Untransferable Rights in a Client-Independent Server Environment

Josep Domingo-Ferrer

Informàtica de Recerca i Docència
Universitat de Barcelona
Travessera de les Corts, 131-159 (Pavelló Rosa)
08028 Barcelona

Abstract. A scheme for ensuring access rights untransferability in a client-server scenario with a central authority and where servers hold no access information about clients is presented in this paper; an extension to a multi-authority scenario is conceivable, since servers are also authority independent. Usurping a right with no information at all about other clients is for a client as hard as the discrete logarithm, and rights sharing between clients does not compromise their non-shared rights as long as RSA confidentiality holds. Transferring rights between clients without the authority's contribution cannot be done unless RSA confidentiality is broken; however, only control on *partial* rights transfers is addressed in this paper, which does not deal with *total* identity transfer or alienation.

1 Introduction

In a distributed computing system, the entities that require identification are hosts, users and processes —see [Woo 92][Linn 90] for a more detailed framework. When one of those entities requests a service from another entity, we will use the term *client* to denote the first entity, and the term *server* to denote the second entity. A server will provide the requested service only after checking that the would-be client possesses a *right* to obtain that service from him.

Consider a typical distributed scenario consisting of a large network with a central authority, a set of 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, rights granting and rights revocation independently of servers; in addition, the latter two are public operations. Finally, it is also thinkable that servers do not depend on the authority, *i. e.*, that they store no confidential information about the authority.

Keeping no access information in the servers is not common in conventional access control schemes such as [Harr 76] or [Grah 72], so that server control on the rights transfers between clients —like the one implemented with *copy flags* in the [Harr 76] version of the access matrix model— is not feasible. Now the

T. Helleseth (Ed.): Advances in Cryptology - EUROCRYPT '93, LNCS 765, pp. 260-266, 1994.

question is: *How to achieve rights untransferability in a scenario where servers are client-independent?*

Remark 1. Note that if servers are trusted and client-dependent, *i.e.* if they hold some kind of access matrix exclusively updated by an authority, then the authority can trivially enforce untransferability by just having each client request checked by the servers against the client's rights in the access matrix. So, no right can be successfully transferred without the authority's contribution.o

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 RSA key. The degree of security is such that

- Usurping a right with no information at all about other clients is for a client as hard as solving a discrete logarithm.
- As long as RSA confidentiality holds, rights sharing between clients does not compromise their non-shared rights.
- As long as RSA confidentiality holds, for a client to transfer some of her rights to another client, the transfer must be performed by the authority.

Remark 2. Our primary goal is to prevent a client c_k from unauthorizedly transferring *some* of her rights to another client c_l. Notice that it is always possible for c_k to completely reveal her identity to c_l, so that c_l could use all rights belonging to c_k, 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 c_k's *identity* consists of a secret number a_k owned by c_k and only shared with the authority.o

The initial assumptions for the scheme are listed in section 2. Section 3 contains the scheme itself along with a theorem on security when a client has no information at all about other clients. 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 contains a functional summary and an extension of the system for a scenario with several authorities.

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.

Let N be a public RSA modulus [Rive 78], *i. e.* $N = q_1 q_2$, where q_1 and q_2 are two secret large primes; choose N to be greater than p. Take an RSA key pair (e, d) with modulus N, so that e is *public* and d is *only known to the servers.*

Definition 2. Let $E_{e,N}()$ and $D_{d,N}()$ be the usual RSA encryption and decryption functions, such that $E_{e,N}(y) = y^e \bmod N$ and $D_{d,N}(y) = y^d \bmod N$.

3 The Identification Scheme

The mathematical structure used is a modification of the one described in [Domi 91]: thanks to the use of RSA, the scheme becomes much simpler, but also dependent on the difficulty of factoring and on the servers securely storing a private RSA key. The algorithm in [Domi 91] relies solely on the discrete logarithm problem, but does not address untransferability.

Let *Auth* be the central network authority. In the presence of several clients $c_0, \cdots, c_{m-1}$ 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. Let us make some definitions before giving the algorithm.

Define n_k as the number of rights to be granted to client c_k, for $k = 0$ to $m-1$. Also, assuming that rights are granted first to c_0, then to c_1 and so on, let t_k be the number of rights, among the n_k to be granted to c_k, that have already been granted to some client in $\{c_0, \cdots, c_{k-1}\}$. Now *Auth* runs

Algorithm 1 *For $k = 0$ to $m-1$*

1. *Assume that the t_k rights having been already granted to someone else are $y_{k_0}, \cdots, y_{k_{t_k-1}}$. Choose $n_k - t_k$ random integers $x_{k_i}, t_k \leq i \leq n_k - 1$ over $\mathcal{Z}/(p-1)$.*
2. *Pick a random a_k over $\mathcal{Z}/(p-1)$, such that a_k is prime to $p-1$.*
3. *Generate n_k random integers r_{k_i} over $\mathcal{Z}/(p-1)$, for $0 \leq i \leq n_k - 1$.*
4. *Find n_k nonzero numbers z_{k_i} over $\mathcal{Z}/(p-1)$, for $0 \leq i \leq n_k - 1$, such that*

$$x_{k_0} + r_{k_0} = a_k z_{k_0} \bmod (p-1) \qquad (1)$$

$$\vdots$$

$$x_{k_{n_k-1}} + r_{k_{n_k-1}} = a_k z_{k_{n_k-1}} \bmod (p-1)$$

   To compute z_{k_i} solve the i-th equation for z_{k_i} using that a_k can be inverted over $\mathcal{Z}/(p-1)$.
5. *Compute $y_{k_i} := \alpha^{x_{k_i}} \bmod p, t_k \leq i \leq n_k - 1$ and append them together with their meaning to the certified public rights list available to both servers and clients.*
6. *Give the numbers $z_{k_i}, E_{e,N}(r_{k_i})$, $0 \leq i \leq n_k - 1$ to c_k in a public way.*
7. *Give the number a_k to c_k in a confidential way.*

It is possible to *publicly* give a right $y_{k_{n_k}} = \alpha^{x_{k_{n_k}}} \bmod p$ to a client c_k having rights $y_{k_i} = \alpha^{x_{k_i}} \bmod p, 0 \leq i \leq n_k - 1$ and a secret number a_k. This is straightforward since, according to the previous algorithm, it is possible for the authority to pick a random $r_{k_{n_k}}$ over $\mathcal{Z}/(p-1)$ and compute an integer $z_{k_{n_k}} \in \mathcal{Z}/(p-1)$, such that $x_{k_{n_k}} + r_{k_{n_k}} = a_k z_{k_{n_k}} \bmod (p-1)$. After this, the resulting $z_{k_{n_k}}, E_{e,N}(r_{k_{n_k}})$ 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 *Auth* also publishes the new numbers $z_{k_i}, E_{e,N}(r_{k_i})$ corresponding to the rights y_{k_i} which are maintained for each client c_k.

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

Theorem 3. *If the authority Auth has completed algorithm 1 for a client c_k, then c_k is able to show possession of her rights $y_{k_0}, \cdots, y_{k_{n_k-1}}$ (or a subset of them) to any server in the network, that need not previously know about her. The proof can be zero-knowledge and, no matter the value of n_k, it consists of proving knowledge of one logarithm. Stealing a nongranted right is for a client with no information at all about other clients as hard as solving a discrete logarithm.*

Proof. Client c_k supplies the server with integers $A_k(\neq 1)$ and $z_{k_i}, E_{k_i}(\neq 0)$, for $i = 0, \cdots, n_k - 1$, satisfying the following set of equations

$$\begin{aligned} y_{k_0}\alpha^{D_{d,N}(E_{k_0})} &= A_k^{z_{k_0}} \bmod p \\ &\vdots \\ y_{k_{n_k-1}}\alpha^{D_{d,N}(E_{k_{n_k-1}})} &= A_k^{z_{k_{n_k-1}}} \bmod p \end{aligned} \tag{2}$$

Now if c_k is able to prove her knowledge of $\log_\alpha A_k$ over $\mathcal{Z}/(p-1)$, then the server can check that, if c_k knew his key, she *could* express the y_{k_i}'s as powers of α, *i. e.* that c_k *could* obtain the logarithms of the y_{k_i}'s for $i = 0$ to $n_k - 1$. Notice that c_k has been given a_k in the last step of algorithm 1, and it is straightforward from equations 1 that $A_k := \alpha^{a_k} \bmod N$ satisfies equations 2 when for all E_{k_i}'s it holds that $E_{k_i} = E_{e,N}(r_{k_i})$ and the same z_{k_i}'s are used in both systems. Now Protocol 1 or 2 of [Chau 88] can be used to show possession of the logarithm of A_k in zero knowledge.

As for security, equations 2 are verifiable by the server since the y_i's are public and certified for $0 \leq i \leq n_k - 1$. Assume that c_k does not own a particular y_{k_i} but has invented or obtained the corresponding random number r_{k_i} (see equations 1); now if c_k is able to compute by herself a number z_{k_i} satisfying the i-th equation 2, then c_k is able to solve the discrete logarithm problem of finding x_{k_i}. On the other hand, if c_k invents z_{k_i} and manages to compute then a matching r_{k_i}, then c_k is also able to solve the discrete logarithm problem of finding x_{k_i}.

The proof is now complete and the server has needed no particular previous information about client c_k. Also, the construction can be applied to a subset of the y_{k_i}'s if the client does not wish to prove all of them.**QED**

4 How Dangerous Is Rights Sharing?

Theorem 4 Security of Rights Sharing. *If RSA confidentiality is not broken, then it is not feasible for a client c_k to derive the identity of another client c_l —and thus c_l's non-shared rights— by using the fact that c_k and c_l share a right —or a group of rights—.*

Proof. When clients c_k and c_l share a right y_i, they are not likely to share a left-hand side of any of equations 1, since each logarithm x_i has been added a random number. The probability of randomly picking different r_{k_i} 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_k and c_l share a right y_i results from equations 1 and is

$$a_k z_{k_i} - D_{d,N}(E_{e,N}(r_{k_i})) = a_l z_{l_i} - D_{d,N}(E_{e,N}(r_{l_i})) \bmod (p-1) \qquad (3)$$

c_k knows $a_k, z_{k_i}, z_{l_i}, E_{e,N}(r_{k_i})$ and $E_{e,N}(r_{l_i})$ in equation 3. Now, if c_k can derive a_l from the above equation, then $D_{d,N}(E_{e,N}(r_{k_i})) - D_{d,N}(E_{e,N}(r_{l_i}))$ must be known to her. In general, this is only possible if c_k can get r_{k_i}, r_{l_i} from decryption under $D_{d,N}$ —notice that $r_{k_i} - r_{l_i} \neq 0$, according to the beginning of the proof.**QED**

5 Untransferability of Rights

Theorem 5 Untransferability. *If RSA confidentiality is not broken, then it is not feasible for a client c_k to transfer a right to another client c_l without the authority's contribution.*

Proof. Thanks to the use of randomization and subsequent encryption of the random numbers, neither of the integers on the left hand side of equations 1 is known to the client. For a client c_k to transfer a right y_i to another client c_l, it is necessary to find a pair z_{l_i}, E_{l_i}, such that

$$x_i + D_{d,N}(E_{l_i}) = a_l z_{l_i} \bmod (p-1) \qquad (4)$$

But even if c_k knows a_l (collusion with c_l), c_k ignores x_i, because her own E_{k_i} is an encrypted random number. On the other hand, in order for a server to believe that c_l possesses y_i, α raised to the second term on the left-hand side of equation 4 times y_i must coincide with α raised to the right-hand side over $\mathcal{Z}/(p)$. So inventing a right-hand side of equation 4 and an E_{l_i} that decrypts into a coherent left-hand side second term is not feasible due to the ignorance of x_i by the clients.**QED**

If c_k wants to transfer y_i to c_l, then the only way is to have the job done (and monitored) by the authority. For example

Algorithm 2 (Authorized Transfer) *1. Client c_k shows possession of right y_i to the authority by following a procedure analogous to the one in the proof of theorem 3 (the logarithm being shown possession of is x_i). The procedure requires that c_k prove her knowledge of a_k, which allows the authority to authenticate the giving client.*

2. Client c_l shows possession of a_l to the authority in zero knowledge. In this way, the receiving client is authenticated by Auth.

3. The authority Auth gives y_i to client c_l using the procedure for granting new rights discussed in section 3 (the logarithm being granted is x_i).

6 Requirements and Conclusion

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_k to prove her identity a_k in order to use any subset of her rights, because a_k is the only secret parameter she holds.

As for the storage required, we have

Authority 1. Secret storage for logarithms x_i.
2. Secret storage for all client numbers a_k.
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' secret exponent 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_{k_i}, E_{k_i}, $0 \leq i \leq n_k - 1$.
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 A and B 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 their rights, 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* —compare to a network of teller machines shared by several credit card issuing corporations—.

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

Table 1. Functional dependencies.

References

[Chau 88] Chaum, D., Evertse, J.-H., and Van de Graaf, J. 1988. An Improved Procotol for Demonstrating Possession of Discrete Logarithms and Some Generalizations. *Proceedings of Eurocrypt'87*, Springer-Verlag, pp. 127-141.

[Domi 91] Domingo-Ferrer, J. 1991. Distributed User Identification by Zero-Knowledge Access Rights Proving. *Information Processing Letters*, vol. 40, pp. 235-239.

[Grah 72] Graham, G. S., and Denning, P. J. 1972. Protection: Principles and Practices. *Proceedings of the AFIPS Spring Joint Computer Conference*, pp. 417-429.

[Harr 76] Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. 1976. Protection in Operating Systems. *Communications of the ACM*, vol. 19, pp. 461-471.

[Linn 90] Linn, J. 1990. Practical Authentication for Distributed Computing. *Proc. IEEE Symposium on Research in Security and Privacy*, IEEE CS Press, pp. 31-40.

[Rive 78] Rivest, R. L., Shamir, A., and Adleman, L. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, vol. 21, pp. 120-126.

[Woo 92] Woo, T. Y. C., and Lam, S. S. 1992. Authentication for Distributed Systems. *IEEE Computer*, vol. 25, pp. 39-52.