Designs, Codes and Cryptography, 8, 263–271 (1996)

Achieving Rights Untransferability with Client-Independent Servers

JOSEP DOMINGO-FERRER jdomingo@etse.urv.es
Statistics and OR Group, Departament d'Enginyeria Química, Universitat Rovira i Virgili, Autovia de Salou, s/n., 43006 Tarragona

Communicated by: A. Beutelspacher

Received May, 27, 1992; Accepted April 4, 1995

Abstract. This article presents a scheme for enforcing access rights untransferability in a client-server scenario. Assumptions include a central authority and servers which are trusted and hold no access information about clients. For a client sharing none of her rights, usurpation of a right seems as hard as the discrete logarithm. Also, rights sharing between clients does not compromise their non-shared rights when a sound public-key cryptosystem is used. Transferring rights between clients without the authority's contribution cannot be done if a sound public-key cryptosystem is used. However, only control on *partial* rights transfers is addressed in this paper, which does not deal with *total* identity transfer or alienation.

Keywords: Rights Untransferability, Client-Server Architecture, Distributed Access Control, Cryptography, Computer Security

1. Introduction

For a distributed computer system to work properly, the entities that require identification are hosts, users and processes—see [7][5] for a more accurate description. When one of those entities requests a service from another entity, the term *client* is used to denote the first entity, and the term *server* is used for the second entity. Servers are trusted and will provide the requested service only after checking that the would-be client possesses a *right* to obtain that service from them.

Consider a typical distributed scenario consisting of a large network with a central authority, a set of trusted servers giving access to certain resources, and a community of clients. Clients are granted rights by the authority, and servers need only a certified list of available access rights in order to perform access control. *Servers store no access information about clients*, neither access lists nor capabilities, and thus the authority is able to perform client registration and rights granting independently of servers. In addition, granting and revoking rights are public procedures.

Keeping no access information in the servers is not common in conventional access control schemes such as [4] or [3], so that server control on the rights transfers between clients—like the one implemented with *copy flags* in the [4] version of the access matrix model—is not feasible. Now the question is: *How to achieve rights untransferability in a scenario where servers are client-independent?*

The mechanism presented in this article fulfills all the requirements of the distributed

scenario above, with the only additional constraint that servers be able to securely hold a private key. The degree of security is such that

- Usurping a right seems for a client who shares no rights as hard as solving a discrete logarithm.
- If a sound public-key cryptosystem is used, rights sharing between clients does not compromise their non-shared rights.
- If a sound public-key cryptosystem is used, for a client to transfer some of her rights to another client, the transfer must be performed by the authority.

This paper generalizes, develops and illustrates the mechanism sketched in [2], which relied on RSA.

Remark 1. Our primary goal is to prevent a client c from unauthorizedly transferring *some* of her rights to another client c'. Notice that it is always possible for c to completely reveal her identity to c', so that c' could use all rights belonging to c, by impersonating her. The problem of *total* identity transfer or alienation will not be dealt with here; this possibility always exists because in our context the *identity* of c consists of a secret number a owned by c and only shared with the authority.

The initial assumptions for the scheme are listed in section 2. Section 3 contains the scheme itself along with a theorem on its correctness and soundness. Section 4 assesses the risks of rights sharing. Untransferability is dealt with in section 5, where an algorithm to perform rights transfers with the authority's contribution is given as well. Finally, section 6 summarizes the features of the system and gives some practical examples where it can be used.

2. Initial Setting

DEFINITION 1 *A server is said to be client-independent if it does not store protected access information about its potential clients (neither access lists nor capabilities).*

As it was pointed out above, client independence allows the authority to register clients, as well as granting and revoking rights to them without having to communicate secretly with every server.

Consider two public numbers p and α, with p a large prime and α a generator of $\mathcal{Z}/(p)^*$. Take $p \gg (mn)^2$, where m is the number of clients and n is the number of rights.

Consider any sound public key cryptosystem (for example RSA [6] with modulus $N > p$). Take a key pair (e, d) such that e is *public* and d is *only known to the servers*.

DEFINITION 2 *Let $E()$ denote encryption under the public key e and let $D()$ denote decryption under the private key d (for RSA, $E(y) = y^e \bmod N$ and $D(y) = y^d \bmod N$).*

Remark 2. The reason for assuming trusted servers is now clear: they must be able to hold a private key, and they are not supposed to use it for collusion with clients. Since clients are untrusted, servers can act as clients but not conversely.

3. The Identification Scheme

Let $Auth$ be the central network authority. In the presence of several clients $c_0, \ldots, c_{m-1}$ a way must be found to be able to grant the same right to more than one client, while keeping a single numerical expression y for it (the rights are also client-independent). Algorithm 1 gives a solution to this problem. When a client c is to be granted her initial rights $y_0, \ldots, y_{n-1}$ the authority does

ALGORITHM 1

1. *Assuming that the first t rights among the $y_0, \ldots, y_{n-1}$ were granted to someone else in the past, choose $n - t$ random integers x_i, $t \leq i \leq n-1$ over $\mathcal{Z}/(p-1)$.*

2. *Pick a random number a over $\mathcal{Z}/(p-1)$, such that a is prime to $p-1$.*

3. *Generate n random integers r_i over $\mathcal{Z}/(p-1)$, for $0 \leq i \leq n-1$.*

4. *Find n nonzero numbers z_i over $\mathcal{Z}/(p-1)$, for $0 \leq i \leq n-1$, such that*

$$\begin{aligned} x_0 + r_0 &= az_0 \bmod (p-1) \\ &\vdots \\ x_{n-1} + r_{n-1} &= az_{n-1} \bmod (p-1) \end{aligned} \tag{1}$$

   where, for $0 \leq i \leq t-1$, x_i is such that $y_i = \alpha^{x_i} \bmod (p-1)$ (logarithms of granted rights are stored by the authority). To compute z_i solve the i-th equation for z_i using that a can be inverted over $\mathcal{Z}/(p-1)$.

5. *Compute $y_i := \alpha^{x_i} \bmod p$, $t \leq i \leq n-1$ and append these numbers together with their meaning—what the right y_i gives access to—to the certified public rights list available to both servers and clients.*

6. *Give the numbers z_i, $E(r_i)$, $0 \leq i \leq n-1$ to c in a public way.*

7. *Give the number a to c in a confidential way.*

Remark 3. It must be noticed that if the same right y_i is granted to two clients c and c', then the authority will give them different z_i, $E(r_i)$ and z_i', $E(r_i')$ in two different instances of the above algorithm, because each of those pairs depends on the client's identity (a or a').

It is possible to *publicly* give one more right $y_n = \alpha^{x_n} \bmod p$ to a client c having rights $y_i = \alpha^{x_i} \bmod p$, $0 \leq i \leq n-1$ and a secret number a. This is straightforward since,

according to the previous algorithm, it is possible for the authority to pick a random r_n over $\mathcal{Z}/(p-1)$ and compute an integer $z_n \in \mathcal{Z}/(p-1)$, such that $x_n + r_n = az_n \bmod (p-1)$. After this, the resulting z_n, $E(r_n)$ are given *in a public way* to the client and the procedure is finished.

A way to perform *rights revocation* is for the authority $Auth$ to publish a new certified rights list; then, *for each client c*, $Auth$ publishes the new numbers z_i, $E(r_i)$ corresponding to the rights kept by the client.

Bearing the above in mind, the following result holds for each client c

THEOREM 1 *If the authority Auth has completed algorithm 1 for a client c, then c is able to show possession of her rights $y_0, \ldots, y_{n-1}$ (or a subset of them) to any server in the network, that need not previously know about her. Proving knowledge of a single logarithm by the client is enough, no matter the value of n. Stealing a right seems to be for a client who shares no rights at least as hard as solving a discrete logarithm.*

Proof.

Correctness To get access to the service represented by right y_i a client c and a server go through the following protocol

PROTOCOL 1 (ACCESS TO A SERVICE)

1. *Client c supplies the server with integers $A(\neq 1)$, z_i and $E_i(\neq 0)$ satisfying the following equation*

$$y_i \alpha^{D(E_i)} = A^{z_i} \bmod p \tag{2}$$

2. *Client c proves her knowledge of $\log_\alpha A$ over $\mathcal{Z}/(p-1)$—this can be done in zero-knowledge using protocol 1 or 2 of [1]. Notice that the identity a has been given to c in the last step of algorithm 1, and it is straightforward from equations 1 that $A := \alpha^a \bmod N$ satisfies equation 2 when $E_i = E(r_i)$ and the same z_i is used in the i-th equation 1 and above.*

3. *The server checks that z_i, E_i satisfy equation 2. If this check and the proof in step 2 have been successful, the server grants access to the service represented by y_i.*

In order to simultaneously get access to the rights represented by $y_0, \ldots, y_n$, the same protocol above can be followed, by considering n equations instead of the single equation 2 in step 1 and by having the server perform n checks in step 3. Step 2 remains the same.

Soundness Equation 2 is verifiable by the server since y_i is public and certified. Next, it will be shown that protocol 1 makes it very difficult for a client c to usurp a right y_i that she does not own. Usurpation is only possible if c is able to supply the server with a triple z_i, E_i, A satisfying equation 2. But if c can find such a triple, then c can compute

the discrete logarithm of y_i as

$$x_i = \log_\alpha y_i = D(E_i) + az_i \bmod (p-1) \tag{3}$$

Therefore, usurpation is at least as hard as computing the discrete logarithm of y_i, and this seems to be just as hard as the general discrete logarithm problem. Remark that the difficulty of deriving $D()$ is not relevant to this argument, which implicitly assumes that c knows $a = \log_\alpha A$ and $D()$.

The proof is now complete and the server has needed no particular previous information about client c. □

4. How Dangerous Is Rights Sharing?

PROPERTY 1 (SECURITY OF RIGHTS SHARING) *If a sound public-key cryptosystem is used, then it seems not feasible for a client c to derive the identity of another client c′—and thus the non-shared rights of c′—by using the fact that c and c′ share a right—or a group of rights.*

Justification. When clients c and c' share a right y_i, they are not likely to share a left-hand side of any of equations 1, since different random numbers r_i, r_i' have been added to the logarithm x_i. The probability of picking different random numbers in equations 1 over $\mathcal{Z}/(p-1)$ for all m clients and n rights is

$$\frac{(p-1)(p-2)\cdots(p-mn)}{(p-1)^{mn}}$$

which approaches unity if $p - 1 \gg (mn)^2$.

So, the only equality in terms of the exponents that can be established when c and c' share a right y_i results from equations 1 and is

$$D(E(r_i)) - D(E(r_i')) = az_i - a'z_i' \bmod (p-1) \tag{4}$$

c knows $E(r_i)$, $E(r_i')$, z_i, z_i' and a in equation 4. Now, if c can derive a' from the above equation, then $D(E(r_i)) - D(E(r_i'))$ must be known to her. This seems difficult without knowledge of $D()$—notice that $r_i - r_i' \neq 0$, according to the beginning of the justification.

When two clients c and c' share a right, no way is apparent for c to forge a right ownership other than deriving the identity of c'.

□

5. Untransferability of Rights

PROPERTY 2 (UNTRANSFERABILITY) *If a sound public-key cryptosystem is used and clients do not alienate their identities, then it is not feasible for a client c to transfer a right to another client c′ by using the fact that c owns that right.*

Justification. Thanks to the use of randomization and subsequent encryption of the random numbers, neither of the integers x_i, r_i on the left hand side of equations 1 is known to a client c having a right y_i. Possession of such a right only means that the authority has issued two public numbers z_i, E_i linking the identity a of c with the right y_i (as in equation 2). So, for a client c to transfer a right y_i to another client c', it is necessary to find a pair z_i', E_i', such that

$$x_i + D(E_i') = a' z_i' \bmod (p-1) \tag{5}$$

c can eliminate the unknown x_i from equation 5 and the analogous equation with her values E_i, a, z_i, thus obtaining

$$D(E_i') - D(E_i) = a' z_i' - a z_i \bmod (p-1) \tag{6}$$

Equation 6 is analogous to equation 4, but here c knows E_i, z_i, a. The problem of determining values E_i', z_i' that satisfy the above equation seems difficult without knowledge of $D()$ or a'.

□

Remark 4 (Alienation and untransferability). If c' revealed a' to c, then a trivial solution of equation 6 would be $E_i' := E_i$ and $z_i' := a z_i / a' \bmod (p-1)$. Alternatively, if c gave $a z_i \bmod (p-1)$ to c', then c' could determine a, because z_i is public. Alienation makes right transfer trivial but is beyond the scope of this paper, as stated in remark 1.

Remark 5 (Attempting to use an alias). In order to avoid alienating a', c' can reveal to c an alias identity a^* different from a'. Without knowledge of $D()$, it seems that the only solution of equation 6 that c can compute is $E_i' = E_i$ and $z_i' = a z_i / a^* \bmod (p-1)$. But in this case c' can determine a from z_i', since she knows a^* and z_i is public. Thus, this kind of transfer avoids alienation of c' but implies alienation of c.

If c wants to transfer y_i to c' without alienation, then the only way is to have the job done (and monitored) by the authority. For example

PROTOCOL 2 (AUTHORIZED TRANSFER)

1. *Client c shows possession of right y_i to the authority by following a procedure analogous to the one in the proof of theorem 1 (the logarithm being shown possession of is x_i). The procedure requires that c prove knowledge of her identity a, which allows the authority to authenticate the giving client.*
2. *Client c' shows possession of a' to the authority in zero knowledge. In this way, the receiving client is authenticated by Auth.*
3. *Depending on the current security policy, the authority Auth may give y_i to client c' using the procedure for granting new rights discussed in section 3 (the logarithm being granted is x_i).*

6. Summary and Applications

As it has been shown, the proposed scheme is very flexible, since client management can be done independently of servers and, thanks to the linear transformation 1, the secret piece held by the client is constant and does not depend on the rights she owns at a given moment. Actually, it suffices for the client c to prove her identity a in order to use any subset of her rights, because a is the only secret parameter she holds.

As for the storage required, assuming n rights $y_0, \ldots, y_{n-1}$ and m clients $c^0, \ldots, c^{m-1}$ we have

Authority

1. Secret storage for logarithms x_i, $0 \leq i \leq n-1$.
2. Secret storage for all client numbers $a^{(k)}$, $0 \leq k \leq m-1$.
3. Read-write access to α, p, N, e and the list of the y_i's and their meanings (public certified data).

Servers

1. Secret storage for the servers' private key d.
2. Read access to α, p, N and the list of the y_i's and their meanings (public certified data).

Client $c^{(k)}$

1. Secret storage for her number $a^{(k)}$ (if the client is a human user, a smart card protected ROM is a good place for $a^{(k)}$).
2. Normal storage for her numbers $z_i^{(k)}$, $E_i^{(k)}$, $0 \leq i \leq n^{(k)} - 1$.
3. Read access to α, p and the list of the y_i's and their meanings (public certified data).

If we say that two elements are mutually dependent when there is some secret information relating them, then we have shown that functional dependencies between the different element classes of the access control system are those in table 1. The only actual dependencies are between a community of clients and the authority that gave them their identity, and also between a set of rights and the authority that publishes and certifies them in a list. So we see that servers are also authority-independent, and thus we might think of extending the proposed scheme so that *several authorities* (each with its client community and rights list) *share the same set of servers*.

The following are scenarios where the scheme presented in this paper could be used.

Example 1. Consider an access control system for a company's headquarters. The authority is the company, represented by a control computer connected to a local area network, clients are the employees represented by their smart-cards and servers are special devices located by the doors in the various buildings and connected to the network; rights enable access to

Table 1. Functional dependencies.

Depends on	Authority	Client	Server	Right
Authority	—	Yes	No	Yes
Client	Yes	—	No	No
Server	No	No	—	No
Right	Yes	No	No	—

the various rooms at different times of the day. The certified rights list is a digitally signed public file maintained by the control computer and retrieved by the door servers whenever they detect an update. When an employee is engaged, she is given a smart card initialized with a secret number a (along with some initial rights, following algorithm 1). Updates of the numbers z_i, E_i corresponding to each employee's rights are regularly broadcast through the company's internal bulletins. For convenience, these numbers are also available in a public file stored in the control computer; this file can be loaded on the employee's smart card by inserting it into any computer connected to the network. When an employee wants to enter a room, she inserts her smart card into the door server, and protocol 1 starts between card and server. Actually, the employee herself could input the z_i, E_i numbers to the server, but it is more convenient to leave the handling of such large integers to the card. For an employee to transfer a right to another employee, both employees must—maybe remotely—resort to the control computer, and follow protocol 2; the control computer decides on the transfer according to the corporate security policy and keeps a record of granted transfers for auditing purposes.

In the example above, door servers can be cheap microcomputers consisting of a clock, door-opening circuitry, a small processor, a standard ROM to implement protocol 1, a protected ROM to securely hold a private key, a cache memory to hold the current rights list, and a network connection. Smart cards consist of a processor, a standard ROM to implement protocols 1 and 2, a protected ROM to hold the identity a, and a writable memory to store the current numbers z_i, E_i for each right y_i. Rights granting and employee registration can be performed by the authority independently of servers. Both granting and revoking rights are public procedures. Cards and servers must hold only *one* secret parameter during their lifetime (identity number for cards and a private key for servers).

Example 2. As an example with several authorities, consider a network of teller machines shared by several credit card issuing corporations. The card issuing corporations are the authorities, each one with their client community (the card users) and a shared set of servers (the teller machines); rights enable to perform different banking operations. Thanks to the simpleness and independence of its server functionality, our scheme is very suited to deal with such a complex scenario.

Acknowledgement

I would like to thank two anonymous referees for their suggestions, which helped to thoroughly revise and greatly improve the contents of this paper.

References

1. D. Chaum, J.-H. Evertse and J. Van de Graaf, An improved procotol for demonstrating possession of discrete logarithms and some generalizations: Proceedings of Eurocrypt'87, Springer-Verlag (1988) pp. 127–141.
2. J. Domingo-Ferrer, Untransferable rights in a client-independent server environment: Proceedings of Eurocrypt'93, Springer-Verlag (1994) pp. 260–266.
3. G. S. Graham and P. J. Denning, Protection: Principles and practices: Proceedings of the AFIPS Spring Joint Computer Conference (1972) pp. 417–429.
4. M. A. Harrison, W. L. Ruzzo and J. D. Ullman, Protection in operating systems, *Communications of the ACM*, Vol. 19 (1976) pp. 461–471.
5. J. Linn, Practical authentication for distributed computing: Proc. IEEE Symposium on Research in Security and Privacy, IEEE CS Press (1990) pp. 31–40.
6. R. L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, Vol. 21 (1978) pp. 120–126.
7. T. Y. C. Woo and S. S. Lam, Authentication for distributed systems, *IEEE Computer*, Vol. 25 (1992) pp. 39–52.