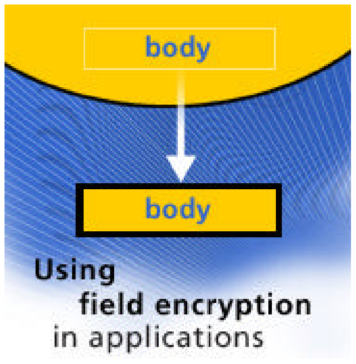


by
Richard
Schwartz

Level: Intermediate
Works with: Domino 5.0
Updated: 09/04/2001

Since its first release more than ten years ago, Lotus Notes has been the premier solution for sharing information in a corporate environment. From day one, the developers at Iris Associates realized that for sharing to be successful, there had to be a strong security system in place so that sharing could be limited.

That sounds a bit paradoxical, but it reflects a basic reality: users won't put information into a system if they don't trust that the system will only give that information to the right people. One of the key concepts that the developers of Notes understood was that although programmers and system administrators are very cool, upstanding, and important people, some users don't necessarily want to have to trust them with all their information. To solve this problem, the developers at Iris turned to the modern version of an ancient technology: encryption.

This article explains the basic theory of encryption. It then shows you how to implement Domino's field encryption feature in a version of the Document Library template so that you can easily build encryption into your own applications. This article assumes you have an intermediate level understanding of Domino Designer and Domino application development.

The theory of secret key encryption

Encryption, in layman's terms, is the application of transformations to data in order to hide information while it is stored or communicated. The study of encryption techniques and related technologies is known as cryptography. RSA Security is a company that pioneered the commercialization of so-called "public key" encryption techniques based on complex mathematical transformations, and Lotus Notes was the first major commercial application to adopt their technology. Today, Lotus has the largest installed customer base of Public Key Infrastructure (PKI) users in the world.

Notes and Domino use encryption in several different ways. This article concentrates only on field encryption. You can find out more about the other ways Notes and Domino use encryption in the *Iris Today* article, "Notes from Support: Notes encryption: Locks for a digital world."

Before we discuss the details of field encryption in Notes, let's take a slightly deeper look at what encryption is. (For a *really* deep look at it, check out *Applied Cryptography*, by Bruce Schnier, but be aware that this book may lead to nightmares for readers who are math phobic!) I described encryption above as "the application of transformations to data in order to hide information." If I had wanted to make you work a little to figure out the definition, I could have written this instead:

uif bqqmjdbujpo pg usbotgpsnbujpot up ebub jo psefs up ijef jogpsnbujpo

This is an example of the simplest form of encryption: a character substitution cipher. Every occurrence of each letter in my original sentence has been replaced by the very next letter in the alphabet, with *z* wrapping around to *a*. I could have done this by hand quite easily, using an equivalence table like this:

Plain	a	b	c	d	e	f	g	h	i	j	k	l	m
Cipher	b	c	d	e	f	g	h	i	j	k	l	m	n

Plain	n	o	p	q	r	s	t	u	v	w	x	y	z
Cipher	o	p	q	r	s	t	u	v	w	x	y	z	a

The encryption method simply consists of finding a character in the top row and replacing it with the corresponding character in the bottom row. The decryption method is the reverse. Some of you probably learned this technique the same time that I did—from your PF Flyers Secret Decoder Ring. This type of encryption really is a toy. It's almost too simple to even call it encryption, but it does make a good illustration of two things. First of all, it demonstrates that the security of the data is dependent on the secrecy of the key. The layout of the equivalence table must be known to everybody who needs to read messages, but it must not be disclosed to anyone else.

Second, a character substitution cipher illustrates the fact that computers are naturally well-suited to do encryption. Computers always represent text as a series of bits. Most commonly, they use ASCII or some other well-known character set to represent text, but software can easily be written to use a different representation. To show this, here is the LotusScript code for the character substitution method just described. I generalized it so that a parameter that ranges from 1 to 26 determines the offset of the two rows of the table. (For example, the table above represents the algorithm with the keynum parameter set to one.)

```
Sub encipher(keynum As Integer, ptext As String, ctext As String)

    Dim i As Integer
    Dim c As String
    Dim n As Integer

    For i = 1 To Len(ptext)
        c = Lcase(Mid(ptext,i,1))
        If c < "a" Or c > "z" Then
            ctext = ctext & c
        Else
            n = Asc(c) - Asc("a")
            c = Chr( ((n + keynum) Mod 26) + Asc("a")  )
            ctext = ctext & c
        End If
    Next i

End Sub
```

The main reason I'm showing you this code is to illustrate the fact that this simple encryption method is what is known as a *symmetric* encryption algorithm. In the above code, the input is the variable ptext, which stands for plaintext—the term that cryptographers like to use for unencrypted data. The output is the variable ctext, which stands for ciphertext. The variable keynum is an integer between 1 and 26. If you look carefully at the code, you will see that it processes characters one at a time, converting them to integers in the range 0 to 25. Then it does simple integer addition and modulo operations and finally converts back to a character. The reason we call this technique symmetric is that all the operations are reversible. The same keynum value can be plugged into a decryption algorithm that takes the ciphertext and the same key as input and reverses those operations to reproduce the plaintext. Here's the corresponding decryption algorithm.

```
Sub decipher(keynum As Integer, ctext As String, ptext As String)
```

```
    Dim i As Integer
    Dim c As String
    Dim n As Integer

    For i = 1 To Len(ctext)
        c = Lcase(Mid(ctext,i,1))
        If c < "a" Or c > "z" Then
            ptext = ptext & c
        Else
            n = Asc(c) - Asc("a")
            c = Chr( ( (26 + n - keynum) Mod 26) + Asc("a")  )
            ptext = ptext & c
        End If
    Next I

End Sub
```

Apart from the reversing of the ptext and ctext variables, there are only two differences between this decryption code and the encryption code. The most significant difference is that keynum is subtracted instead of added to each character value, thus reversing the direction of the character substitution. The other difference is the addition of 26, which assures that we don't give a negative value to the Mod operator.

In case you're worried about how insecure symmetric encryption is based on the above example, you can rest assured that much stronger symmetric encryption techniques have been known for a few centuries. This example is just about as weak as you can get. One of the critical factors used to evaluate the strength of encryption is the key length, and this example has a key length of five bits, because the key values of 1 to 26 can be represented as a five bit number. Notes uses a much stronger symmetric encryption algorithm, with a larger key, for field encryption. Several sophisticated symmetric algorithms that work well with large keys are in common use, and they can provide very strong security for your data.

The theory of public key encryption

It wouldn't make much sense to talk about symmetric encryption if there wasn't also an asymmetric version, and indeed there is. The basic principle behind asymmetric cryptography is that you can construct a pair of mathematically related keys, *a* and *b*, and make *a* public; but it is extremely difficult to figure out *b* even though you know the relationship algorithm. This means that even a simple asymmetric algorithm has to be based on a type of mathematical problem that is often called a "one-way trap door function." One example of such a function is multiplication of two prime numbers, resulting in a composite number as a product. By picking the factors large enough to make the product as long as you want, you can assure that it will take an unreasonably long time for a very fast computer to factor it back to the two original prime numbers.

Asymmetric encryption was invented in the private sector in the 1970s by Whitfield Diffie and Martin Hellman and later improved by Ron Rivest, Adi Shamir, Len Adelman, and others. Many experts believe that both the NSA and British intelligence service had the required knowledge and ability to develop asymmetric encryption as many as ten years earlier, and it seems likely that some other security services probably had it too, but there is (quite understandably) no firm proof of who really developed it first.

Unfortunately, demonstrating an asymmetric encryption scheme in LotusScript is no easy task. The very nature of the trap door functions is difficult to explain, and a realistic example requires doing arithmetic with numbers that are larger than LotusScript knows how to work with. A concise explanation of the mathematics behind asymmetric encryption can be found in the **Lecture Notes for Use with *Cryptography and Network***

***Security* by Williams Stallings** under **Number Theory and Public Key Cryptography**. An example of Java source code for an implementation of asymmetric encryption can be found at **Cracking the Code**.

Asymmetric encryption is commonly known as public key encryption, because the actual encryption is always done using a key that doesn't have to be kept secret. Users disclose their public key, but keep the mathematically related private key secret. The very cool thing about public key encryption is that each user can be completely responsible for keeping his own private key secure, because it never has to be disclosed to anyone else. In symmetric encryption, every user must be trusted to keep a single secret key properly secured, and a single breach ruins security for everyone.

Encryption in Notes

In Notes, symmetric encryption is known as secret key encryption, and Domino Designer has features that allow developers to easily use it for encrypting fields in documents. Users can create secret keys and give these keys names. Applications reference the keys by their names in a special field called SecretEncryptionKeys. When a document is saved, the keys named in this field are retrieved from the user's ID file, and all fields marked with a special property are encrypted with those keys**.**

Notes uses public key encryption for electronic mail, and Domino Designer also provides developers with the ability to use it for encrypting fields in documents. Public keys are always associated with users. Applications reference the keys by the users' names in a special field called PublicEncryptionKeys. When a document is saved, all the user names in this field are located in the Domino Directory or the user's personal address book, the corresponding keys are retrieved, and all fields marked with a special property are encrypted with those keys.

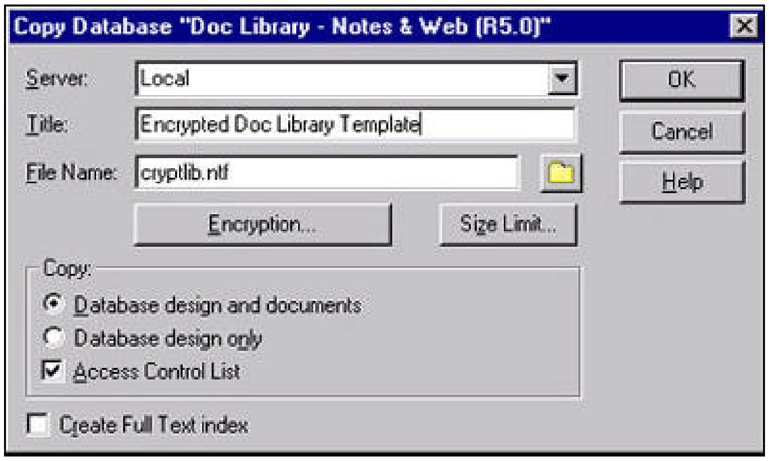
Implementing field encryption in an application

To demonstrate both secret key and public key field encryption, we'll implement it in a version of the standard Document Library template that ships with Domino. We'll give it the following features:

- The Body field in the Document, Response, and Response To Response documents can be encrypted.
- Encryption can be done using either public keys or secret keys.
- Secret keys will be selected from a list maintained in a profile document.

The overall goal is to demystify field encryption and make deployment of databases that use field encryption easy, even for end users. You can download and examine a copy of the finished template from the **Iris Sandbox.**

Step 1: Make a copy of the Document Library template
In Domino Designer, open the Doc Library - Notes & Web (R5.0) template (doclbw50.ntf). Select File - Database - New Copy from the menus, and save the copy with a different file name and title; for example, title it Encrypted Doc Library Template and change the file name to cryptlib.ntf.


Copy Database "Doc Library - Notes & Web (R5.0)"

Server: Local

Title: Encrypted Doc Library Template

File Name: cryptlib.ntf

Copy:

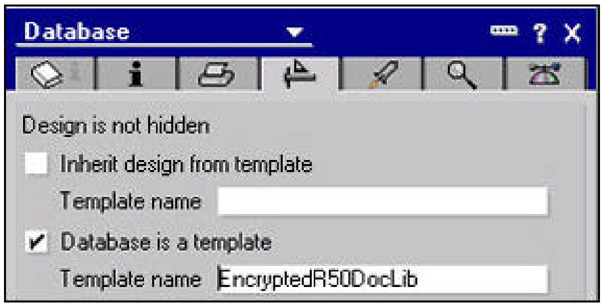
(●) Database design and documents

( ) Database design only

[✓] Access Control List

[ ] Create Full Text index

Close the Doc Library - Notes & Web (R5.0) template. Then, in order to make sure that you don't end up with two templates with the same design template name, open the template you just created, open the Database properties box, go to the Design tab, and change the database's template name; for example, change it to EncryptedR50DocLib.


Database

Design is not hidden

[ ] Inherit design from template

Template name

[✓] Database is a template

Template name EncryptedR50DocLib

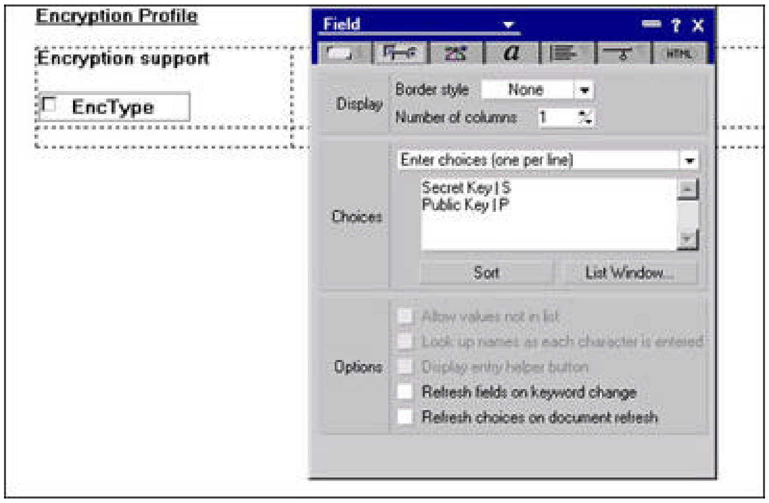
Step 2: Design a form for a profile document
Next, create a new form that will be used to generate a profile document.

When you start to make changes to your copy of the template, you will get a message stating "This database has been authorized for use by Lotus Notes Mail users. This action will invalidate the authorization. Are you sure that you would like to proceed?" The message is caused by a special licensing stamp that Lotus applied to the doclb50w.ntf template that you started with. You can safely click Yes and proceed with your work.

Now, start designing the form by put a heading, Encryption Profile, at the top of the form and creating a two-row, two-column table beneath the heading. These are merely aesthetic features—not part of the required functionality—but creating a nice layout helps with usability and maintainability of your work.

Add a checkbox field called EncType in the top left cell of the table on the form. This field will be used to turn each of the two encryption methods on or off for a particular database. Set up two values in the Choices property for this field: Secret Key | S and Public Key | P. (This is an example of aliasing in keyword fields. The Notes client will display the text to the left of the vertical bars, but store the values that are on the right of the bar. Code on other forms will look for the S or P value in this field in the profile document.)

Since the most common type of encryption used in applications is secret key encryption, set the default value of the EncType field to S. Then, in the top right cell of the table, enter a brief comment explaining the purpose of the field, such as "Check one or both boxes to enable encryption. Secret key encryption uses keys stored in user id files. Public key encryption uses keys stored in person documents in the Domino Directory or your personal address book."


Encryption Profile

Encryption support

☐ EncType

Field

Display: Border style: None; Number of columns: 1

Choices: Enter choices (one per line)

```
Secret Key | S
Public Key | P
```

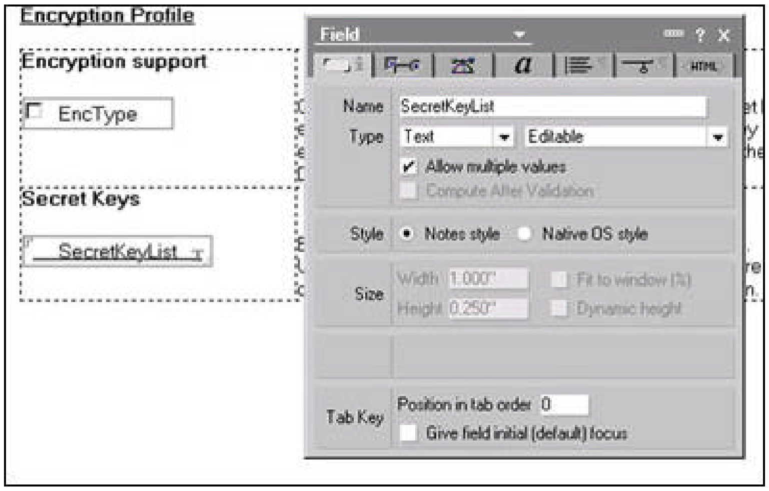
Sort
List Window...
Options
Allow values not in list
Look up names as each character is entered
Display entry helper button
Refresh fields on keyword change
Refresh choices on document refresh

Next, create an editable text field called SecretKeyList in the lower left cell of the table. Be sure to check the "Allow multiple values" box in the Field properties box. Set the Input validation formula for this field to:

```
@If(EncType = "S" & SecretKeyList = "" & @IsDocBeingSaved;
    @Failure("You must supply the name of at least one secret key");
    @Success)
```

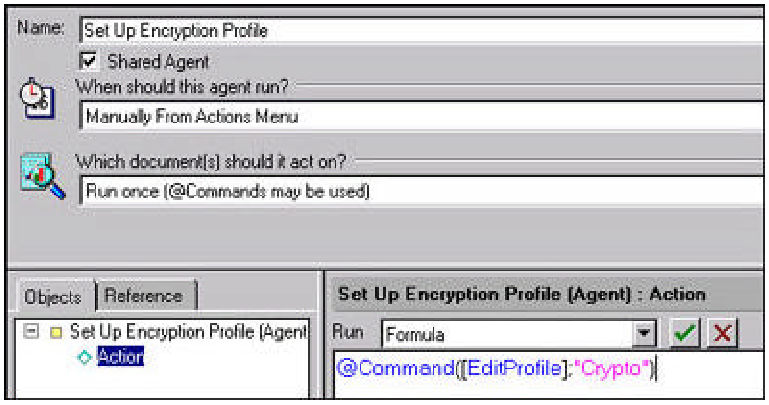
This field will be used to set up a list of all the secret keys that can be used to encrypt documents in a database built from this template. The validation formula assures that it is not left blank when Secret Key encryption is selected in the EncType field. Providing a list this way is a convenient way to make sure that users of the application don't have to remember obscure key names, and that they don't accidentally use an inappropriate key that might give unintended access to unauthorized users.

In the bottom right cell of the table, enter a comment such as "Enter the names of keys that are stored in user id files. Users who do not have these keys will not be able to read documents that are stored using secret key encryption."


Encryption Profile
Encryption support
EncType
Secret Keys
SecretKeyList
Field
Name SecretKeyList
Type Text Editable
✓ Allow multiple values
Compute After Validation
Style ● Notes style ○ Native OS style
Size Width 1.000" Fit to window (%) Height 0.250" Dynamic height
Tab Key Position in tab order 0
Give field initial (default) focus

Now, save the form with the name Crypto Profile | Crypto, and set the Form properties so that it will not be shown in the Create menu.

Step 3: Create an agent
The form we just created is used to create a single profile document that isn't visible in any of the database's views. If we just opened the form from the Create menu, a regular document would be created instead of a profile, so we'll use an agent to create the profile document instead. Simply create a new agent and do the following:
- Name the agent Set Up Encryption Profile.
- Check the Shared Agent checkbox
- Select "Manually From Actions Menu" for "When should this agent run?"
- Select "Run once (@Commands may be used) for "Which document(s) should it act on?"
- Select Formula in the Run drop-down box.
- Enter @Command([EditProfile]) in the formula box.

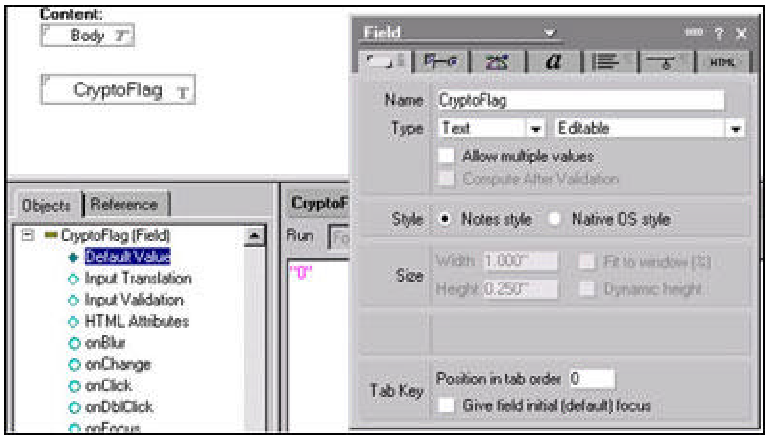

Name: Set Up Encryption Profile
✓ Shared Agent
When should this agent run?
Manually From Actions Menu
Which document(s) should it act on?
Run once (@Commands may be used)
Set Up Encryption Profile (Agent) : Action
Run Formula

```
@Command([EditProfile];"Crypto")
```

Step 4: Create three subforms
The next thing we'll do is create three subforms that are almost, but not quite, identical. Forms in the database will use one subform for plaintext documents, another for documents encrypted with secret key encryption, and another for documents encrypted with public key encryption.

These new subforms will actually replace a small section that is already on the existing forms in the doblb50w.ntf template, so the starting point is a cut-and-paste operation. Open the Document form and select the two lines that contain the "Content" label and the Body field. Cut them to the clipboard.

Now, create a new subform and paste the clipboard contents into it. Under the Body field, create an editable text field called CryptoFlag, with a default value of "0". On the Hide tab, set the hide attributes on this field so that it doesn't display in either Notes or a browser.



"0"

Name CryptoFlag

Type Text Editable

Allow multiple values

Compute After Validation

Style [✓ Notes style] Native OS style

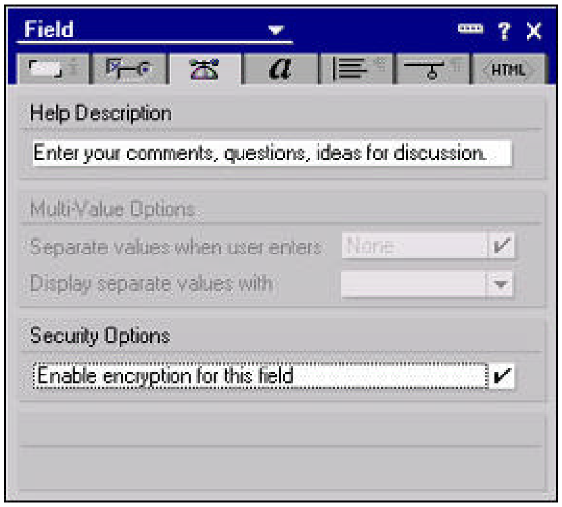
Size Width 1.000" Height 0.250" Fit to window (%) Dynamic height

Tab Key Position in tab order 0

Give field initial (default) focus

Save this subform with the name Plaintext Body | PlainBody.

Create another new subform and paste the same contents into it. Select the Body field in this subform, and open the Field properties box. On the Advanced tab, select "Enable encryption for this field" under Security Options.



Help Description

Enter your comments, questions, ideas for discussion.

Multi-Value Options

Separate values when user enters None

Display separate values with

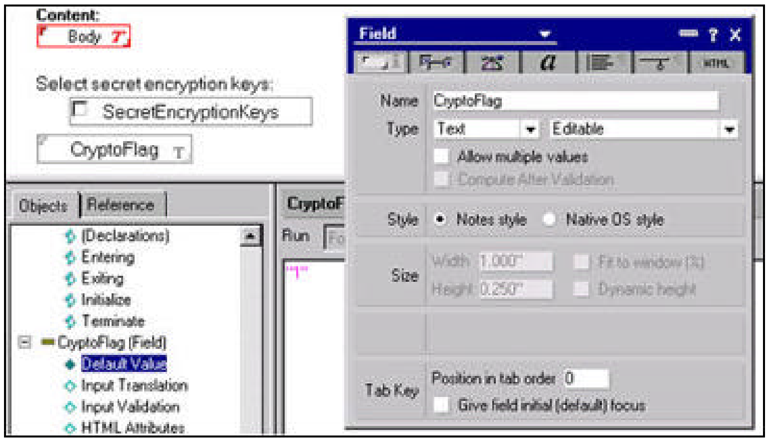
Security Options

Enable encryption for this field ✓

Beneath the Body field, add the field label "Select secret encryption keys" and then add a checkbox field called SecretEncryptionKeys beneath the label. In the Choices section of the Control tab of the Field properties box, select "Use formula for choices" and set the formula for the choices to:

```
@GetProfileField("Crypto";"SecretKeyList")
```

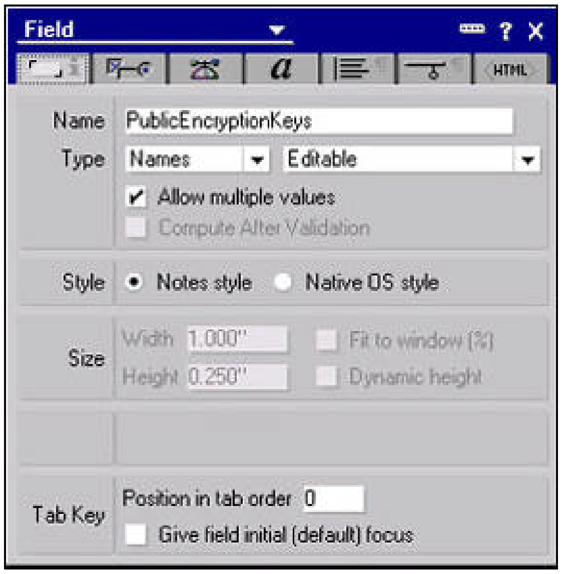
Add a new line at the bottom of the subform, and create an editable text field called CryptoFlag, with default value of 1. On the Hide tab, set the hide attributes on this field so that it doesn't display in either Notes or a browser.


Content:
Body
Select secret encryption keys:
SecretEncryptionKeys
CryptoFlag
Objects | Reference
(Declarations)
Entering
Exiting
Initialize
Terminate
CryptoFlag (Field)
Default Value
Input Translation
Input Validation
HTML Attributes
Field
Name CryptoFlag
Type Text Editable
Allow multiple values
Compute After Validation
Style ● Notes style ○ Native OS style
Size Width 1.000" Height 0.250" Fit to window (%) Dynamic height
Tab Key Position in tab order 0
Give field initial (default) focus

Save this subform with the name Secret Encryption Body | SecretKeyBody.

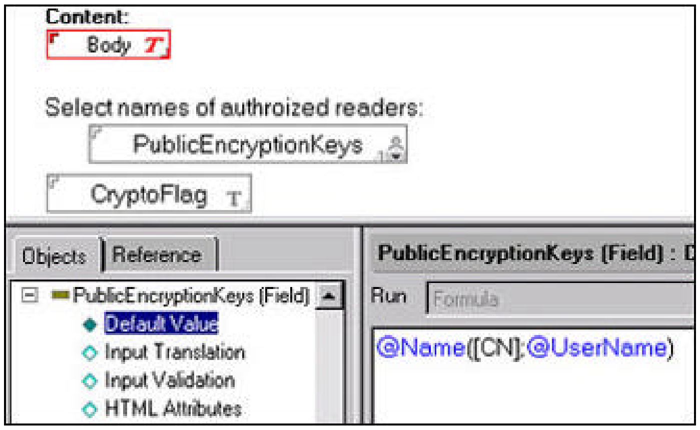
Now go back to the view of subforms, select the Secret Encryption Body subform, and copy and paste it (you can simply press Ctrl-C followed by Ctrl-V) to make a new copy of it. You will get a message saying "The source database 'Encrypted Doc Library Template' is a Design Template named 'EncryptedR50DocLib'. After being pasted, would you like these Subforms to be automatically updated when those in 'EncryptedR50DocLib' change?" Answer No to this message, since you are pasting the subforms into the same template they came from.

Open the newly-created copy of the subform, bring up the Subform properties box, and change the subform's name to Public Encryption Body | PublicKeyBody. Next, select the SecretEncryptionKeys field. Open the Field properties box, change the field name to PublicEncryptionKeys, change the field type to Names, and check the box to allow multiple values in the field.


Field
Name PublicEncryptionKeys
Type Names Editable
✓ Allow multiple values
Compute After Validation
Style ● Notes style ○ Native OS style
Size Width 1.000" Height 0.250" Fit to window (%) Dynamic height
Tab Key Position in tab order 0
Give field initial (default) focus

On the Control tab of the Field properties box, select “Use address dialog for choices” in the Choices section and “Display entry helper button” in the Options section. Finally, set the default value of the field to:

```
@Name([CN];@Username)
```



```
@Name([CN];@UserName)
```

Save these changes to the subform.

Step 5: Modify the forms
Next, we need to modify the existing forms in the template to make them use the new subforms. Go back to the Document form, and position your cursor where the field label *Content* and the Body field used to be. (If you've lost your place, it should be the second blank line beneath the visible header section of the form.) Choose Create - Insert Subform, select the "Insert Subform based on formula" checkbox in the Insert Subform dialog box, and click OK. Then enter the following formula as the computed subform's default value:

```
options := @GetProfileField("Crypto";"EncType");
oldSubform := @If(CryptoFlag = "0";
                "PlainBody";
            @IsAvailable(SecretEncryptionKeys);
                "SecretKeyBody";
            @IsAvailable(PublicEncryptionKeys);
                "PublicKeyBody";
            options = "";
                "PlainBody";
            ""
            );
respList := @Trim("No Encryption" :
            @If(options = "S";"Secret Key Encryption";"") :
            @If(options = "P";"Public Key Encryption";"")
            );
response := @If(  oldsubform !="";
                "";
            @Prompt([OKCANCELLIST];"Encryption Options";
                "Please select how you want this document encrypted.";
                "No Encryption";
                respList)
            );
subformList := @Trim("PlainBody" :
            @If(options = "S";"SecretKeyBody";"") :
            @If(options = "P";"PublicKeyBody";"")
            );
@If(      oldSubform != "";
            oldSubform;
        response = "";
            "PlainBody";
        @Replace(response;respList;subformList)
 )
```

For new documents, the above formula prompts the user to choose whether the document should be encrypted and what type of encryption to use. The formula only lets the user pick an encryption type if it was selected in the application's profile document. For existing documents, the formula relies on the fact that the CryptoFlag value was set when the document was created, and it also relies on the facts that the SecretEncryptionKeys field was created only if secret key encryption was used when the document was created and the PublicEncryptionKeys field was created only if public key encryption was used.

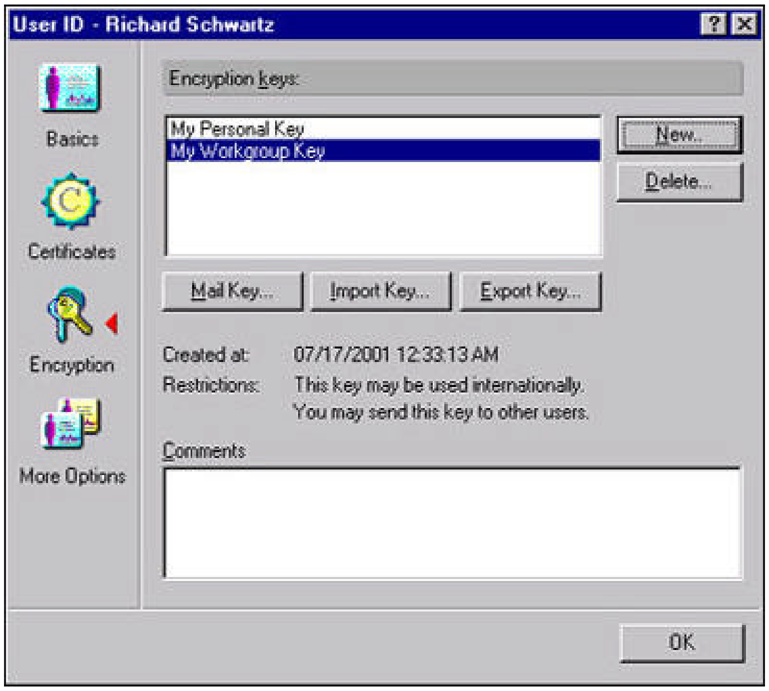
To include these same options in the other forms in the application, copy the computed subform field to the clipboard. Then open the Response and Response to Response forms. Remove the field label *Content* and the Body field from each form, insert a computed subform in their place, and paste in the same formula.

You can now save and close all the design elements that you have open in the Domino Designer. The coding for the template is complete.

Creating and deploying secret keys

If you want to use secret key encryption, the first thing you will need to do is create the keys that you want to use with your application. Keys are kept in your Notes ID file.

From the Notes client, choose File - Tools - User ID and enter your password. In the User ID dialog box, click the Encryption icon, and then click the New button. In the New Encryption Key dialog box, give the key an appropriate name, for example, My Personal Key. To demonstrate the ability of the application to handle multiple keys, click the New button again and create a second key called My Workgroup Key.



My Personal Key
My Workgroup Key

Created at: 07/17/2001 12:33:13 AM
Restrictions: This key may be used internationally.
You may send this key to other users.

If you want to keep data private for your own exclusive use, you can use a key that you don’t distribute to anyone. If you do that, and if you keep your ID file secure with a good password that nobody else knows, then you can keep personal information on any server without having to worry about whether the server administrators can read it.

If you want to share data with some people but not others, then it is up to you to distribute a secret key to the users who need them in order to use your application. Whether or not you distribute a key to others, it is very important to make sure that you make a backup of your ID file whenever you generate a new key, and that you keep it in a secure place. If all copies of a key are lost, all data encrypted with that key will be unreadable. It is also very important that you, and all the co-workers you distribute your key to, keep your ID files secure with a good password that nobody else knows.

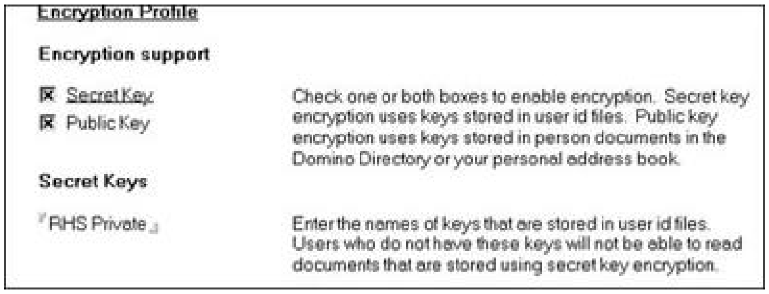
Notice that you have two key distribution options in the User ID dialog box: Mail Key and Export Key. The easier distribution method is definitely mail. The Notes client automatically uses public key encryption to protect your secret key when it is mailed, and it gives the recipient the ability to add the key to his ID file with a single click. If you use the export method instead, you will get a KEY file, which you can put on a floppy and hand to a co-workers, who will then have to use the Import Key button in the User ID dialog box on their own computers.

There are two reasons why you might want or need to consider using the export feature and having your co-workers do a manual import of the key. First, only Notes mail users can receive keys via e-mail, and some of your application's users may not use Notes mail. Second, if you don't have personal knowledge that your system administrators follow strict procedures that assure that nobody (including themselves) ever had access to your co-workers ID file and password, then you shouldn't trust that the mail system is secure enough for something as potentially sensitive as a secret key.

Note that one of the biggest advantages of public key encryption is that public keys are already distributed automatically to all Notes users when their ID files are created. Under normal circumstances, there is never a need to create a new one, so the problem of distributing secret keys to application users is moot.

Deploying the application

Once your keys are distributed, use File - Database - New to create a new database file using the Encrypted Doc Library Template you have just built. (Alternatively, you can create a database using the sample encryption template, which you can download from the **Iris Sandbox.**) Open the new database, and choose Set Up Encryption Profile from the Actions menu.



Encryption Profile

Encryption support

☒ Secret Key
☒ Public Key

Check one or both boxes to enable encryption. Secret key encryption uses keys stored in user id files. Public key encryption uses keys stored in person documents in the Domino Directory or your personal address book.

Secret Keys

RHS Private

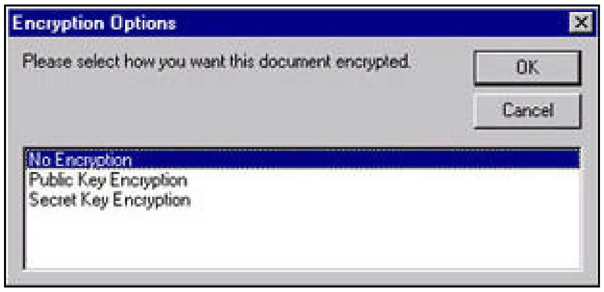
Enter the names of keys that are stored in user id files. Users who do not have these keys will not be able to read documents that are stored using secret key encryption.

If you have one or more secret keys that you want to use, put a check in the Secret Key box and enter the names of the keys you want application users to choose from. If you also want users to be able to use public key encryption, put a check in the Public Key box. Then save the document.

The only other things you will need to do are set the Access Control List to give your co-workers access to the database and replicate the database onto one of your organization's servers.

Using the application

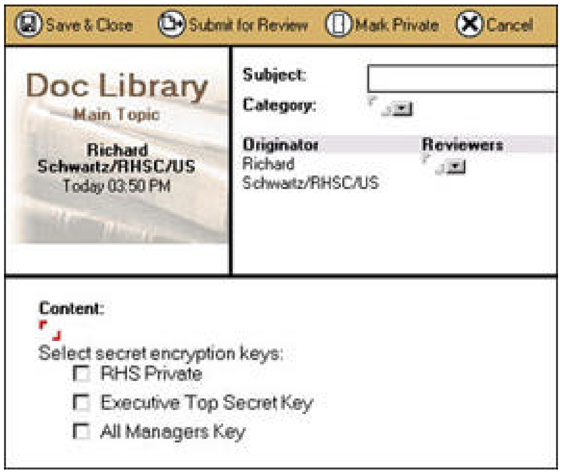
If you are already familiar with applications built using the standard Document Library template, then there's very little else to know. Whenever you create a Document, a Response, or a Response to Response document, you will be prompted to select the type of encryption you want to use.



Please select how you want this document encrypted.

No Encryption
Public Key Encryption
Secret Key Encryption

If you select secret key encryption, you will see a series of checkboxes in the document. Check the one that corresponds to the key that you want to use. The selected key will be used to encrypt the Body field when the document is saved.



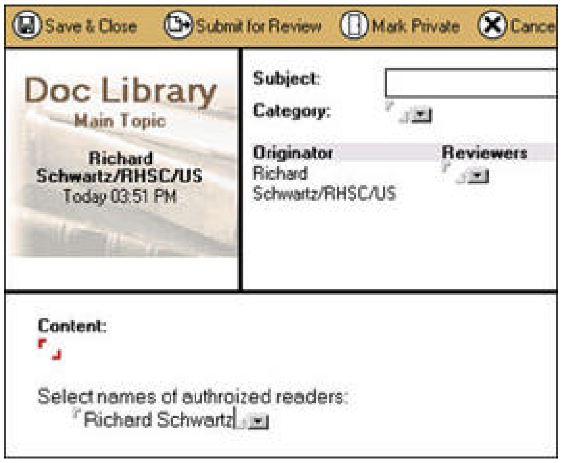
Richard Schwartz/RHSC/US
Today 03:50 PM

Subject:
Category:
Originator Richard Schwartz/RHSC/US
Reviewers

Content:

Select secret encryption keys:
☐ RHS Private
☐ Executive Top Secret Key
☐ All Managers Key

If you select public key encryption, you will see a name selection field with your own name filled in. The helper button next to the field will bring up the standard Notes addressing dialog box if you want to use it to select names from the Domino Directory or your personal address book. Add the names of other users to this field. These users' public keys will be used to encrypt the Body field of the document when it is saved.



Richard Schwartz/RHSC/US
Today 03:51 PM

Subject:
Category:
Originator Richard Schwartz/RHSC/US
Reviewers

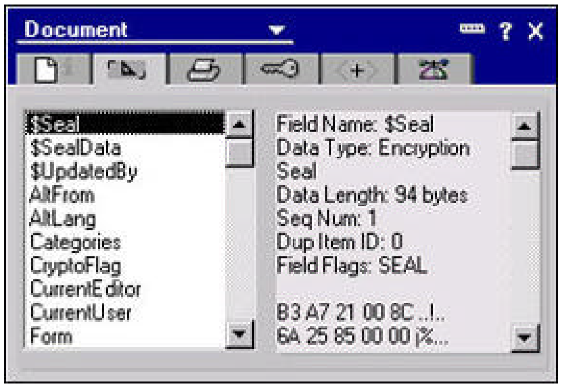
Content:

Select names of authroized readers:
Richard Schwartz

If you have the appropriate secret key for a document, or if your name has been listed in a document, you can open a saved document for editing and select new or additional keys or names and re-save the document. Notes will automatically re-encrypt the Body field when the document is saved.

The Body field is the only field encrypted in this template. It is especially important to note that the Subject field is not encrypted. No field that appears in a view can be encrypted because the indexer task on the server would need to have access to the keys in order to build the views, and giving the server the keys would eliminate the security advantage of encryption.

Under the covers
Let's examine what goes on behind the scenes a little more closely. When you create a document using a form with encrypted fields, two hidden fields are added to the document. The fields are named $Seal and $SealData.



$Seal
$SealData
$UpdatedBy
AltFrom
AltLang
Categories
CryptoFlag
CurrentEditor
CurrentUser
Form

Field Name: $Seal
Data Type: Encryption Seal
Data Length: 94 bytes
Seq Num: 1
Dup Item ID: 0
Field Flags: SEAL

```
B3 A7 21 00 8C ..!..
6A 25 85 00 00 j%...
```

Whenever these fields are present in a document, Notes checks the special fields SecretEncryptionKeys and PublicEncryptionKeys to determine what type of encryption has been applied and what the keys are.

All encryption and decryption operations are automatically handled by the Notes API routines. Encryption operations are performed with the RSA and RC2 algorithms. When secret key encryption is used, the Notes API routines store the following items in the document:

- $SealData
  Contains field data encrypted with RC2 using a randomly generated, 64 bit key. The random key is known as the “bulk encryption key.”
- $Seal
  Contains the bulk encryption key, encrypted with RC2 using the 64 bit secret keys in the current ID file whose names match the SecretEncryptionKeys field.

A random key is used to encrypt the actual fields because the Notes API allows database designers to specify multiple keys in the SecretEncryptionKeys field. Since a document usually contains a lot more than 64 bits of data, encrypting the randomly generated key multiple times will almost always be more efficient than encrypting the actual data multiple times.

When public key encryption is used, the Notes API routines store the following items in the document:

- $SealData
  Contains field data encrypted with RC2 using a randomly generated, 64 bit, bulk encryption key.
- $Seal
  Contains the bulk encryption key, encrypted with RSA using the 630 bit public keys stored in the Domino Directory for the users whose names

match the PublicEncryptionKeys field.

For more information about the encryption techniques used in Notes, see the IBM Redbook ***Lotus Notes and Domino R5.0 Security Infrastructure Revealed***.

What about the Web?

Unfortunately, field encryption cannot work in Web applications. Encryption support is a Notes client feature. Secret keys are stored in Notes ID files. Although public keys are stored in the Domino Directory, the private keys that are used to decrypt data that is encrypted with a public key are also stored in Notes ID files. Browsers don't have the built-in support, and browser users don't have Notes ID files.

In theory, Java applets or ActiveX components that run in the browser could provide equivalent functionality, and a secure container for browser users' keys equivalent to the Notes ID files could be used by the Java or ActiveX code. In practice, this turns out to be more difficult than it seems it should be. Things like cross-browser compatibility issues, JVM compatibility issues, and platform security issues make it quite difficult, and no solution has been implemented as of yet.

Conclusion

There are a few things that you must keep in mind about encryption. First of all, for all intents and purposes, Domino's encryption is unbreakable, and that is both good and bad. It means that nobody can read your data unless they have the appropriate key in their ID file, but it also means that you can't read the data if *you* lose the key. Remember to back up your ID file any time you add a new secret key to it!

Also, bear in mind that encrypted fields can't be seen in views. This is an architectural restriction. Even if the indexer on the server had access to the appropriate keys, which it normally would not, there would be a significant performance impact if data were decrypted when sorting views on the server, or when displaying them on the client. This is the one significant design trade-off that you must make when using encryption.

When you really need to protect your data, encryption is a very effective security tool. Domino Designer makes encryption easy to implement; no other development tool makes it as easy. You can easily extend the sample code described here to work with any of your applications.

About the author

Richard Schwartz is founder of RHS Consulting Incorporated, a Lotus Business Partner that does Notes and Domino work for many large corporations, including Lotus and Iris. Rich is the author of several of the templates and samples that shipped with Domino 4.6. He is a Certified Lotus Professional and has been a frequent contributor to various Notes and Domino-related publications. He is also a founder of Penumbra Group, a Lotus Premium Business Partner; and his activity in various Notes-related forums on the Internet led to his being honored with a Lotus Business Partners' Beacon Award in January, 1995. A key contributor to e-mail, directory, and other network application product development at Wang Laboratories from 1983 to 1991, Rich has more than 18 years of hands-on experience with groupware technology.