits the addition of up to R numbers with level-1 security. Finally, Domingo-Ferrer and Herrera-Joancomartí (1998) present a PH allowing all field operations (addition, subtraction, multiplication and inverse multiplication) on an arbitrary number of ciphertexts. Though it is just level-1 secure, it would still force the potential attacker to acquire plaintext information from the client in order to be successful (Boyens and Günther 2002), which transfers at least some of the responsibility from the service provider to the customer. If a plaintext-ciphertext pair has been known to the attacker, it will be difficult for the customer to deny at least part of the responsibility for the break-in. We will now describe the PH in use with the proposed architecture.

3.2 The transformation scheme deployed in the proposed architecture

The PH we base our architecture on is adapted slightly from the scheme proposed by Domingo-Ferrer and Herrera-Joancomartí (1998). After installing the key infrastructure, the procedure depicted in Figure 4 is applied.

Input

p: The private key

q: The public key

a: A secret number ∈ Zp

d: Some kind of sensitive data AUS ∈ Zp

Output

Tp(d): The sensitive data encrypted with the private key p

Encryption procedure

Solve the modular equation a*x = d (mod p) for x ∈ Zp

Tp(d)= a*x (mod q)

Figure 4: The encryption procedure

Note that this scheme differs from the PH proposed by Domingo-Ferrer and Herrera-Joancomartí (1998) in the sense that *a* is *not* chosen arbitrarily but as a fixed and secret prime ∈ Z_p. As modular equations a*x= d (mod p) have unique solutions for a, x, d ∈ Z_p, unique plaintext identifiers have the same ciphertext correspondents. This would not have been the case if a had been chosen arbitrarily. This feature is important since, for instance, primary keys for a database can now be addressed by a unique ciphertext. The check for equality allows for picking single records out of the encrypted database, thus permitting the updating, deleting and retrieving of records that already exist in the database.

The decryption works in a similar manner. The difference consists of the fact that *A* can be chosen arbitrarily, as the transformation scheme guarantees the plaintext originally provided as the result (Figure 5).

Input

p: The private key
q: The public key
t: Some encrypted data ∈ Zq

Output

$T_p^{-1}(t)$: The plaintext corresponding to t

Decryption procedure

Pick A∈Zq arbitrarily
Solve the modular equation A*y = t (mod q) for y ∈ Zq
$T_p^{-1}(t)$= A*y (mod p)

Figure 5: The decryption procedure

3.3 Another example

As a simple example, we will compute the product of 3 and 5 with a small (5-bit) prime p.

Given

p= 17
p'= 31
a= 13, a∈ {2, 3, 4, …, 16}
q= 17*31= 527
d_1= 3 (∈Z_{17})
d_2= 5 (∈Z_{17})

Encrypt

→ Solve a*x= 13*x= 3 (mod 17) → x= 12; a*x= 156 (mod 527)= $T_{17}(3)$ ∈Z_{527}
→ Solve a*x= 13*x= 5 (mod 17) → x= 3; a*x= 39 (mod 527)= $T_{17}(5)$ ∈Z_{527}

Calculate product on encrypted data

$T_{17}(3) *_{(\text{mod } 527)} T_{17}(5)$ = 156*139= 6084 (mod 527)= 287

Decrypt

Pick A arbitrarily $\in Z_q$: A= 412

Solve A*Y= 287 (mod 527), A*Y= 25056

$T_{17}^{-1}(T_{17}(3) *_{(\text{mod } 527)} T_{17}(5))= 25056 \text{ (mod } 17)= 15$

4 „Enabled Services“: Which services can be performed?

4.1 Processing the encrypted data

Now that the basic service idea and the corresponding transformation scheme are introduced, we will discuss *which* actual services the server is able to carry out on the modified information he disposes of. Naturally, encrypted data cannot be processed with the same extent of operations as unencrypted data would allow to. As already indicated in section 2, we will distinguish between two different elementary service types.

The first elementary service type concerns basic database queries, such as retrievals and updates. We will analyze the basic relational operators concerning their suitability for handling encrypted records and give examples in section 4.2.

The second elementary service type consists of the basic arithmetic operations, addition and subtraction, multiplication and division. We will show to which extent and on which kind of plaintext data these operations can be applied in section 4.3.

Of course, real-life services would often include both types. A sample service of practical relevance is discussed in section 5.1.

4.2 Database operators

Here we introduce an example that will serve as a reference throughout the rest of this paper. The client is a company who wants to outsource part of its HR Management System. It transfers information about its employees and about the monthly wage accounts in the following two tables.

- `employee (employee_no, name, year_of_birth, department);`
- `monthly_account (employee_no, month, absence, overtime, payment);`

The `employee` table contains general information about the staff as employee number, name, year of birth and department. A typical data record would contain the following.

```
(432321, 'Schmidt', 1963, 'Finance')
```

The `monthly_account` table in contrast yields information about the monthly payment account as absent hours, overtime hours and payment.

```
(432321, 'AUG 2002', 12, 23, 3247)
```

We will now explain if and how standard Structured Query Language (SQL) queries can be mapped such that the encrypted database can be accessed.

Selection

```
(“SELECT name, year_of_birth FROM employee WHERE (department='Finance')”)
```

The value to retrieve is simply encrypted in the query

“…`WHERE department=` T_p`(ascii('Finance'))`”, where *ascii(‘Finance’)* would be the corresponding ASCII coding. The exact and complete value must be specified, as the

transformation scheme does not allow for "partial encryption". Therefore, working with wildcards ("…`WHERE (department LIKE 'F%')`") is not possible.

Projection

("**`SELECT name, year_of_birth`** `FROM employee WHERE (department='Finance')`")

Projection is possible without restrictions, as usually all the attribute names must be specified with their exact and complete names. Furthermore, it is up to the client to decide whether just to encrypt the values or to encrypt the attribute names, too. In the latter case, the query would start with:

("SELECT T_p(name), T_p(year_of_birth)…")

Join

("SELECT payment FROM employee e, monthly_account m WHERE **(e.employee_no = m.employee_no)**")

The Join command for data from different tables works well as long as the matching is done with complete attributes (no wildcards). The privacy homomorphism guarantees that identical unencrypted values will have the same ciphertext correspondent. For example, T_p(employee_no) will be the same in table `employee` as in the table `monthly_account`.

Sorting

("SELECT name, year_of_birth FROM employee **SORT BY year_of_birth)**")

The ability to sort presumes the existence of a total order over the encrypted data. However, Rivest et al. (1978a) show that PHs that preserve total order in spite of the transformation cannot be even level-1-secure. Therefore, "SORT BY" cannot be conducted *at all* over encrypted data. An approach concerning how to facilitate this with some involvement of the client was recently proposed by Hacigumus et al. (2002).

In order to *modify* the encrypted database, additional operators are necessary for record insertion, deletion or updating. However, they all depend on the discussed query operators. Hence e.g. deletion is possible for specifically selected values, but not for wildcard values. As a result, all records whose "name" attribute is equal to "Miller" could be deleted, but not those with "name" attributes starting with "M%", as discussed for the "Selection" operator.

Table 1 sums up these results:

OPERATOR	FEASIBLE ON T_P(D)?	REMARKS
Selection	Partially	No wildcard selection possible
Projection	Yes	Attribute name not necessarily encrypted
Join	Partially	Only over exactly matching data
Sorting	No	Impossible on secure data

Table 1: Database query operators on transformed data

4.3 Arithmetic operations

All arithmetic operations discussed are principally *modular* operations. Yet on the plaintext domain Z_p, the very large prime p allows for the calculation of large sums and products without creating remainder terms through division by p. Hence addition, subtraction and

multiplication can normally be used as if the algebraic space was the regular algebraic ring $(Z, +, *)$. Furthermore, as $(Z_p, +_{\text{mod } p}, *_{\text{mod } p})$ is equivalent to an algebraic field, it allows for the computation of multiplicative inverses. All these properties are transferred to the algebraic space $(Z_q, +_{\text{mod } q}, *_{\text{mod } q})$ after applying the transformation scheme presented in section 3.2. The basic difference between $(Z_p, +_{\text{mod } p}, *_{\text{mod } p})$ and $(Z_q, +_{\text{mod } q}, *_{\text{mod } q})$ lies in the fact that *every* unencrypted datum is converted into a cipher of almost the same bit length as q, i.e. up to 256 bits. That means that e.g. the addition of salaries, say of 3275$ and 4023$ turns from the addition of 12-bit-integers to the addition of its 256-bit-long encrypted correspondents.

In the following, we will discuss the four basic arithmetic field operations. Afterwards, we will indicate for which aggregate operations the algorithm fits best.

Addition $d_1 + d_2 := T_p^{-1}(T_p(d_1) +_{\textbf{(mod q)}} T_p(d_2))$

The regular (non-modular) addition of the unencrypted data is mapped to the modular addition of the encrypted numbers. It works for all $d_1, d_2 \in Z_p$, as long as $[d_1+d_2 < p]$, which is not a strong condition because p is large.

Subtraction $d_1 - d_2 := T_p^{-1}(T_p(d_1) -_{\textbf{(mod q)}} T_p(d_2))$

As Z_p does not contain negative integers, this only works as long as $d_1 > d_2$. From $[(d_1-d_2) > 0]$ and $[(d_1-d_2) < d_1 < p]$ then follows $[(d_1-d_2) \in Z_p]$

Multiplication $d_1 * d_2 := T_p^{-1}(T_p(d_1) *_{\textbf{(mod q)}} T_p(d_2))$

This works as regular (non-modular) multiplication as long as $[d_1*d_2 < p]$. This can actually turn out to be a strong condition if the number of factors is very high.

Inverse Multiplication $d_1 * d_2^{-1} := T_p^{-1}(T_p(d_1) *_{\textbf{(mod q)}} T_p(d_2)^{-1})$

This only works as the common "division" as long as d_2 in fact divides d_1. If division leads to a remainder, one may still compute the multiplicative inverse of $T_p(d_2)$, but the decrypted product does not correspond to the a readable figure (as d_1 *DIV* d_2, the integer division, would). It should therefore only be used as the regular division when the property *d_2 divides d_1* can be ensured beforehand.

Table 2 sums up these findings.

OPERATION	FEASIBLE ON $T_P(D)$?	CONDITIONS
Addition	Yes	$d_1+d_2 < p$
Subtraction	Partially	$d_1-d_2 > 0$
Multiplication	Yes	$d_1*d_2 < p$
Division	Partially	$d_2 \mid d_1$

Table 2: Arithmetic operators on encrypted data

5 Practical applications

5.1 Sample services from the HR area

In this section, we will discuss a few sample services based on the encrypted `employee` and `monthly_account` tables presented in the previous paragraph. We think that HR data is particularly appropriate for this purpose, for two reasons. First, sensitive data can be found in various forms such as regular wages and bonus payments, absent and overtime hours, and sometimes even church affiliation. Second, HR Management tools are often subject to outsourcing and therefore represent a suitable application field for the proposed architecture.

S_1: Mean monthly absent hours in specific departments

Formally, this figure is calculated as the average μ_i over the absent hours of the employees *e* in department *dep1*

$\mu_{dep1} = (\_{(e.department= dep1)}\ e.absence) / |\{e \mid e.department= dep1\}|$

In order to calculate the mean absent hours for the 'Finance' department in August, the following actions are required on the provider's part.

1) Retrieve the `absence` attribute of all employees in the finance department.

   ```
   SELECT absence AS department_absence FROM employee e, monthly_account m
   WHERE (e.employee_no = m.employee_no) AND (e.department = Tp('Finance'))
   ```

   Note that this query includes a join over encrypted data, namely the employee number in both tables.

2) Calculate the sum over `department_absence`.

   $sum_{'Finance'} = \_{(e.department= Tp('Finance'))}$ `e.absence`

3) Return the encrypted $sum_{'Finance'}$ and the plain $record\_count_{'Finance'} = |\{e \mid e.department= 'Finance'\}|$ to the client.

4) The client decrypts the sum and divides it by the count to obtain the result μ_{dep1} finally.

   $\mu_{'Finance'} = T_p^{-1}(sum_{'Finance'}) / record\_count_{'Finance'}$

Note that lacking the possibility of dividing the two numbers leads to at least some involvement of the client. A good example for a service that does not need any kind of client intervention is the multiplication of matrices, as only multiplication and addition is required.

S_2: Standard deviation of payments among departments

This metric measures the income disparities among different departments. We will use a service similar to S_1 to calculate μ_i^*, the mean incomes per department.

$\_{all} = ((\_{(departments\ i)} |\mu_{all} - \mu_i^*|^2) / |\{ i \mid dep_i \text{ is department } \}|)$ –

with $\mu_{all} = (\_{(departments\ i)}\ \mu_i^*) / |\{ i \mid dep_i \text{ is department } \}|$

1) Compute the mean payments μ_i^* for all departments using a similar service to S_1.
2) Compute the average μ_{all} over all μ_i^*'s using S_1 again.
3) Compute the sum of the squared differences: $squared\_dev := \_{(departments\ i)} |\mu_{all} - \mu_i^*|^2$.
4) Return squared_dev and the number of departments department_count to the client.

5) The client decrypts squared deviation sum, divides it by the department count and draws the square root.

Again, some client involvement is required. However the major part of the calculation is done by the provider, which especially pays off if the underlying databases are large.

5.2 Performance and memory requirements of the architecture

In order to evaluate the proposed architecture, we created the employee and monthly_account tables with n=1000 data records. We first built them with unencrypted test data, then encrypting them using the proposed algorithm and a 32 bit, a 64 bit and a 128 bit key. As we focus on the protection of sensitive data, especially the transformation of absent hours, overtime and payment is relevant. The resulting tables have the following shape.

```
employee (Tp(employee_no), name, year_of_birth, department);
monthly_account (Tp(employee_no), month, Tp(absence),
     Tp(overtime), Tp(payment));
```

The time and memory requirements for the table creation are shown in tables Table 3 and Table 4.

	No key	32 bit	64 bit	128 bit
Creation Time (sec)	9.111	11.818	13.199	19.35
Surcharge for encryption	0%	30%	45%	112%
Table size (KB)	84	120	208	232
Surcharge for encryption	0%	43%	148%	176%

Table 3: Creation times and disk space for the unencrypted and the encrypted *employee* table

	No key	32 bit	64 bit	128 bit
Creation Time (sec)	9.405	11.951	28.236	51.139
Surcharge for encryption	0%	27%	200%	444%
Table size (KB)	80	208	348	456
Surcharge for encryption	0%	160%	335%	470%

Table 4: Creation times and disk space for the unencrypted and the encrypted *monthly_account* table

Furthermore, we compared the time required to perform the service S_1 within the same framework (Table 5).

	No key	32 bit	64 bit	128 bit
S_1: Average absence per dept. (ms)	88.2	92.684	95.404	97.762
Surcharge for encryption	0%	5%	8%	11%

Table 5: Service performance duration on unencrypted data and on encrypted data

Although only displaying a rough trend, these figures suggest that service performance suffers only slightly. On the other hand, the time necessary to create the tables and the space required to store the encrypted data are not insignificant.

5.3 Implementation issues

The implementation at hand is based on a JAVA applet that performs encryption and decryption as well as the post-processing on the client side. The applet would be loaded by the client every time the service is requested.

A more efficient approach would require the service provider to deliver a certified *browser plug-in,* which contains the transformation scheme and needs to be installed and parameterized by the client. The latter includes creation of the secret key. Sensitive data to be transmitted would then be marked with a specific HTML tag that forces the plug-in to encrypt the information before sending it.

Enterprise solutions could eventually take advantage of a proxy server through which every IP packet needs to pass. The proxy could check every packet for marked-up sensitive data and, if applicable, would transform the tag's content (Figure 6). This would yield the advantage that the secret key is only kept at the proxy and not on every client's machine.

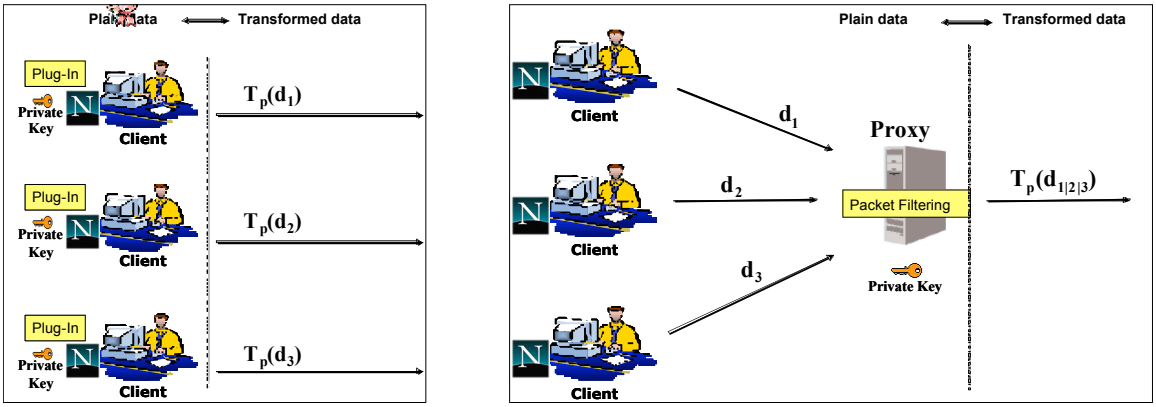

Figure 6: Implementation via plug-ins (left) and via proxy server (right)

Both approaches assume the existence of locally installed browsers. In the future, this may not always be necessary, as new techniques like the *remote GUI* only require the presentation layer to be processed at the customer site. With the data management completely shifted to the central facility, transforming sensitive information must then take place at the (untrusted) server location.

5.4 Opportunities and limits of the proposed approach

The proposed architecture should not be considered as a "one-size-fits-all" solution that works for every kind of network services. It focuses on applications that require some basic database and arithmetic operations on sensitive data. It is especially valuable for service bundles whose main value lies in their variety and their completeness in many smaller, granular services.

Different attributes require different encryption measures. The proposed algorithm is best suited for numbers that will later be processed with arithmetic operators. Primary key values in contrast often just serve as identification means, and are not subject to later processing. Hence it would be useful to encrypt these values with a very secure cryptographic algorithm as *RSA* (Rivest et al. 1978b). As the private key p has been picked already, it can be used to encode the key values with this alternate algorithm.

Regarding this, the proposed software solution is not suited for complex calculation problems, but for the aggregation of many single, rather simple services. A good example is the calculation of aggregate HR figures discussed previously.

6 Discussion and further research

We presented an architecture that allows a service provider to conduct a restricted number of services on encrypted client data. The fact that no unencrypted information is ever passed to the provider implies ultimate privacy preservation in the case that the server is not assumed to be trustworthy. The enabled services include basic arithmetic and database operations. A good application area is human resource management, as the number of concerned data records is often high and useful metrics can already be obtained by simply adding, multiplying and dividing over a specific set of values.

The limited extent of the enabled services of course restricts the architecture's fit into arbitrary systems. However, it is still useful to outsource some computations, especially when basic arithmetical operations are involved (e.g. for matrix multiplication). The proposed approach could then be part of a hybrid service architecture.

As the service infrastructure requires at least some personalization in the form of secret key generation and transformation scheme integration, the proposed approach would more likely be useful in customizable ASP services than in highly standardized, fine-grained web services. Lack of customer trust is still one of the major impediments in the ASP market and thus promotes the rise of privacy-preserving service models such as this one.

Future research includes the integration of different encryption schemes for different attribute types. Integer values, binary values and primary keys cannot be encrypted usefully with just one transformation scheme. Moreover, it is crucial for the acceptance of such an architecture that it can be easily installed and maintained. Therefore, a higher level of integration is necessary when it comes to client functionality, both for Plug-In and for proxy server solutions.

Acknowledgements

The authors would like to thank Sarah Aerni and Matthias Fischmann for helpful comments on earlier versions of this paper.

References

Ahituv, N., Lapid, Y., Neumann, S. (1987). Processing encrypted data. In *Communications of the ACM* 20:777-780.

Amazon (2002). *Amazon.com Privacy Notice* [http://www.amazon.com/exec/obidos/tg/browse/-/468496/103-4360611-7850263]. Viewed 2002/11/07.

Asonov, D., Freytag, J. C. (2002). Almost Optimal Private Information Retrieval. In *Proceedings of the 2nd Workshop on Privacy Enhancing Technologies (PET2002)*, San Francisco.

Boyens, C., Günther, O. (2002). Trust is not enough: Privacy and Security in ASP and Web Service Environments. In *Proceedings of Advances in Database and Information Systems (ADBIS 2002)*, Bratislava.

Brickell, E., Yacobi, Y. (1988). On privacy homomorphisms. In *Eurocrypt'87*, Springer, Berlin.

Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M. (1995). Private Information Retrieval. In *Proceedings of the 36th Annual IEEE Conference on Foundations of Computer Science*, pp. 41-50, IEEE, New York.

Darrow, B. (2000). Amazon.com Move Worries Privacy Pros. In *Tech Web News* September 4, 2000. [http://www.techweb.com/wire/story/TWB20000903S0001]. Viewed 2002/11/12.

Denning, D. (1982). *Cryptography and Data Security*, Addison-Wesley.

Domingo-Ferrer, J., Herrera-Joanconmartí (1998). A privacy homomorphism allowing field operations on encrypted data. In *Jornades de Matemàtica Discreta i Algorísmica*, Barcelona.

Hacigumus, H., Mehrotra, S., Iyer, B., Li, C. (2002). Executing SQL over Encrypted Data in the Database Service Provider Model. In *Proceedings of the ACM SIGMOD Conference on the Management of Data*, 2002.

Power, R. (2001). Computer Security Issues and Trends, *2001 CSI/FBI Computer Crime and Security Survey*. Vol. VII No. 1. Computer Security Institute, Spring 2001.

Rivest, R., Adleman, L., Dertouzos, M. L. (1978a). On Data Banks and Privacy Homomorphisms. In *Foundations of Secure Computations*, Academic Press, New York.

Rivest, R., Shamir, A., Adleman, L. (1978b). A Method for Obtaining Digital Signatures and Public-key Cryptosystems. In *Communications of the ACM* 21,2.

Smith, S. W., Weingart, S. H. (1999). Building a High-Performance, Programmable Secure Coprocessor. In *Computer Networks*, Special Issue on Computer Network Security, 31:831-860.

Stallings, W. (1999). *Cryptography and Network Security: Principles and Practice*. Prentice-Hall.

Sweeney, L. (2001). *Computational Disclosure Control.* A Primer on Data Privacy Protection. Ph.D. Thesis, MIT.