

Appendix A Encryption times for spatial data stored within DFS

Table 1 shows encryption (Enc.) and decryption (Dec.) times of different sizes of spatial files using 256-bit AES encryption (16-character key) using three approaches:

- within ECMSDK using DBMS_CRYPTO_TOOLKIT,
- within ECMSDK using OpenSSL encryption tool, and
- within Windows NTFS using OpenSSL encryption tool.

The first two approaches compares the encryption and decryption performance of files within ECMSDK using the DBMS_CRYPTO_TOOLKIT (implemented in PL/SQL) against the performance provided by the OpenSSL tool (a Windows executable, called externally from our encryption scripts). The third approach compares the encryption of spatial files within ECMSDK versus those stored directly on the OS (i.e. Windows NTFS). The ESRI shapefiles used in the investigation are sourced from publicly available datasets (<https://mapcruzin.com/download-free-arcgis-shapefiles.htm>).

In all experiments the tests were performed on a machine with the following specifications: an Intel 1.6 GHz i5 processor with 8 GB RAM on Dell Inspiron system and the software utilized consisted of Oracle 11g database Release 2, running on Windows 10 Operating System.

GIS File	Filesize (Mb)	AES		OpenSSL on CMSDK		OpenSSL on NTFS	
		Enc.	Dec.	Enc.	Dec.	Enc.	Dec.
france-points.shp	1.50	1.13	1.07	0.48	0.17	0.04	0.04
france-waterways.shp	6.32	4.73	4.49	0.55	0.29	0.06	0.05
france-natural.shp	11.67	8.74	8.30	1.15	0.45	0.11	0.07
indonesia-natural.shp	11.95	8.95	8.50	1.20	0.55	0.07	0.07
germany-points.shp	13.89	10.41	9.88	1.25	0.92	0.07	0.14
britain-waterways.shp	16.12	12.08	11.46	1.93	1.31	0.08	0.09
china-buildings.shp	26.53	19.87	18.87	2.25	1.90	0.12	0.34
india-natural.shp	32.13	24.07	22.85	3.81	2.38	0.13	0.37
china-natural.shp	33.75	25.28	24.00	4.08	2.68	0.14	0.48
india-waterways.shp	36.21	27.13	25.75	4.18	2.77	0.15	0.56

Table 1: AES, 3DES, and DES encryption time for spatial files within Internet File System.

Appendix B Oracle CMSDK session-based trigger

This section outlines the DFS user-session based trigger implemented within Oracle CMSDK. On each CMSDK user session creation and termination, the session-based trigger is used to call decryption and encryption procedure respectively. Those two procedures first obtain the single-user as well as the multi-user GIS files to which user has privilege, and then calls either the decryption or the encryption procedure so that these files can be decrypted or encrypted. To prevent the execution of this trigger when multiple users are using the same account, which would corrupt the file resulting from double encryption and decryption, the trigger first queries the number of existing sessions of the user; only if no existing session of the user exists, then the files are decrypted or encrypted.

```
CREATE OR REPLACE TRIGGER dec_enc_spatial_files

BEFORE INSERT OR DELETE ON ifssys.odmz_session
FOR EACH ROW
DECLARE
    v_sessions NUMBER;
    CURSOR active_CMSDK_user_sessions IS
        SELECT USERID FROM IFSSYS.ODMZ_SESSION
        WHERE USERID = (:NEW.USERID) OR USERID = (:OLD.USERID);
BEGIN
    OPEN active_CMSDK_user_sessions;
    FETCH active_CMSDK_user_sessions INTO v_sessions;
    --      Only if no existing sessions of this user, then decrypt or encrypt
    IF active_CMSDK_user_sessions%NOTFOUND THEN
        IF INSERTING THEN
            CMSDK_spatial_crypt.get_10gcmsdk_user_files_dec (:NEW.USERID);
        ELSIF DELETING THEN
            CMSDK_spatial_crypt.get_10gcmsdk_user_files_enc (:OLD.USERID);
        END IF;
    END IF;
    CLOSE active_CMSDK_user_sessions;
END dec_enc_spatial_files;
/
```

Appendix C Proposed security solution demonstration for use with classical shared folders

We illustrate the user-interaction model with the proposed storage security solution in a classical GIS setup where users log onto a central server by using shared folders to use single-user and multi-user spatial files. Two GIS users, **Scott and Alan**, require access to their home folders and to these GIS files that must be decrypted for use and encrypted back for protection in storage.

Single-user and Multi-user files and access: The single-user files are in the user's home folder. There are eight multi-user files; Rivers.DAT, Rivers.MAP, Rivers.ID, Rivers.TAB, Roads.DAT, Roads.MAP, Roads.ID, and Roads.TAB placed in the "O:\projects\classified\Fiji\" DFS directory.

User's privileges: Scott has the privilege to two single-user files: Places.MAP and OilLocation.doc placed in his home folder) and four multi-user GIS files: the four Rivers layer files. These files require protection with encryption. GIS user Alan, on the other hand, should have access to all eight multi-user GIS files. For brevity reasons, the user does not have any single-user files.

Multi-user files	Access	
	Scott	Alan
Rivers.DAT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Rivers.MAP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Rivers.ID	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Rivers.TAB	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Roads.DAT		<input checked="" type="checkbox"/>
Roads.MAP		<input checked="" type="checkbox"/>
Roads.ID		<input checked="" type="checkbox"/>
Roads.TAB		<input checked="" type="checkbox"/>

Table 2. Access to multi-user files placed in "O:\projects\classified\Fiji\" directory

Each user must not have access to the other user's home folder.

Protocol services: For brevity, we only demonstrate the security model using the network folder sharing using the Oracle CMSDK SMB server. In GIS, this is the most common mode of accessing spatial data; clients access files on the server through mapped network shares and interact with files through GIS applications at the client computer. However, working with the security solution using these other protocol servers (e.g. FTP, HTTP, Command line, etc.) provided by Oracle CMSDK DFS product will be similar, since the DFS enforces access control at the repository level and the execution of the encryption-decryption procedures depend on DFS user sessions, that are created when the user uses any protocol service.

To outline the essential information only, we assume that the single-user and multi-user cryptographic keys for users Scott and Alan have already been created and initially Scott's single-user files and multi-user files are already in encrypted form. We assume that the administrator has encrypted all eight multi-user files and user Scott has already specified that the **Places.MAP** and **OilLocations.doc** need encryption by renaming them with an additional .enc file extension; resulting in **Places.MAP.enc** and **OilLocations.doc.enc**. In addition, we assume that he has subsequently logged out of DFS upon which the files have been encrypted (see Fig. 1 and 2).

The following sequence carries on from that point illustrating GIS users Scott and Alan using the proposed security solution where their single-user and multi-user GIS files are decrypted for use, are available for multiple users during their DFS sessions, and are later encrypted back for protection.

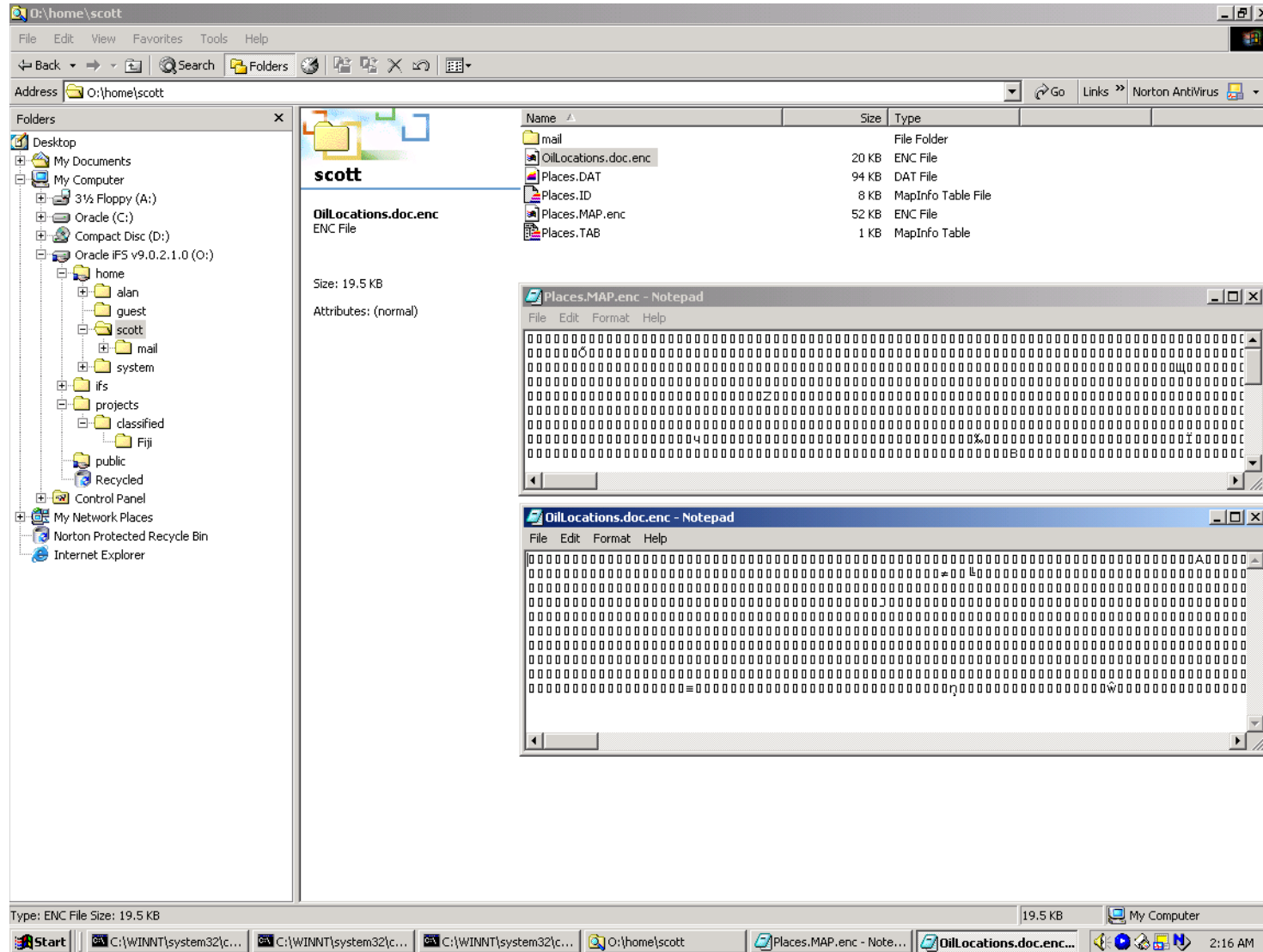


Figure 1: GIS user Scott's single-user GIS files are originally in encrypted form on the DFS server.

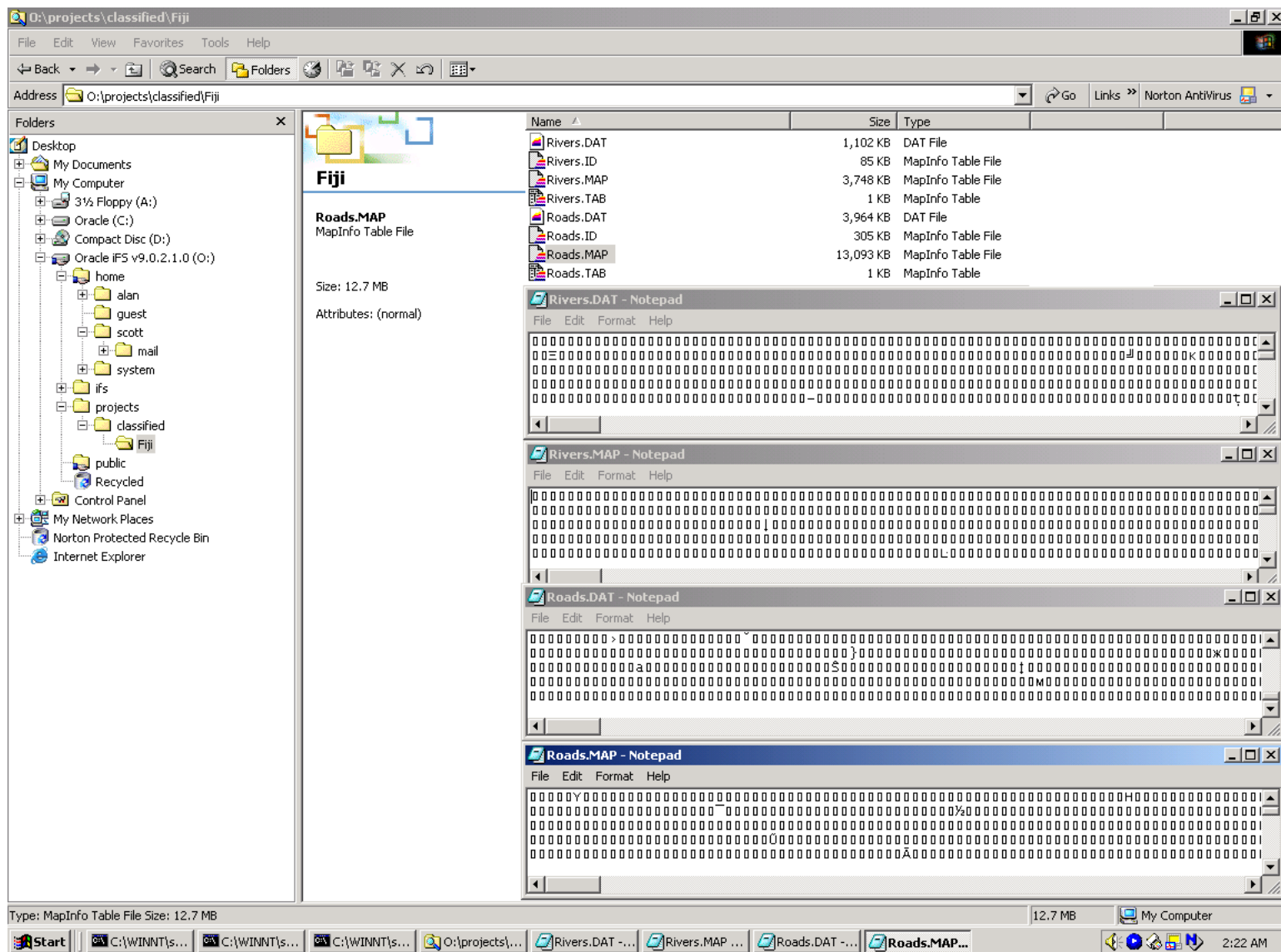


Figure 2: GIS user Scott's multi-user GIS files also are originally in encrypted form on the DFS server.

Illustrating proposed security solution using network folder sharing and GIS application

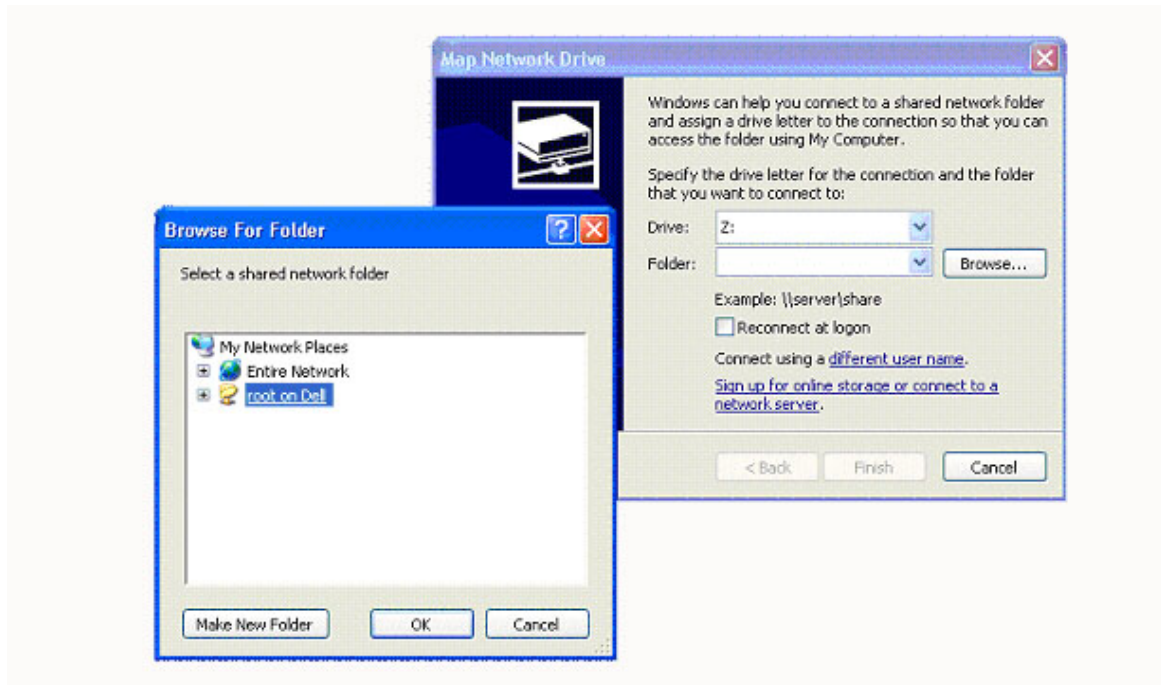


Figure 3: GIS user Scott chooses DFS shared resource to map a network share.

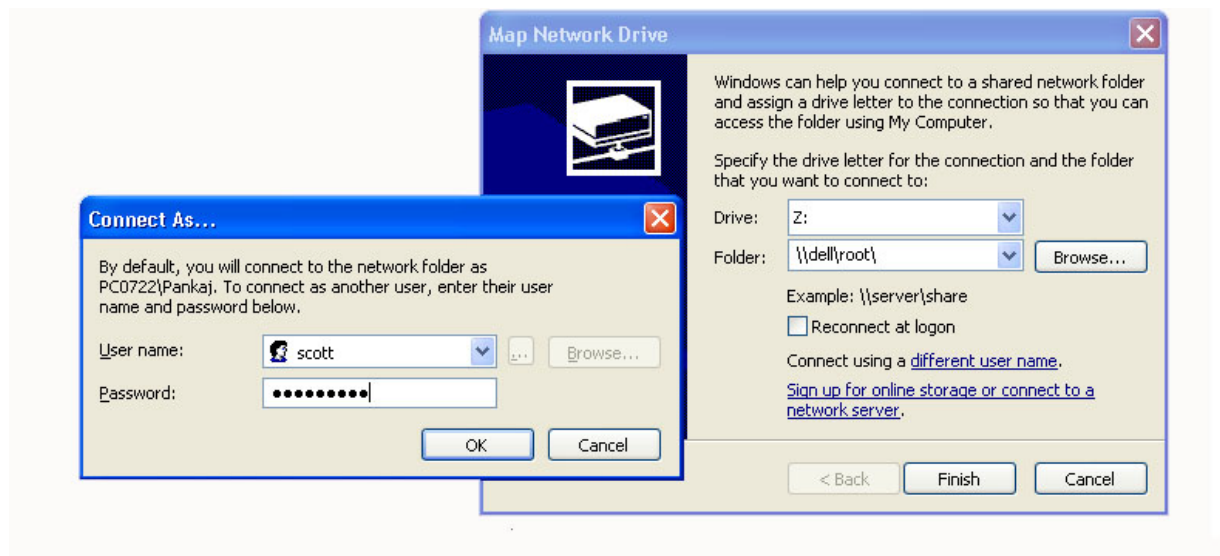


Figure 4: Spatial user “Scott” provides credentials for authentication to the DFS protocol server; in this case, it is the Oracle CMSDK SMB server. Scott is authorized based on the validity of his credentials.

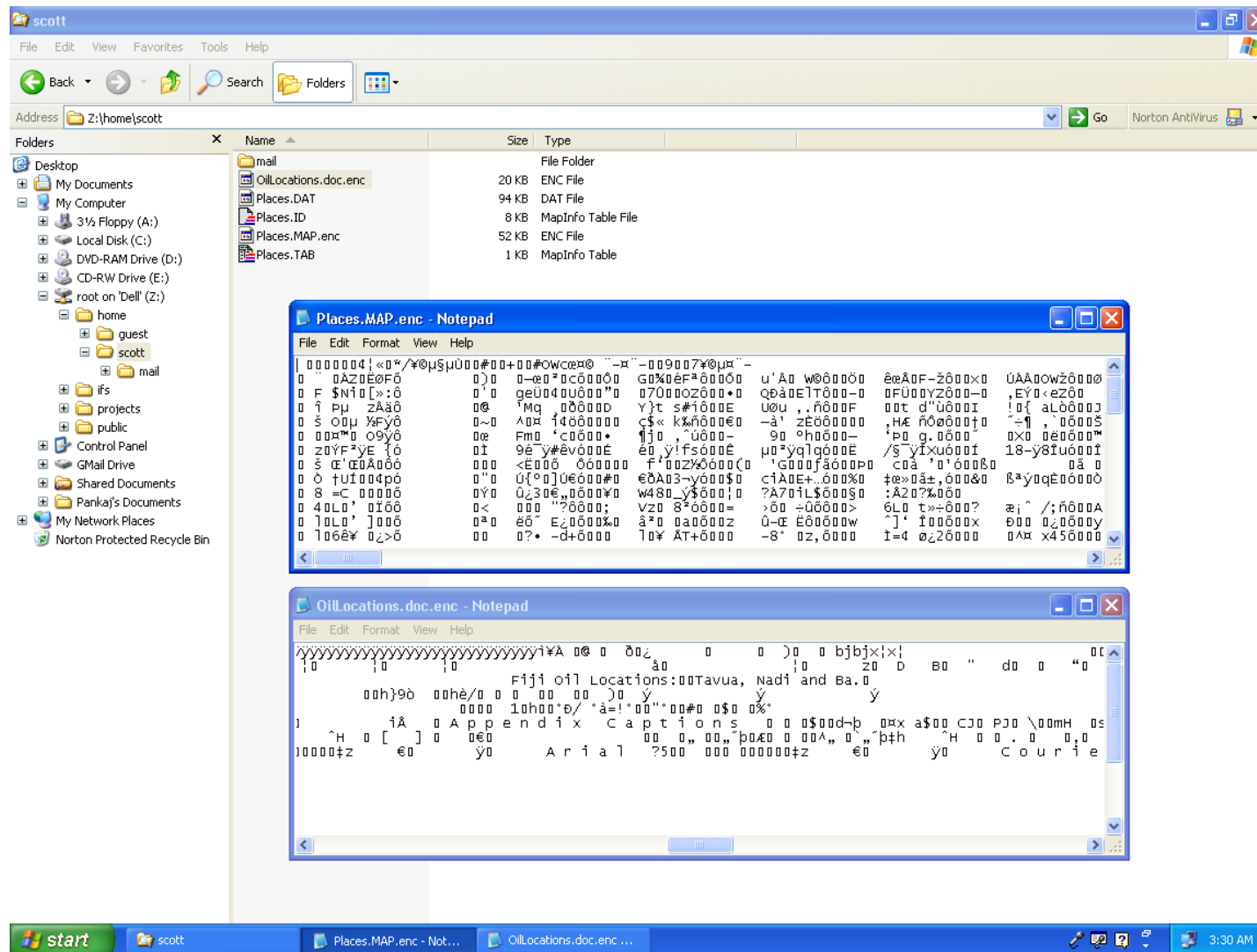


Figure 5: Once user Scott logs in to the DFS, and after making session, as to list folder contents using Windows Explorer, the DFS user session-based trigger is fired and files to which user Scott needs access are decrypted for use. This includes his single-user files (shown decrypted in Figure 5) as well as multi-user GIS files to which Scott has privilege (shown decrypted in Figure 6).

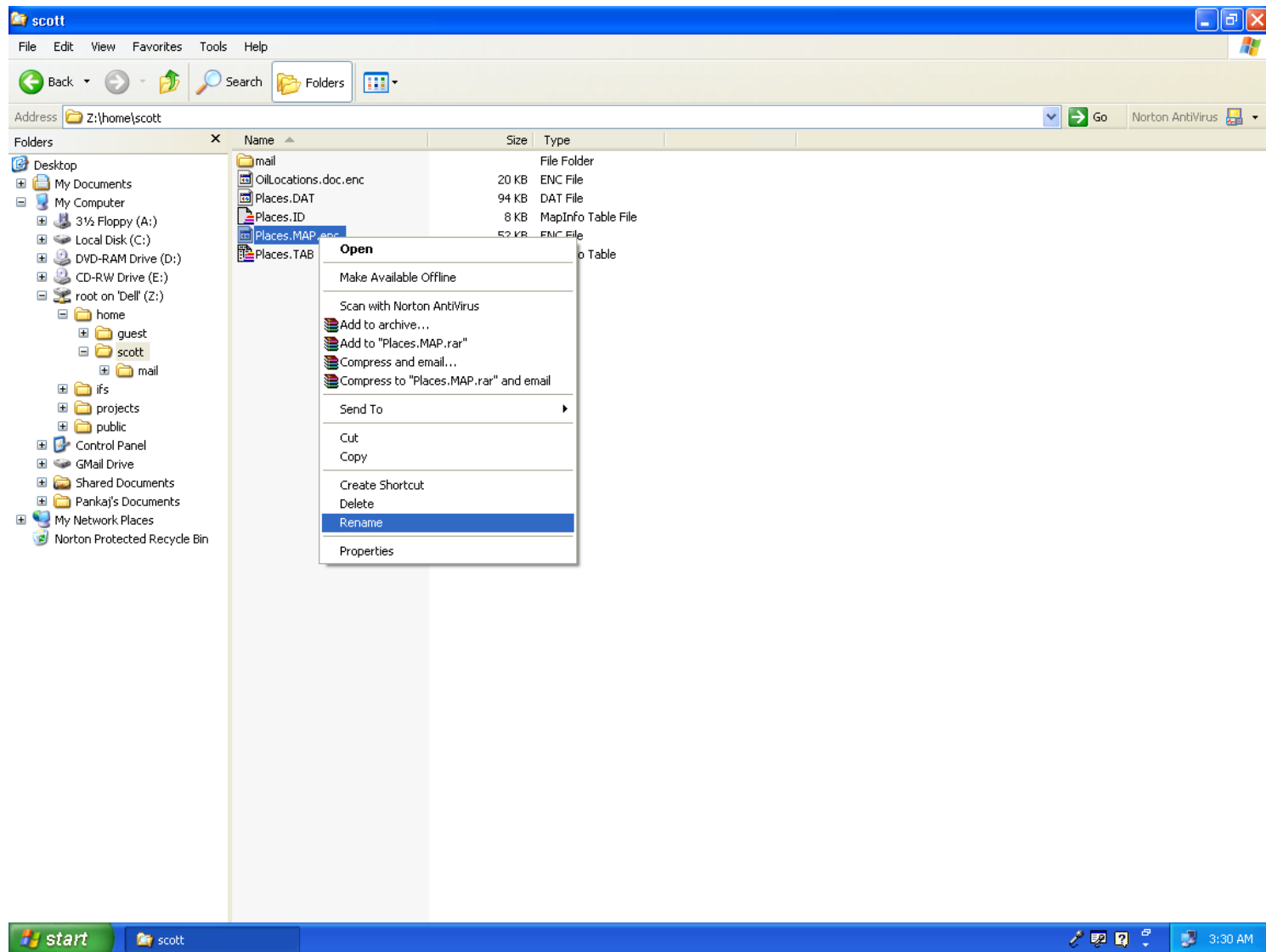


Figure 7: Once Scott's single-user and the multi-user files to which he has access, have been decrypted, he can access and work with the files. The illustration shows user Scott renaming the Places.MAP.enc filename back to Places.MAP for use.

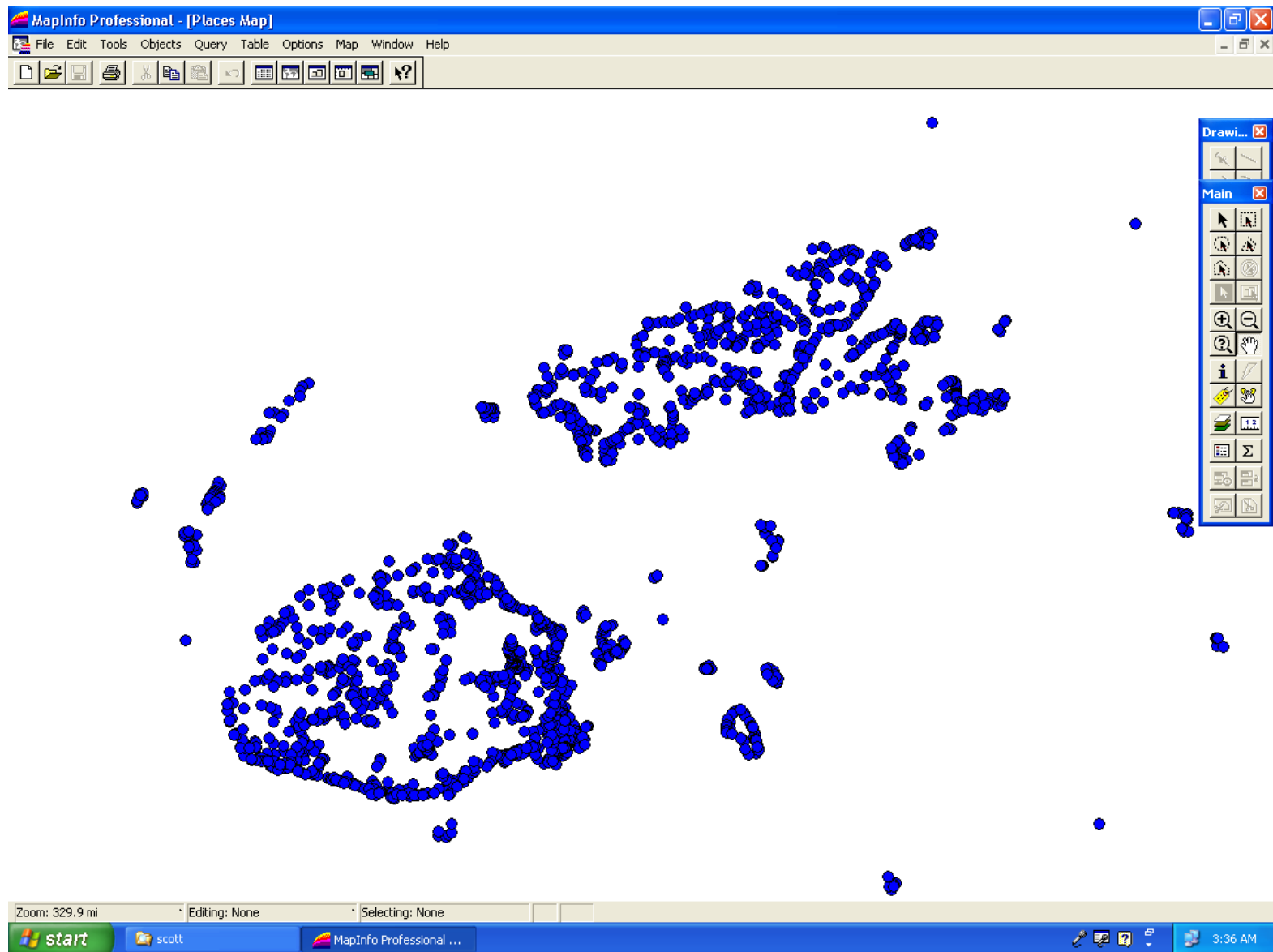


Figure 8: Scott can now access and work with the Places.MAP file.

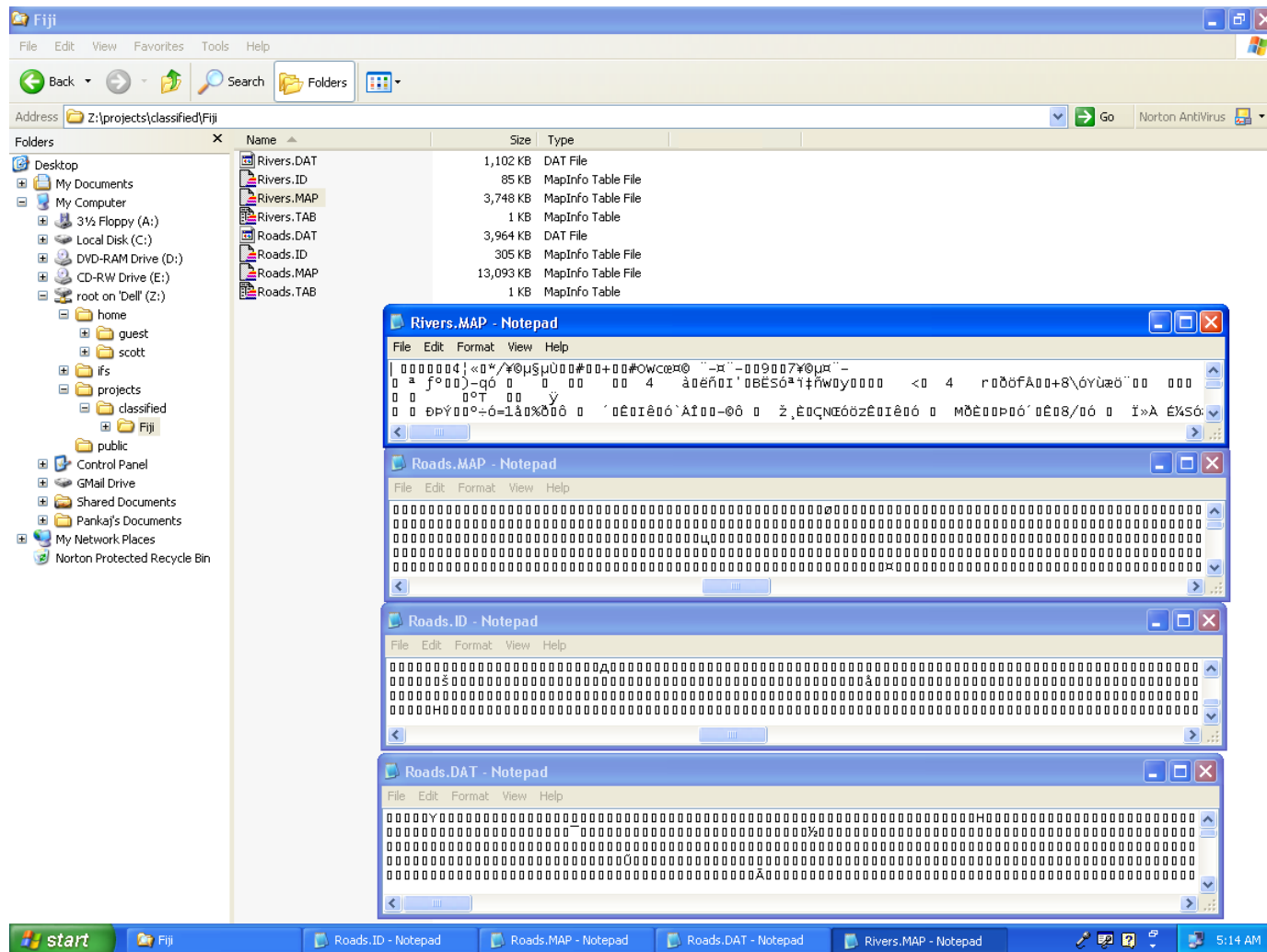


Figure 10: User Alan's logout encrypts only those multi-user GIS files not in use. When user Alan logs off, the multi-user GIS files are encrypted back for protection. However, files which are currently being used by privileged users, are not encrypted, based on active DFS user sessions. Since, Scott has active sessions in the DFS, which would almost certainly indicate is he is still using the 4 multi-user files, these 4 files are left in decrypted form so that Scott can continue using them.

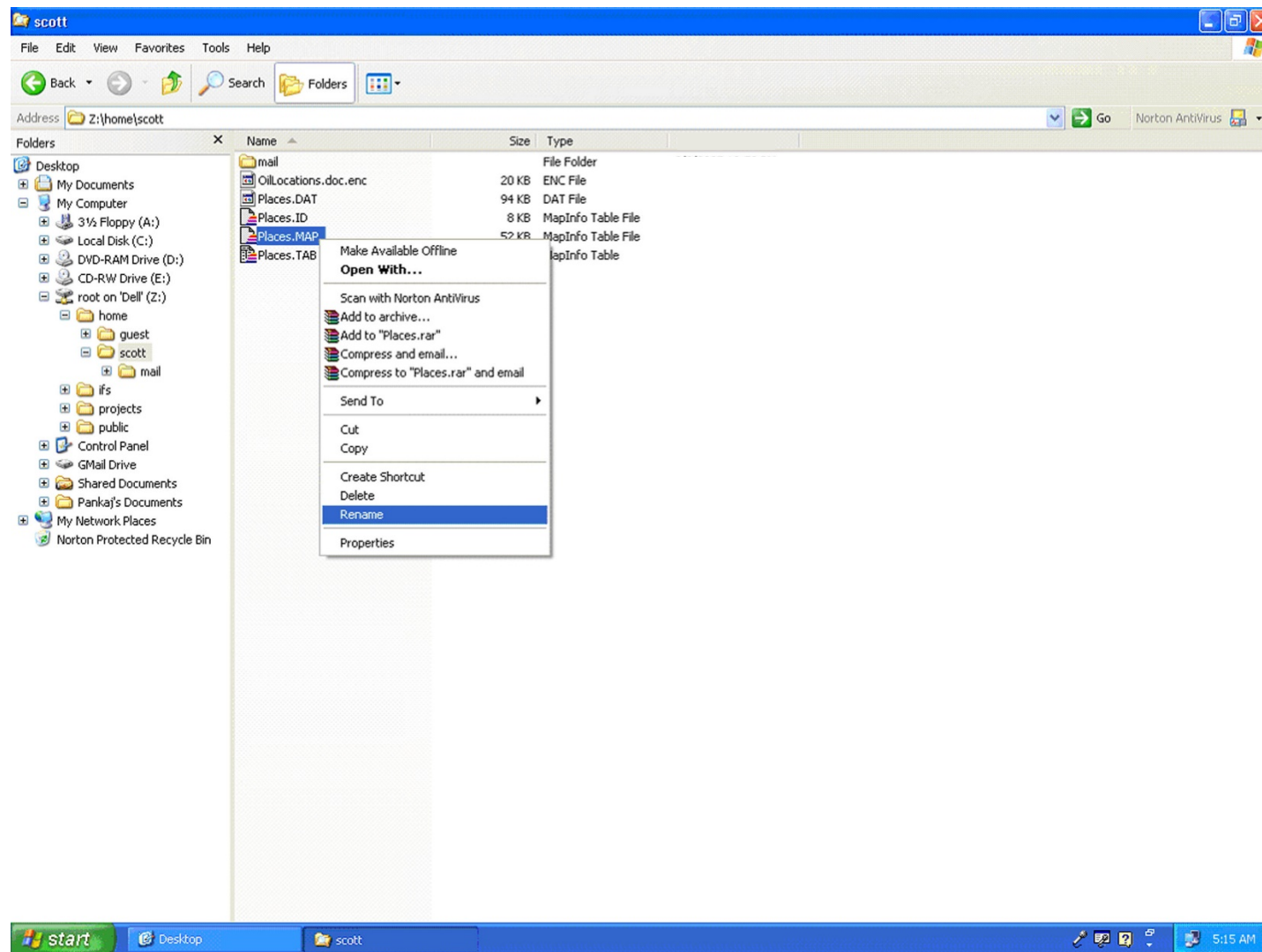


Figure 11: After using the files, user Scott can protect the GIS file during storage with an additional layer of encryption by adding an .enc file extension. This can be added to any file with any ACL but it may be suitable to apply private ACLs as it would prevent access from other standard users. In Figure 12 shows user Scott renaming the Places.map file.

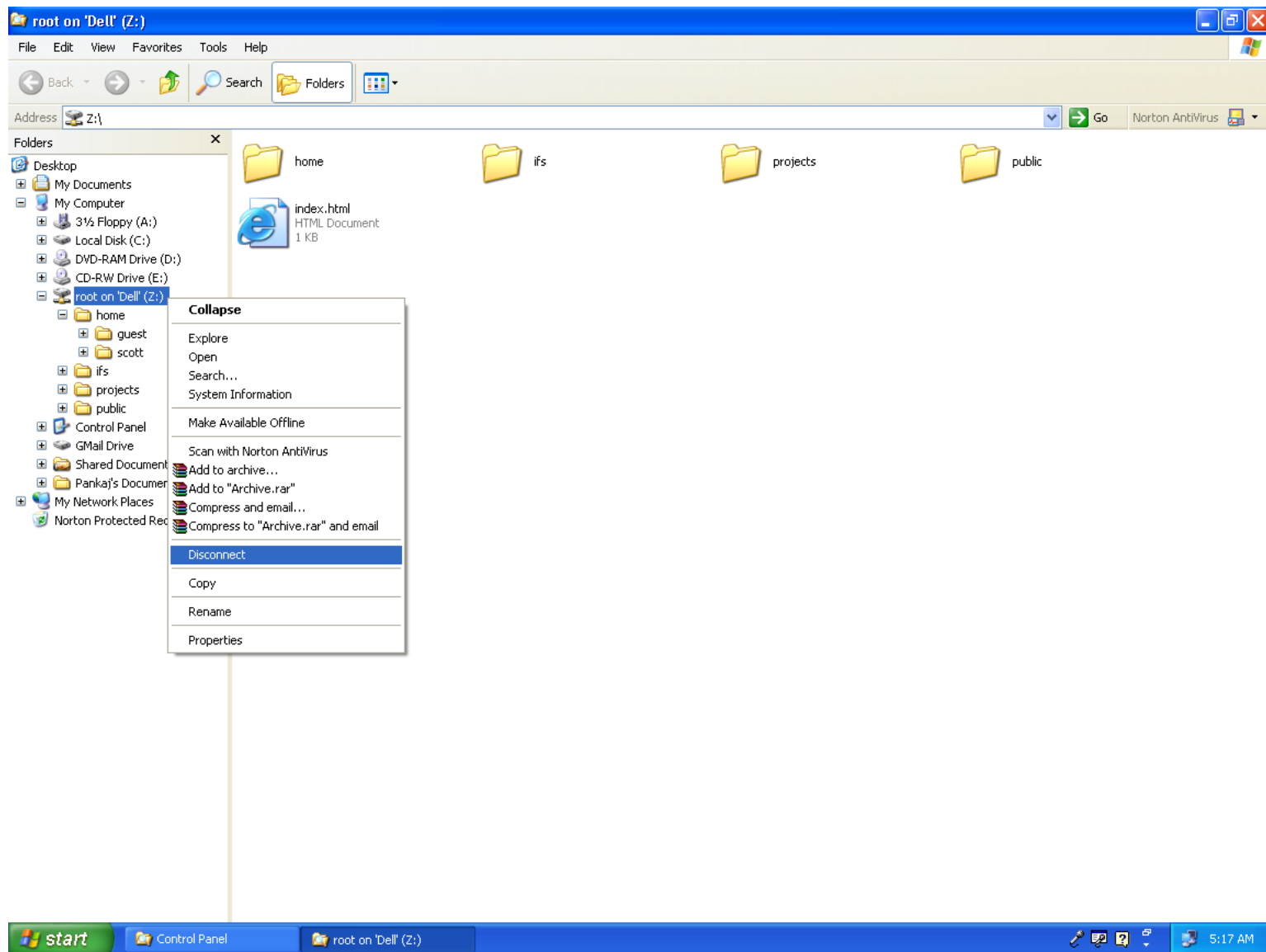


Figure 12: User Scott logs out.

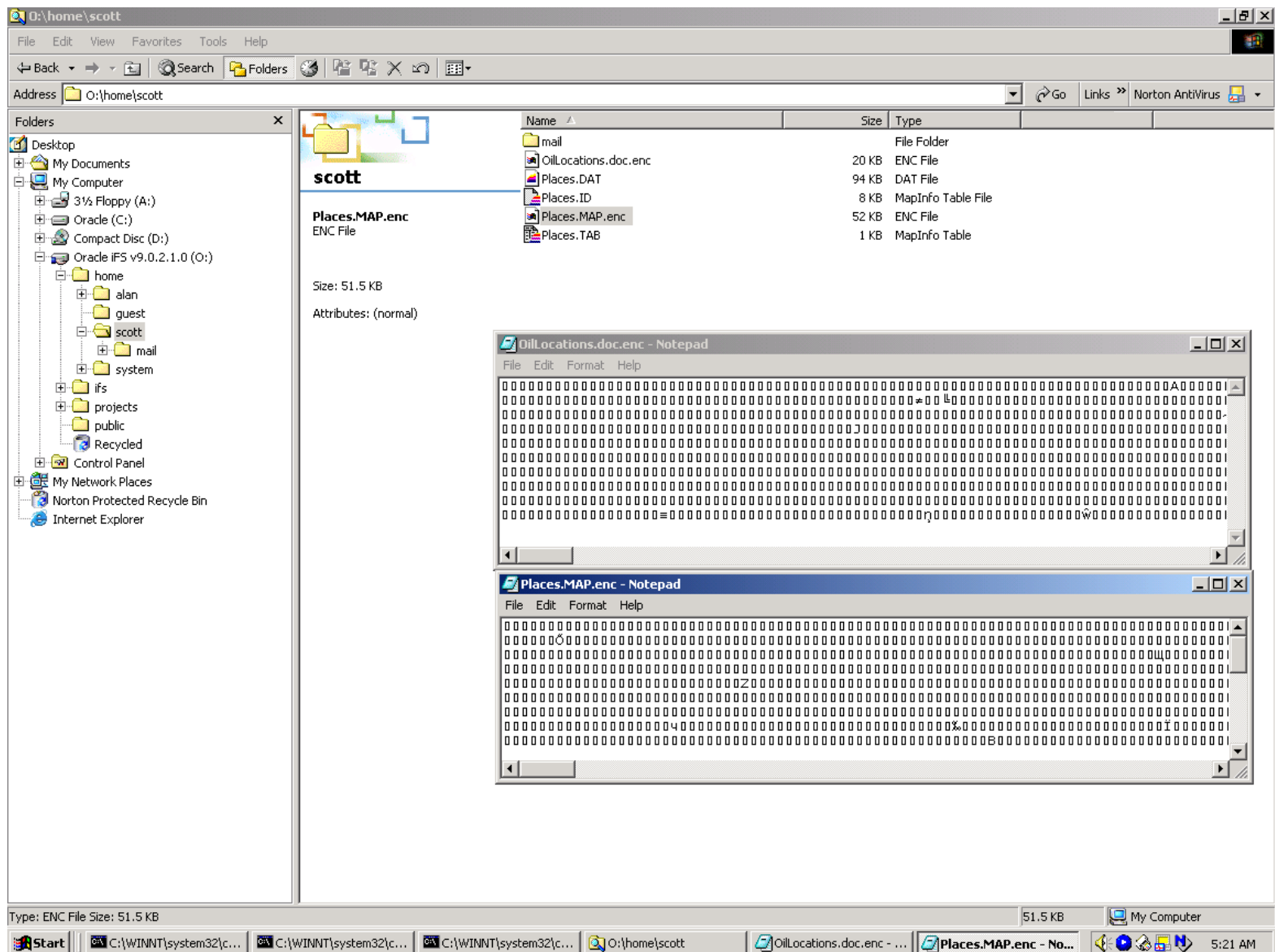


Figure 13: On DFS server, Scott's single-user GIS files are encrypted back for protection.

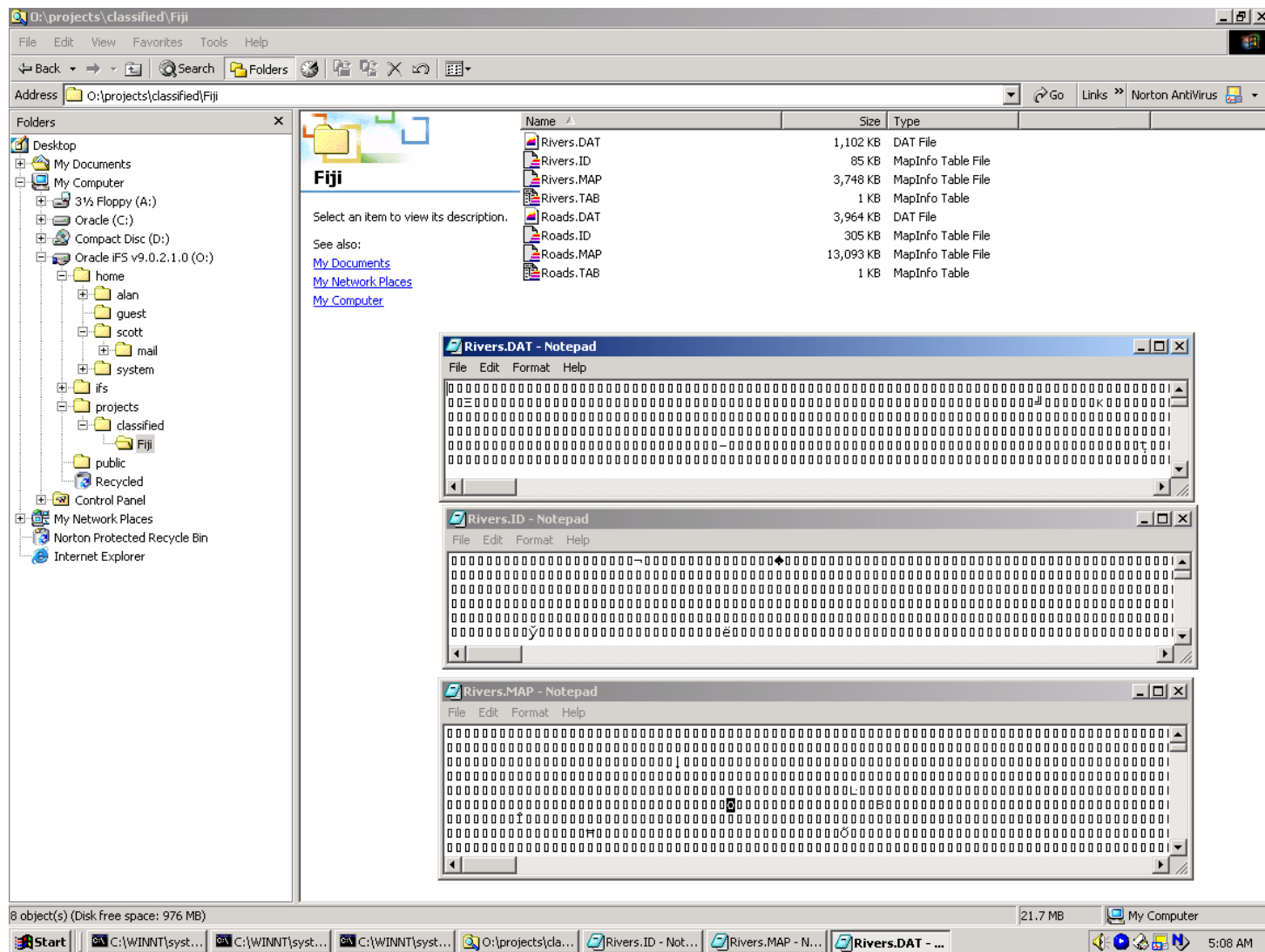


Figure 14: On DFS server, Scott's multi-user GIS files are also encrypted back for protection. The multi-user files are encrypted back only when the last user who requires access logs out. In this case Scott was the last user to log out, who had access to these files.

Appendix D Protection for multi-user files

This section demonstrates how the security solution protects the corruption of multi-user spatial files during existing active sessions of another privileged user. Two figures demonstrate how the multi-user GIS files are prevented from double decryption when they have already been decrypted for use for a privileged GIS user as well as preventing encryption of these files when another privileged user is currently working on them, which would make it unusable. For demonstration purposes of the inner-workings, SQL commands are used, while in practice these would be implemented to execute automatically as a set of DFS session-based triggers.

The figure captions are used for explanation of the processes.

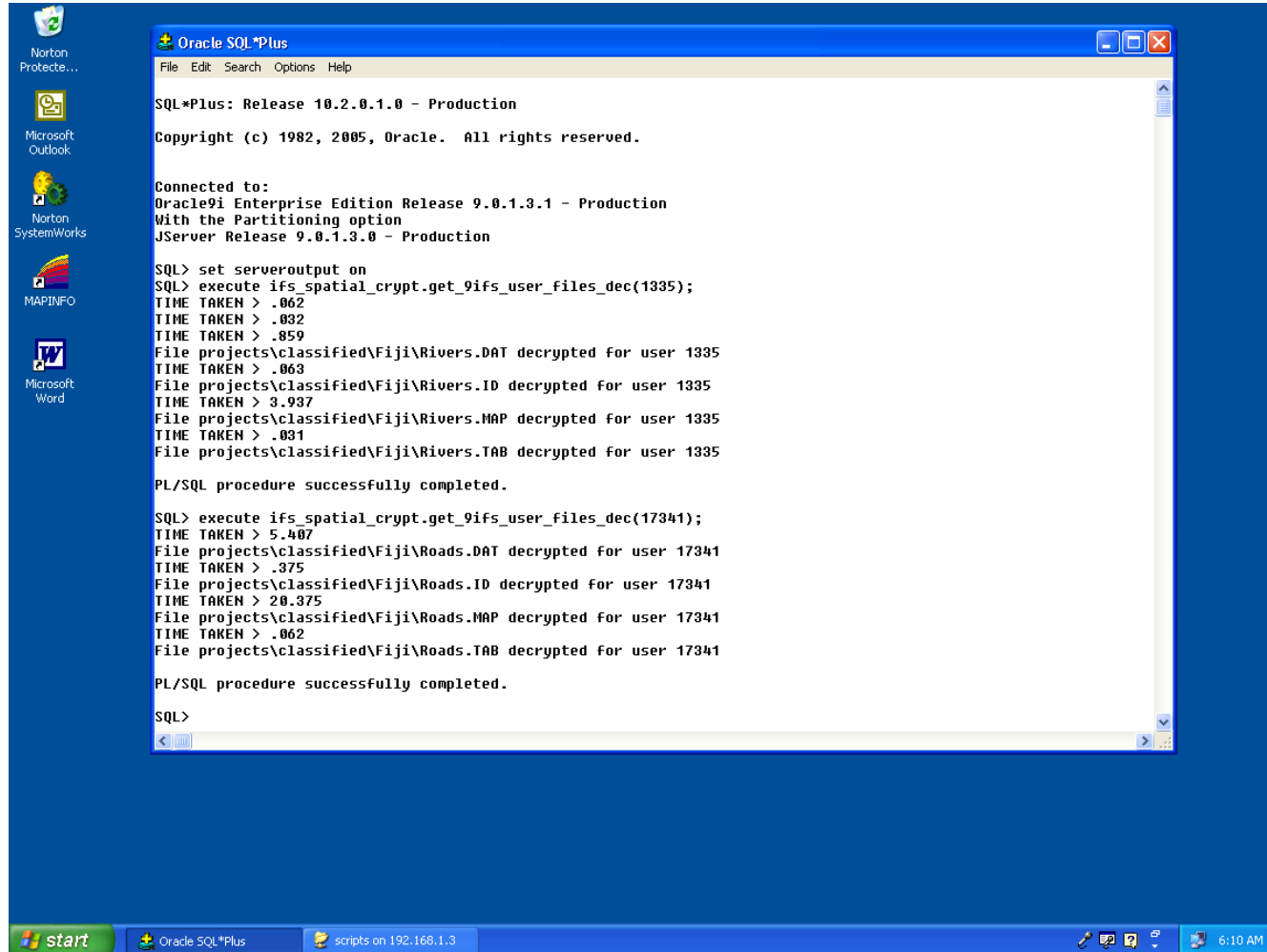
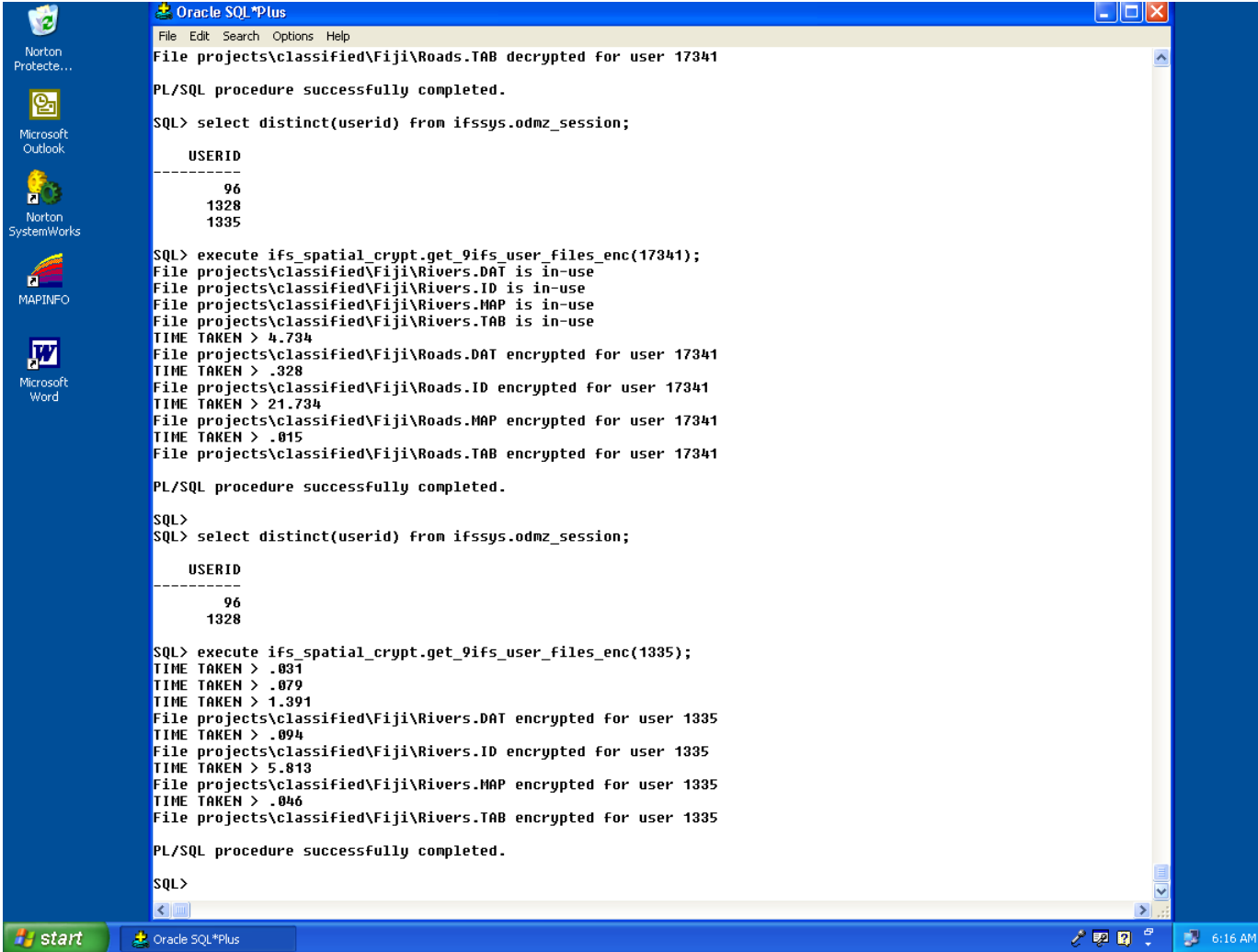


Figure 15: Multi-user GIS file decryption for two GIS users. The Figure shows prevention of these set of files against double-decryption. User Scott (Oracle CMSDK user id 1335) login would decrypt his two single-user files as well as four of the eight multi-user GIS files. Subsequently on user Alan (Oracle CMSDK user id 17341) login; since four of these eight files have already been decrypted; only the four remaining have been decrypted.



```

Oracle SQL*Plus
File Edit Search Options Help
File projects\classified\Fiji\Roads.TAB decrypted for user 17341
PL/SQL procedure successfully completed.
SQL> select distinct(userid) from ifssys.odmz_session;
-----
      USERID
-----
          96
         1328
         1335

SQL> execute ifs_spatial_crypt.get_9ifs_user_files_enc(17341);
File projects\classified\Fiji\Rivers.DAT is in-use
File projects\classified\Fiji\Rivers.ID is in-use
File projects\classified\Fiji\Rivers.MAP is in-use
File projects\classified\Fiji\Rivers.TAB is in-use
TIME TAKEN > 4.734
File projects\classified\Fiji\Roads.DAT encrypted for user 17341
TIME TAKEN > .328
File projects\classified\Fiji\Roads.ID encrypted for user 17341
TIME TAKEN > 21.734
File projects\classified\Fiji\Roads.MAP encrypted for user 17341
TIME TAKEN > .015
File projects\classified\Fiji\Roads.TAB encrypted for user 17341
PL/SQL procedure successfully completed.
SQL>
SQL> select distinct(userid) from ifssys.odmz_session;
-----
      USERID
-----
          96
         1328

SQL> execute ifs_spatial_crypt.get_9ifs_user_files_enc(1335);
TIME TAKEN > .031
TIME TAKEN > .079
TIME TAKEN > 1.391
File projects\classified\Fiji\Rivers.DAT encrypted for user 1335
TIME TAKEN > .094
File projects\classified\Fiji\Rivers.ID encrypted for user 1335
TIME TAKEN > 5.813
File projects\classified\Fiji\Rivers.MAP encrypted for user 1335
TIME TAKEN > .046
File projects\classified\Fiji\Rivers.TAB encrypted for user 1335
PL/SQL procedure successfully completed.
SQL>

```

Figure 16: Multi-user GIS file encryption for two GIS users. The figure shows that while user Alan, who has privilege to all eight multi-user GIS files, logs out; only four of these files are encrypted back as user Scott still has active user session in DFS, allowing Scott to continue using them. When user Scott logs out, and therefore no more remaining privileged users on these four files, they are encrypted back.

Appendix E Security solution implemented within database-driven GIS Web portals & Web maps

This section describes the implementation of the encryption solution for spatial files within database-centric Web Portal. When GIS users first access the Portal, they see only the public pages and content. In order to see the protected GIS content, they must be authenticated as one of the authorized Portal user or administrator. Furthermore, the encryption model for files stored in Oracle Portal is similar to our encryption solution for DFS using CMSDK, with slight modifications to the procedures, since both products store file content and metadata within the database.

E.1 Single-user and multi-user spatial file encryption schemes

In contrast to CMSDK, Oracle Portal stores the document metadata and content in the same table; in the **WWDOC_DOCUMENT\$**. Thus, information on user-owned sensitive spatial files were retrieved from the same table. The only distinction to the above procedures was the Portal schema, table and column names in the cursor definition statement. Code segment attached below generates the list of Portal single-user files that have an *.enc* file extension from the Portal schema tables. The ids of these files can then be passed to the encryption and decryption script one by one for encryption and decryption respectively.

CURSOR portal_user_files IS

```
SELECT name, id FROM PORTAL30.WWDOC_DOCUMENT$ WHERE creator = v_userid
AND (name LIKE '%.enc');
```

For sensitive files requiring project level multi-user access, the encryption scheme prepared for CMSDK users can be developed with slight modification to the content and metadata tables.

E.2 Automated decryption and encryption

Implementation of an automated decryption and encryption model for sensitive files in Portal can be based on Portal user session table **WWCTX_SSO_SESSION**\$ within the **PORTAL30** schema table which call the decryption and encryption procedures passing the Portal **user id** from the ID column

However, these sensitive files can be vulnerable when Portal users do not log out properly by shutting down the browser instead of clicking the logout link. In this case, the Portal user session will still be active, and the decrypted files would not be encrypted back for protection. To resolve this, a session cleanup procedure provided by Oracle 9i Portal can be used, namely **ctxjsub**, which can be run periodically as a DBMS job to perform session cleanup for the Portal session table **WWCTX_SSO_SESSION\$**.

This would delete all inactive sessions, and subsequently the triggers will be fired which would call the encryption procedures to encrypt files back for protection.

An absolute security approach for the GIS Portal could require encryption of Web Mapping files as well. The following section describes how the Web maps personal geodatabase flat files can be encrypted within in Portal as well as.

E.3 Encryption of WebMap geodatabase files

The Web Map applications (personal geodatabase flat files) use can be stored within DFS, such as Oracle CMSDK for further protection. This would allow added protection within web map with folder and file ACL security, as well as protections of storage media compromise with encryption while files would be decrypted only for use.

Concerning storage security for Interactive Web Atlas, the personal geodatabases can be protected using the project-level multi-user access encryption schemes. The end-users would login into the Portal once, and upon request to access Web Maps, the user credential would be authenticated against those for CMSDK that will be retrieved from Portal SSO servers and OID. Subsequent requests for Web maps would create CMSDK user sessions that would decrypt the files for use and encrypt back for protection when the user session(s) terminate with the logout link.

Figure 17 illustrates the Oracle Portal security model. When users first access the Portal, they only see the public pages and content. In order to see the protected GIS content, they must be authenticated as an authorized Portal user. The Portal first provides SSO page for users to enter credentials for authentication, which are encrypted in-transit. Upon successful authentication and subsequent Portal user session creation, project-specific and single-user Portal files are decrypted for use.

Thereafter, end-user actions on Portal are controlled through page, content area, folder and item level ACL security. These are enforced regardless of whether users access through the page interface or directly through the content area. Requests for Web Mapping services are directed to Portal external application provider that authenticates users against their accounts in the Web Map server on their behalf after retrieving them from the OID, providing single sign-on for Web Map external application. After user has finished working with the Portal and logging out successfully, the files are would be encrypted back for secure storage.

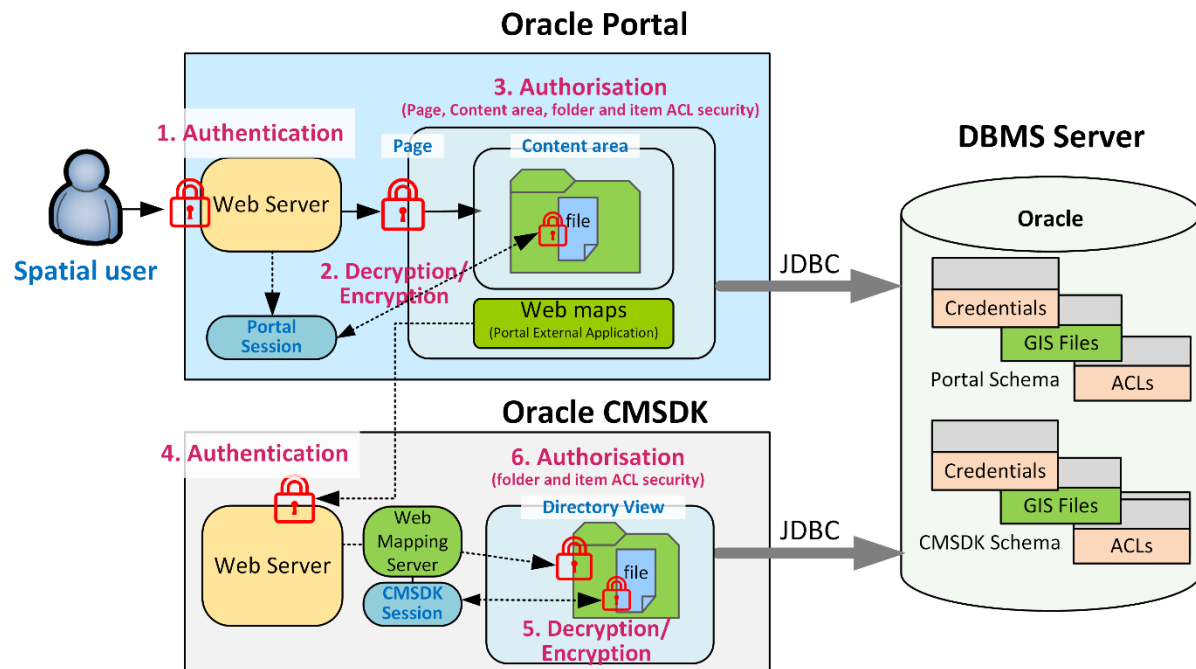


Figure 17: User interaction model for the proposed storage security solution implemented for spatial files within a database-centric web portal and for geodatabases in Web mapping service. Sensitive spatial files are automatically decrypted for use and encrypted back after use.

Security principles to authenticate and authorize users for accessing the protected GIS content through Oracle Portal are as follows:

1. The Portal HTTP Server displays the users with Single Sign-On (SSO) login page for submitting their SSO username and password; this will be the only set of credentials they will have to provide; which are protected in-transit using Portal SSL encryption.

2. The SSO Server verifies the credentials against those stored in OID. If the authentication is successful, the SSO Server creates a session cookie for the user. Information within this session cookie is later used to query the user's privileges specified in OID.
3. Only the authorized Portal users can now have access to the protected content within the Portal.
4. Portal user-session creation will fire the session-based trigger, calling the decryption procedure upon which the project-specific and single-user files will be decrypted for use.
5. Portal page ACL security controls access to pages; the first part of the Portal authorization process. From within these pages, requests for accessing protected GIS content, are directed to the associated category through the respective portlet.
6. All actions on content are controlled at multiple levels: Portal content area, folder, and item ACL security. Categories are used to display the list of items associated with that category at the same time preventing direct user access to folders and its content.
7. Folder and item ACL security is used to control permissions on submission folders and content.
8. User request for Web Map is passed to Portal external application provider.
9. The external application provider requests the Single Sign-On server (SSO) for user's credentials in the Web Map Server (WMS). SSO retrieves this from Oracle Internet Directory and returns it to the application provider.
10. The external application provider authenticates users against their accounts in the WMS on their behalf, providing single sign-on for Web Mapping services.
11. When Portal user logs out, the user-session termination fires the session-based trigger which calls the encryption procedure for encryption of the project-specific and single-user files back for secure storage.

Appendix F DFS encryption solution

This appendix includes the *DFS Encryption Solution*, implemented in Oracle PL/SQL. To implement it within Oracle it requires appropriate licences for the Oracle Database, Application Server, and Content Management Standard Development Kit (CM SDK). The code structure can be used to implement the encryption solution within other DFS products, such as IBM DB2 Content Manager.

The *DFS Encryption Solution* consists of three parts. First, the three database tables used to store the cryptographic keys for single-user as well as multi-user GIS files are described. This is followed by the CMSDK Spatial Crypt package and the CMSDK user-session based trigger is outlined.

F.1 Cryptographic key storage for DFS single-user and multi-user GIS files

This section includes the three database tables that are used to store encryption information that are used to perform the encryption and decryption of spatial files within the DFS Spatial Crypt package. This includes the catalogs implemented in database tables for: (1) encryption and decryption keys for spatial files belonging to specific users (user-specific), (2) files requiring access from multiple users (multi-user files), (3) and cryptographic keys together with the users who should have access to these multi-user files in decrypted form. These tables and the encrypted column definitions are only possible with the Oracle database version 10g Release 2.

The Oracle wallet can be initialised by issuing the following command, which will: create a wallet, create a master key for the entire database, and open the wallet.

```
ALTER SYSTEM SET encryption key identified by "hdgr57fnle39dncv";
```

Subsequently, after each table creation with an encrypted column specification will cause the TDE to create a separate key for each table.

F.1.1 Key storage for single-user spatial files

The following script describes the table used to store the different spatial user's DFS spatial file encryption and decryption keys.

```
CREATE TABLE CMSDK_UserSpecificFile_keys(
  cmsdk_user_id NUMBER,           -- CMSDK user id
  enc_key    VARCHAR2(100) ENCRYPT USING 'AES128', -- encryption key
  dec_key    VARCHAR2(100) ENCRYPT USING 'AES128' -- decryption key
);
```

The administrator can simply create or remove single-user keys for the DFS users by issuing the following SQL commands. For instance, to add and delete cryptographic keys for all files owned by GIS user Scott, identified by CMSDK user id 55514, the SQL commands are:

```
INSERT INTO CMSDK_UserSpecificFile_keys (cmsdk_user_id, enc_key, dec_key) VALUES
(55514,'G#^*!JS^MN!468h6', 'G#^*!JS^MN!468h6');
DELETE FROM CMSDK_UserSpecificFile_keys WHERE CMSDK_user_id = 55514;
```

The similarity of the keys stored in this table for specific users depend on whether symmetric or asymmetric algorithms are used. To retrieve the encryption keys of a specific Internet File System user from this table, the `get_enc_key()` and `get_dec_key()` functions.

F.1.2 Temporary LOB location storage for single-user spatial files

The following script describes the table used to store the user temporary LOB locators internally by the Package.

```
CREATE TABLE CMSDK_user_temp_lobs(
  CMSDK_user_id NUMBER,           -- CMSDK user id
  CMSDK_file_id NUMBER,           -- User File id
  CMSDK_file_name VARCHAR(100),   -- CMSDK file name
  templob    BLOB                 -- temporary LOB
);
```

The table is populated with the CMSDK user's file temporary LOB locator within the temporary tablespace as soon as the file is decrypted. Upon user logout, the LOB locator is used to copy the LOB back into its original location.

F.1.2 Multi-user, project-specific file key storage

The following script describes the database table that stores spatial file's filepath, the encryption and decryption keys for that file, and the state of file that is checked before trying to re-encrypt or re-decrypt files in order to prevent corruption.

```
CREATE TABLE CMSDK_MultiUserFile_keys_state (
  CMSDK_filepath VARCHAR2(500),    -- Full CMSDK filepath
  enc_key    VARCHAR2(100) ENCRYPT USING 'AES128', -- encryption key
  dec_key    VARCHAR2(100) ENCRYPT USING 'AES128', -- decryption key
  file_encrypted VARCHAR2(5) DEFAULT 'FALSE' -- file state
  templob    BLOB           -- temporary LOB
);
```

The administrator can issue the following commands to add or remove filepaths, the encrypted state, and the encryption and decryption keys of the file. For instance, to add and delete spatial file **spottext.MAP** with full CMSDK filepath **"projects\classified\spottext.MAP"**, the SQL commands are:

```
INSERT INTO CMSDK_MultiUserFile_keys_state (CMSDK_filepath,file_state, enc_key,dec_key) VALUES
('projects\classified\spottext.MAP','hfghfdg657hvg^*T','hfghfdg657hvg^*T');
DELETE FROM CMSDK_MultiUserFile_keys_state WHERE CMSDK_filepath LIKE
"projects\classified\spottext.MAP";
```

Once again, the similarity of the keys stored in this table for specific users depend on whether symmetric or asymmetric encryption algorithms are used. To retrieve the encryption keys of a specific CMSDK user from this table, the `get_file_enc_key()` and `get_file_dec_key()` functions are used.

F.1.3 Multi-user, project specific user-file catalog

The following script describes the database table that stores the filepath and the different CMSDK users who require access to them in decrypted form.

```
CREATE TABLE CMSDK_MultiUserFile_user_catalog(
  CMSDK_filepath VARCHAR2(500), -- CMSDK MultiUserFile Full filepath
  CMSDK_user_id  NUMBER         -- CMSDK user id
);
```

Using the following SQL commands, the administrator can insert or delete files, CMSDK users, and the files that should be decrypted for use by a particular user and encrypted back for storage. For instance, to add and delete cryptographic keys for all files owned by GIS user Scott, identified by CMSDK user id 55514, the SQL commands are:

```
INSERT INTO CMSDK_MultiUserFile_user_catalog (CMSDK_user_id, CMSDK_filepath) VALUES (55514,  
'projects\classified\spottext.MAP');
```

```
DELETE FROM CMSDK_MultiUserFile_user_catalog WHERE CMSDK_user_id = 55514 AND WHERE  
CMSDK_filepath LIKE 'projects\classified\spottext.MAP';
```

Appendix G Metadata and content storage in CMSDK

This appendix includes a description of file metadata and content are storage in CMSDK. It describes how CMSDK links filenames to their file id as well as how it stores the file-folder relationships. The CMSDK Spatial Crypt package is developed on this storage scheme, and this may be useful for understanding the code structure.

F.1 Obtaining information on spatial data files stored in CMSDK

In Oracle CMSDK, metadata (data about data) and document content are stored separately. The file metadata is stored in the: ODM_PUBLICOBJECT table which stores the basic information common to any file that can appear in a folder such as the name, last modification date, creator, owner, etc, and the ODM_DOCUMENT which stores the id the file which is used to identify it in the content table.

File content (the body of the document) is stored as BLOB datatype in the either of two separate tables; one for files which can be indexed using the ODMM_CONTENTSTORE table and the other for files which cannot using the ODMM_NONINDEXEDSTORE. CMSDK distinguishes between the two by keeping a predefined list of file extensions that contain searchable text, for instance, files with .doc file extension (Word documents), which normally have searchable information.

The tables ODM_PUBLICOBJECT and ODM_DOCUMENT effectively take on the role of the File Allocation Table (FAT), while the ODMM_CONTENTSTORE and ODMM_NONINDEXEDSTORE tables are used to store the body of the document. The figure below illustrates how the spatial files, Places.MAP and Locations.doc files are stored in the CMSDK repository.

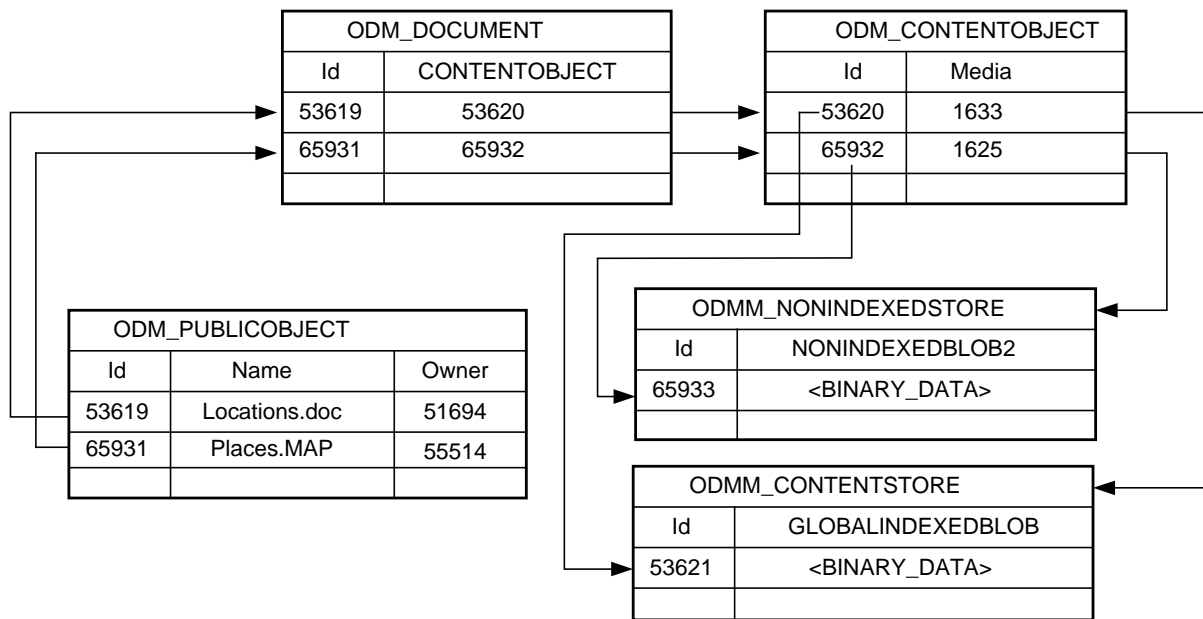


Figure 18: Spatial file metadata and content storage in CMSDK.

The CMSDK repository differentiates files in CMSDK using unique id, therefore performing any kind of operation on the LOB requires the file id first. In CMSDK, the id of the file is incremented by one when represented in the ODM_DOCUMENT table and then incremented by one again when stored in the in the data storage tables. The id of files are retrieved using the `get_10gcmsdk_user_files()` procedure, which takes the filename and queries the ODM_PUBLICOBJECT directly to obtain the id which it directly increments by 2 while passing for encryption and decryption using the encrypt and decrypt procedures. These encrypt and decrypt procedures, then again, use the id passed as parameter to query the ODM_CONTENTOBJECT table to get in which table the file content is stored as BLOB datatype.

While basing queries using the owner is described, the multi-user files which necessary do not belong to any specific user and obtaining the file id was more trivial.

F.2 Folder/document and folder/subfolder relationships in CMSDK

In CMSDK, folders, subfolders and file ids are stored in ODM_PUBLICOBJECT table while their relationships are stored in the ODM_RELATIONSHIP table and are constructed using the columns, LEFTOBJECT (parent) and RIGHTOBJECT (child). The figure below outlines how the relationships are stored in Microsoft Windows version of CMSDK, using the spatial file Places.MAP, with CMSDK filepath "O:\home\scott\Places.MAP".

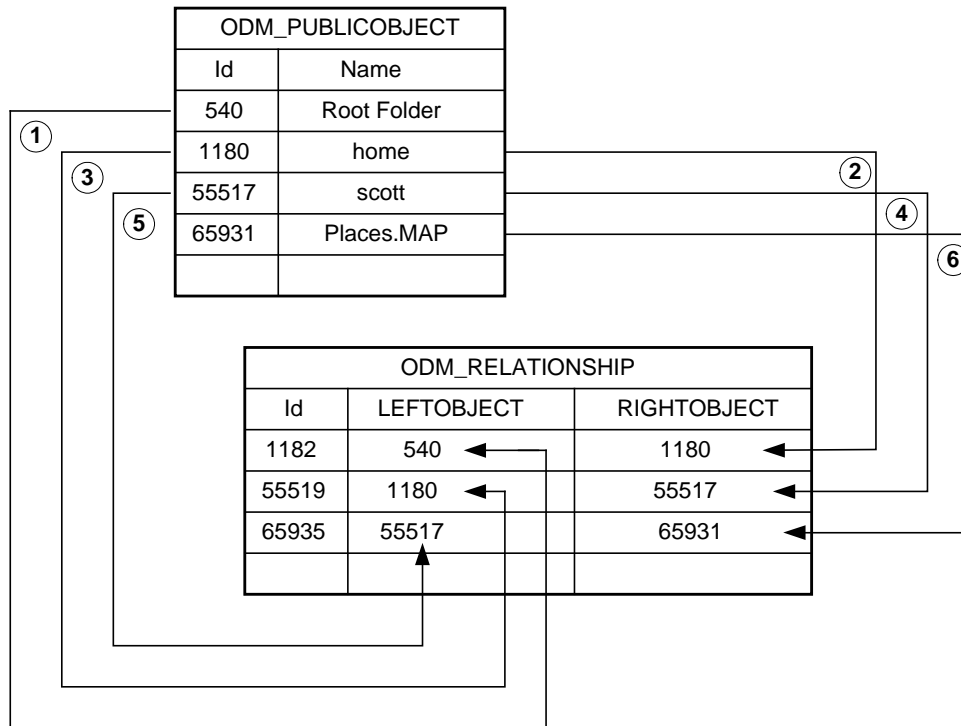


Figure 19: A Folder/Document Relationship and Folder/Subfolder Relationship in CMSDK.

The Root Folder is always a constant number (in this case it is 540), and for CMSDK version for Windows, it is a representation of the O:\ drive. Thereafter, the parent object (folder) ID is stored as the LEFTOBJECT in the Relationship record, and the child object (folder or file) ID, is stored as the RIGHTOBJECT. The relationships for: Root Folder and home folder, home folder and Scott's home folder "Scott", and "Scott" folder and Places.MAP file, follow this arrangement. To obtain the id of the file using the full CMSDK filepath, complex SQL SELECT statements are performed in recursive loops on these two tables. See split() function.