DIVISION OF COMPUTER SCIENCE

# The Notary

Technical Report No. 153

Marie Rose Low

October 1992

# THE NOTARY

Marie Rose Low
School of Information Sciences,
The University of Hertfordshire.

October 1992

## Abstract

In this document we discuss those functions performed by a notary and the requirements that such a functionary places on its environment, location and communications. The discussion takes place with specific reference to the notary's role in DODA, (a Distributed Office Document Architecture), but keeps more general needs in mind. Finally a way of implementing a notary in UNIX using UATP, (Unix Access Table Protection), is described.

## 1. Introduction

A notary is a functionary which provides a service to clients who wish to seal their data so that any tampering of that data after it has been sealed can be detected. 'Sealing data' is a term used in this document when data is put through an algorithm to produce a value which is dependent on that data. Changes to that data should then be reflected in the value produced by the algorithm. If data is sealed and the value appended to it, then anyone receiving that data (and value) who knows the algorithm used, can check that the data has not been modified. To detect tampering, the receiver of that data puts the data through the same algorithm and compares the output value with the one it received with the data. If these are the same then there is a very high probability that the data has not ben touched. This is essentially the property of any checksum algorithm. However not all algorithms will detect all modifications to the data. Some algorithms are stronger than others and better able to detect a larger range of changes to the data. Cryptographic algorithms are regarded as being sufficiently strong so that there is a very low possibility of the data being modified and this modification not being detected.

When such an algorithm incorporates a personal element which is different for each functionary (and known only to that functionary) using the algorithm to seal the data, the term used is 'signing data'. A notary therefore, is a functionary which will accept data from any client that has access to it and which will put this

data through a strong algorithm which produces a value. If this algorithm uses a personal element, then the notary 'signs' the data and the signing of that data can always be attributed to that notary. In this document the notary seals data using cryptographic techniques. The techniques available are discussed in the following section but whichever method the notary uses, its function remains the same. In this document the notary is discussed with specific reference to such a functionary as used in DODA [Sno92]. The term anti-notary is used for the service that is available to clients to verify the sealing of the data.

**Sealing Data**

The point of sealing data is to provide data integrity and not secrecy. However cryptographic algorithms, used in a slightly non-standard manner, are well suited to this task. Cryptographic algorithms have two functions, Encryption, which scrambles the data into cypher text, and Decryption which converts the data back to plain text. Sealing data however just requires one function which is used to produce the same value when the data is sealed and when it is verified. A function which produces a value that is dependent on the input data, and for which there is no inverse (so that the data can be obtained from the value) is a One Way Function (OWF). The value produced from the algorithm must be dependent on all the bits of input data. There are strong hash algorithms which are used as OWFs and standard cryptographic algorithms, such as DES and RSA which are also used as OWFs. The difference between using an encryption algorithm such as DES to seal data as opposed to encrypt data, is that when sealing data, the encryption function is used both to produce the value and to check it and the data remains in clear, and when encrypting data, the data is transformed by the encryption function into cypher text and the decryption function must be used to return it to plain text.

The main requirement when sealing/ signing data, is that a third party cannot get hold of the data, modify it and produce a new value that will be correct when the data and value are verified. This can be achieved in two ways, either the OWF is kept a secret so that no unauthorised third party can use it to produce a correct value after modifying the data, or the algorithm is public but it makes use of a secret key known only to the interested parties. When using a hash algorithm that does not involve a secret element, then the hash algorithm must be kept securely by those that use it, and each must trust that the others will not reveal the algorithm. If the DES encryption function is used, then the party that seals the data and the party that verifies it need to have the same secret key to produce the same value. If every two such parties have their own secret key, then if a secret key is compromised only that pair is affected. If RSA is used then sealing/ signing data involves only the RSA key pair of the party that is performing the function. The secret key is known only to this party and the public key can be made known to all, therefore the party can sign the data and anyone can verify the integrity of that data. The value produced by using RSA in this way is a 'signature block'. The only remaining requirement for this method is that the parties verifying the data are assured that the public key belongs to the party signing that data and not an intruder.

### Why RSA ?

The notary discussed in this document is a server that uses the RSA algorithm to seal/ sign data. Clients that are not able to seal their own data call upon the notary to seal their data for them. The main advantage in using RSA is that there are no shared secrets; the notary signs the data with its RSA secret key (and its public key) and it is the only one that needs to know this secret key; the data is verified using the public RSA key only. This means that **any** anti-notary, not just a specific one, can verify that data because the public key can be made known to everyone. Whoever has access to that data can check its integrity by calling on a trusted anti-notary and providing it with the public key.

A second advantage of RSA is that data signed in this way is always attributable to the notary that signed it. As each notary has its own RSA key pair, and therefore a secret key known only to itself, then any data that can be verified can only have been signed by the notary that holds the secret key associated with the public key used in the verification process; and valid public keys are always identified with particular notaries; a notary cannot deny signing such data. If a notary is compromised the data that was signed by that notary and is no longer trusted can be easily identified.

## 2. General Requirements

### Environment

The notary has to be accessible to all who wish to seal data. It is best seen as a 'server' with 'clients'. It does not need to have the absolute security of the DODA visibility server (once established all users believe in the visibility server's integrity and it to checks that all other functionaries have performed their tasks correctly), but it must exist in a situation where a client, including the DODA server, can either be assured of its integrity or have a means of verifying it at the time of use. It must have secure storage for its secret RSA key.

In a distributed heterogeneous environment the location and integrity of the notary may have to be provided in a different manner for each system. In general it will be viewed as an object which is a server to the clients who use it. The notary must either be trusted or certifiable as must its cryptographic keys. If the notary is violated then the keys are assumed to be compromised.

### Location

How many notaries are needed and where should they be situated? Is it one for each client? One for the whole system (including the distributed case as well)? One for each system? There is certainly a requirement to have a highly trusted notary to be used by the visibility server. Communications between this notary and the server have also got to be highly trusted. Each user must also have access to a notary that is in its own management domain, but users within that domain may share one notary. If the notary was in a different domain, then there must be very secure communications between the notary and user. There must be a notary therefore which is accessible to all, but in a 'secure' situation

(not a part of the operating system, but a server protected by the environment provided by the operating system) within each management domain.

### Functions

The purpose of the notary is to provide a client with the ability to perform a function on his data which leaves the data readable and yet enables that client or any member of a widely dispersed group to detect any modifications to that data. The notary performs the following functions;

a. Generate an RSA key pair.
b. Make its public key known.
c. NOTARISE data.

A question which needs to be addressed is who authenticates the clients who present data to be sealed ? Should the notary assume that this has been done at some other stage ? In the DODA environment, authentication is performed at the final stages of committing the transaction.

There is also an anti-notary used to check the notarised data. This needs the same RSA facilities as the notary and the notary's public key.

### Communications

How does the data get to the notary in its original form?

The client must have the means of ensuring that the data he has notarised is the data he asked to be notarised and the DODA server must be sure that data was notarised as presented (is it sufficient to have a trusted notary?). The data must therefore either be passed to a local notary as parameters in a direct call, or it must get to the notary via secure mail/ secure communications or the client has to check the notarised data against the original data.

### Issues

The anti-notary which checks the integrity of the data must be assured that the public RSA key he is given is owned by the real notary and not an impostor. This may then call for public keys to be certified, so that impostors can be eliminated. Certification is carried out by a trusted agent, a key certification functionary (called a key registry in this document) which verifies that the notary, or any other such entity, is authentic, and then certifies that entity's public key. The key's certification can then be checked by whoever uses that key and if the certification is found to be valid, the key is known to be genuine. This may lead to another notary method to get the key certified.

The following argument shows the need for public key certification. If the integrity of the notary or its communications cannot be ensured and the client has to check the signed data against the original data then it needs an RSA facility (an anti-notary) and the notary's public key to check the signature. If this public key is to be saved by the client then it must be stored such that either no-one can overwrite it with another key. (can a user client have this security/assurance?) or the public key must be certified as described above. Then the anti-notary can verify that the data was signed by an authorised notary and not an impostor. Also, if the notary's public key can be distributed to any anti-notary, then the key

must be certified beforehand, otherwise there is no guarantee that it has not been replaced by a subversor's public key; all the subversor then has to do is replace the original signature block with one signed by his secret key.

If the key is certified, then the client wanting to use the notary's public key must be able to check the certification i.e. it needs to be able to use a key certification algorithm (this could be the same as an anti-notary).

Alternatively if the data can arrive at the notary safely, then the client does not need to check the signed data against the original. This means that the client can trust that the correct data has been notarised. He still has to trust the notary and its public key i.e he has to be sure that a bogus notary has not tampered with the data and then signed it with his own key. If the notary's public key is certified then the user has a higher level of assurance that the notary is trustworthy because the bogus notary should never have been authenticated and its key certified.

If the notary's public key has to be certified to maintain its integrity, there is now a need for a key registry service (how is this to be implemented?). The notary therefore also needs a key registration procedure, to certify its public key and obtain the key registry's public key, and other methods, to store these certified public keys and to check certification. (The key registry is a central functionary which uses its own secret to certify other keys. Can this function also be performed by the visibility server? In this case the server needs a co-located notary facility and all other notaries have to identify themselves to the server to get their keys certified.) Any server that checks the validity of the data also needs the ability to check the key certification and the key registry's public key e.g. anti-notary (the anti-notary must be trusted or certifiable). Either or both the notary and anti-notary must be trusted.

## 3. Notation

A notation is described in this section which is used to both express more clearly and expand upon the restrictions and implications of the options described in the previous section.

| | |
|---|---|
| [] | trusts/ believes |
| [] | trustworthy/ secure |
| {} | verifiable |
| <= | if |
| \| | or |
| & | and |
| = | is the same as |
| $^{x}f$ | subject f is in management domain x |
| $data_{key}$ | data notarised, integrity maintained by encryption with key |
| $key_{CK}$ | key certified with key CK which is the key registry's public key |

NSK notary's secret key

NPK notary's public key

C client

If a client C in management domain X is to trust that his data has been notarised correctly, $data_{NSK}$, then he must have access to one of the following; a trusted notary which is local to C, trusted communications to a trusted notary that is not local to C, no trusted notary or communications but a trusted, local anti-notary. In other words the following conditions must be true:

$^{x}C[data_{NSK}]$ <= $^{x}C[^{x}notary]$

| $^{x}C[^{y}notary] \& {}^{x}C[data]$

| $^{y}notary \& {}^{x}C[^{x}anti\text{-}notary] \& \{data_{NSK} = data\}$

The notary uses RSA to notarise the data and so needs an RSA key pair. Therefore,

$^{x}C[^{x}notary]$ <= $[^{x}notary] \& [NSK] \& {}^{x}C[NPK]$

$^{x}C[^{y}notary]$ <= $[^{y}notary] \& [NSK] \& {}^{x}C[NPK]$

C can trust NPK if it is a certified public key. This implies that the key registration function must be performed either by a new extremely secure functionary or by an existing functionary such as the DODA server. (CK is the key registry's public key. All keys are certified with the key registry's secret key. This certification process must positively identify all entities that want to register a key before producing a certified copy of that key. All trusted functionaries that use a certified key will always check the certification of a key before putting it into use. If the DODA server has a secure copy of CK, then even if the notary is violated and NPK and CK compromised and substituted, the DODA server will detect it when it checks notarisation.)

$^{x}C[NPK]$ <= $NPK_{CK}$

$NPK_{CK}$ <= $[^{z}key\ registry] \& CK$

When C and the notary are located in the same management domain then the method call to the notary must ensure that the data passed in the call is handled securely. When C and the notary are in different management domains then there is long haul communications involved so if C is to trust the data it must either use a secure link.

$^{x}C[data]$ <= $[data]$

or an anti-notary has to exist to enable the receiver of data to verify the notarisation. The client can make use of the anti-notary to check his own notarisation if,

$^{x}C[^{x}anti\text{-}notary]$ <= $[^{x}anti\text{-}notary] \& {}^{x}C[NPK]$

(C and the anti-notary have to exist in the same management domain or have secure communications in which case we are back to square one. This

requirement, [data], brings into play the whole area of secure long-haul communications which is not being discussed.) It does mean that for every request to notarise data C, has to anti-notarise it and also check that the correctly notarised data is the same as the data C originally sent.

A trusted anti-notary is required for functionaries to verify notarised data received from other functionaries. The DODA server in particular needs its own co-located anti-notary to check notarisation before accepting a "doculett".

If the two functionaries are to be trustworthy, i.e.

[$^{x}$notary] and [$^{x}$anti-notary]

then they must either be protected from violation by the operating system or certifiable by C. (The point of this exercise is to see what an operating system can do to enable these functionaries to remain trustworthy so we shall not deal with the possibility of the client certifying the functionary before use.) Furthermore, if the notary is to be trusted the system must provide an environment in which NSK for the notary can be kept secure and protected from read and write access by anyone other than the notary.

In conclusion, for C to trust the notarisation of his data, one of the following must be satisfied:

$^{x}C[data_{NSK}]$ <= [$^{x}$notary] & [NSK] & [$^{z}$key registry] & CK & $NPK_{CK}$

| [$^{y}$notary] & [NSK] & [$^{z}$key registry] & CK & $NPK_{CK}$ & [data]

| $^{y}$notary & NSK & [$^{x}$anti-notary] & [$^{z}$key registry] & CK & $NPK_{CK}$ & {$data_{NSK}$ = data}

The whole point of the notary is to provide data integrity therefore assume that in general it is local to the client as otherwise we need to provide either some other means of data integrity for long-haul communications to a remote notary anyway or a local anti-notary. It may be that a remote notary is used if the local one goes down and in this case either the trusted anti-notary must come into play or this extra facility for secure communications must be available.

## 4. User Authentication

When a user proposes a change to a DODA document the user has got to be authenticated before the change is accepted. Is it the responsibility of the notary to authenticate users before it notarises their request? In DODA it is important that the user who starts a transaction is the only user to participate in that transaction or to delegate work to other users. It is therefore important that a user who starts a transaction can be attributed with that transaction when it is committed.

The method of authentication is also important. It is possible for users to have smart cards that employ algorithmic techniques to authenticate users. This is not a solution that will be found in all systems that make up a heterogeneous distributed system as this requires card reader terminals which are not common

to all systems. If users employ any form of cryptographic technique to authenticate themselves then they have to have a means of keeping their cryptographic secrets secure which is always a problem. Smart cards do solve this problem but as already mentioned, are not available everywhere. If users do use cryptographic techniques, then why don't they use RSA? If they use RSA can they not sign their own data and remove the need for a notary anyway? This would put extra strain on a key registry, but is this a problem? Does this authentication also have to be applied to the other functionaries that also use the notary, such as the sub-agent and server itself or just users?. One of the main points of DODA is that only the server has to be kept really secure and it will detect any malfunction in any other functionary, notaries included, but if all clients, functionaries included, to the notary have to authenticate themselves this may require that each functionary keep a secret secure.

The method of user authentication cannot be standard in a heterogeneous distributed system, therefore the requirement is that users can identify themselves to the same level of assurance on all participating systems. Assuming that users are going to be authenticated and that this is done using an algorithm that does not require that the user keeps some secret securely (e.g. the 'challenge', where 'personal' questions that the user has initiated are asked of the user) there is still the question of who does the authentication. There appear to be three options:

a. Assume that the systems' login sequence is sufficient to authenticate users logging into a system (this must include remote login, therefore one system must trust another system's login procedure - not very satisfactory). A further assumption is that a legally logged in user cannot take on another user's id. - this assumption is not necessarily guaranteed.

b. Let the notary authenticate all requests to notarise data. The notary may need to know that a user is allowed to use its services (by presenting a capability or using ACL) but this does not mean that the user has been authenticated - it does not show up an impostor. If the notary does the authentication and notaries are trusted functionaries, then is it reasonable for remote systems, such as the one housing the visibility server, to believe in the authentication? If the server can verify that the notary is known to it and it has not been violated then this seems reasonable. Can the server know this by checking the notarised data with the notary's certified public key that it, the server, has stored safely? I think that is sufficient (unless of course the notary's secret key is no longer secret and the violation has not been detected).

If the notary is also the user authenticator, then does the notary have to authenticate every request to notarise in every transaction? Some of the notarisation in DODA is requested by other agents on behalf of the user, so the notary is not always talking to the user. Do these agents also need to be authenticated? Instead of authenticating every request, the notary may only authenticate the initial request and then provide a nonce which must accompany all following requests. Does the notary then have to know when a transaction has terminated and the nonce becomes invalid, i.e. does the notary have to keep track of user sessions? This puts extra load on the notary and the user/ functionary is then restricted to using one notary for each session. If this

notary goes down the client cannot use another one without restarting the authentication process.

c. Let the visibility server authenticate users. The visibility server receives the first request from a user for a base reference for a document. The server checks this request against its ACL to see if the user is allowed access to that document. If the user is recognised, then the server can proceed with the transaction. The subsequent parts of the transaction are mostly between the user and the other functionaries and the server has no communications with the user until the end of the transaction. Can the server then assume that all work done by and on behalf of that user until the transaction is to be committed needs no authentication until the last stage or does each stage need to be verified? Alternatively can the server authenticate the user at the start and provide the user with a nonce which is attached to each stage of the transaction that gets to the server? The server is certainly aware when the transaction ends and so can invalidate that nonce more naturally than the notary.

## 5. Notary Implementation

In general the notary is best implemented as a server, if possible with no hardware support, though in some cases this may be necessary to store the secret key - in which case the question arises, why not have the whole of the notary in a black box? This ensures that not only are the keys secure but that it is much more difficult to violate the notary. The discussion however is to see how securely the notary can be implemented in software. The following assumptions will be made:

a. it is a server to the user, submission agent, archive-gnome and DODA server all of which are its clients
b. it is to be implemented as an object type with methods
c. communications between client and notary uses IPC.
d. there may be several notaries, one per client, one per machine or one for the whole system, but assume one per machine.
e. the notary is to be implemented outside the kernel but not in each user area. The DODA server may have its own co-located notary.
f. a key registry exists to certify public keys (where the key registry exists and how certification takes place is not discussed in this document) and the notary is a client to it.
g. IPC call on the one machine is secure i.e. if a method is called with certain parameters, those parameters will not be modified before reaching the server.

### UNIX Implementation using UATP

(The notation used in this section is that used for UATP [Low92].)

System manager allocates three secure areas, P, X and Y.

The methods for the notary are released and verified before use and live in P. P also has the UATP methods. The stubs for calling the notary exist in X as do any of its instances. The UATP stubs and its type instance, the access control table, exist in the secure area Y.

The world has read permissions for P, and both X and Y (as world or as the same group) have permission to execute the methods in P. Only users executing with X and Y's permissions have read and write permissions to anything in X and Y, and the world has execute permission for the stubs.

The only output generated by the notary is the RSA key pair (this is the type instance, the notarised data is a response to the call of the notarise method) and the secret key, resides in X. The secret key is therefore only as secure as a secure user area can be in UNIX with the permissions set as described above. (physically, the drive that holds the key must be kept securely and not copied). It is assumed that an impostor cannot become system manager/ root, that an impostor cannot log into X and read the key file, that the IPC call parameters are passed securely to the notary without modification and that the system/ notary software has not been compromised such that it leaks the secret key when the notary uses it.

The notary has 4 methods:

a. generate RSA key pair
b. certify the public key
c. store the certified key
d. notarise data

The DODA coordinator (on that machine at least) calls the UATP methods to create an access control table for the notary and to enter in the table, the names of the clients that are allowed to use the notary.

The client then calls the notary with a request to notarise its data as follows:

$$\text{Client}({}^{1}NS_{x}^{x}(MNOTAR_{p}^{x}({}^{1}TS_{y}^{y}(TREAD_{p}^{y}\text{—>}UT_{y}))\text{—>}SK_{x}))$$

where NS is the notary stub, MNOTAR is the notarise method, TS is the UATP stub, TREAD is the UATP method to check access control UT is the UATP table and SK is the notary's secret key.

The notary stub is called by the client and it receives the data to be notarised as a parameter (reference or value) from the client. This is passed to the notarise method. This first of all checks with the distributed access control table, using TREAD, that the client is allowed to use the notary, and if he is enabled it then uses its secret key to notarise the data. The signature block produced by notarisation and the certified public key are returned to the client.

**Issues**

How is the data to be notarised passed to the notary? If the actual data is being passed, this could be a great number of bytes. If a reference is to be passed, then how does the notary method, that is running with X's userid and permissions, get permission to read the clients data? The 'doculett' can be a file in the client's user area with read-only permission for the world, the file name is passed as an argument, so that the notary can have access to it. There is no reason why this should not be sufficiently secure within the limitations of UNIX.

**Slightly different approach.**

As the instances of the notary i.e. the key, will only ever be used by the notary and no other methods, it is possible to have both methods and instance (notary and key) reside in X. The notary methods run with the caller's id and permissions and so have access to the users data.

$Client(MNOTAR_x^{client}(^1TS_y^y(TREAD_p^y \text{—>} UT_y)) \text{—>} (data_{client}))$

This would only work if the notary's secret key were a part of the notary code as the methods would not have permission to access any data 'owned' by X, as it is now running with the client's id and permissions. A regeneration of the keys would then require a new installation of the notary methods.

**References**


[Low92] Low M.R., March 1992, *Fine Grained Protection in UNIX* Hatfield: Hatfield Polytechnic Computer Science Division. Technical Report.

[Sno92] Snook J.F., September 1992, *Towards Secure, Optimistic, Distributed, Open Systems*: Hatfield: University of Hertfordshire, Computer Science Division: Thesis (PhD).