



Topic: SOS Database Encryption
Document ID: #111
Product: ALL
Date: December 17, 2007
Author: Seth Krieger

Overview

By default, the SOS database is not encrypted, though the physical layout of the database files is such that assembling a coherent record of data would be difficult. Nonetheless, inspection of the database files could reveal identifying information, so some level of encryption is recommended in most all situations, and always if the database is transported from place to place on a portable computer or on removable media (such as removable disk, CD, DVD, USB memory key, or tape) or if the database is transmitted from one location to another electronically.

You may choose to protect the data in the database by selecting either "simple encryption" or "strong encryption" (AES, Advanced Encryption Standard).

Simple Database Encryption

Simple encryption is equivalent to obfuscation and prevents someone using a disk utility, text editor, or even a word processor from being able to read patient names or other information stored in the database files. Simple encryption uses a built-in key and therefore does not require the user to provide a key to encrypt the database. Once encrypted, the information can still be accessed using the SOS and Sybase programs exactly as with an unencrypted database. In addition, utilities such as DBTRAN (which translates the transaction log into readable text) can still be used by anyone who knows how, without the need to provide a password or encryption key. It is therefore essential to secure the computer and drive on which the transaction log (SOSDATA.LOG) is stored, even if using simple encryption. The folder(s) in which the database and log files are stored should NEVER be shared across a network for this reason. Windows shares are entirely unnecessary for use of the SOS system, and for the reasons just given, undercut efforts to keep your data secure.

Strong Database Encryption

Strong encryption is a bit more trouble to configure but renders the database completely inaccessible without an encryption key. **There is no back door or recovery possible if the encryption key value is lost or forgotten.**

Changing the Encryption Level of an Existing Database

In order to change the encryption level of your database, you must rebuild it. Starting with Release 2007.02 (December, 2007 build), the SOS Database Rebuild Utility includes the following encryption options:

Encryption Option	Advantages	Disadvantages
No encryption	No rebuild required. Best performance. Best chance of database recovery in the event of corruption. No encryption key to remember or safeguard.	Patient information, such as names, can be read by opening the database files with a text editor or word processor. Unless physical access to the database is carefully restricted, this option could constitute a HIPAA violation.
Simple encryption	No encryption key to remember or safeguard. Minimal performance or database recovery impact. Obscures data in database files, preventing it from being accessed by unsophisticated methods.	Does not provide any encryption of the database log (the sosdata.log) file, which can be translated to readable text using a Sybase utility. Any user can still start the database, and once it is started, can access data as long as he or she has an SOS password that permits access.
Strong encryption (recommended)	Very strong data protection. Your database and log cannot be started or accessed without the encryption key that you have created.	Most importantly, loss of your encryption key will render your database useless. Secondly, use of strong encryption will incur a performance penalty, in all likelihood in the range of two to five percent, though there are many variables in play.

When selecting the *Strong Encryption* option, you must provide an encryption key, which is a string of characters that will be used to encrypt and decrypt the contents of your database and transaction log. The normal rules for password generation apply:

- Y Use no fewer than 8 characters. SOS supports keys of up to 60 characters. A length of 8 to 30 is recommended. Longer keys are more difficult to crack, but theoretically could result in slower performance. The better your hardware, the less likely you would be able to detect a difference.
- Y Include upper and lower case letters, plus numbers and special characters in your encryption key. The key may contain any characters on your keyboard except <space>, semi-colon, apostrophe, and quotation mark characters.
- Y Strong keys are really random - no pet names or birth dates! Many free and low cost password generators are available to assist in generating random sequences of characters for use as passwords and encryption keys. One example is Quicky Password Generator from www.quickysoftware.com. The very popular utility Roboform (www.roboform.com) also includes a password generation utility.

Whenever you select or change your SOS database encryption password, we STRONGLY recommend that you call SOS and have it recorded in your account records at SOS. There is no way to start or recover your database without it, so having a backup copy at SOS is an easy way to assure that you will not lose your data because of a forgotten key.

Performance Considerations Related to Database Encryption

There is a performance penalty for the use of encryption. The penalty is slight for the simple encryption method, but can be more significant when choosing strong encryption. If using strong encryption, the customer should pay even more attention to the selection of server hardware, choosing faster processors and disk systems, and, especially, more RAM than might otherwise be necessary. If the amount of RAM is sufficient to permit the entire database to be cached, the performance penalty should diminish and become relatively small after a period of active use. Note also that the length of the encryption key is related to performance, so if you are using a very long key and have performance issues, you can try rebuilding the database with a shorter key.

Recommendations

Unless you are using seriously underpowered hardware, SOS recommends that you implement at least the *Simple Encryption* option. If your database is carried out of the office on a laptop or in unencrypted backups (such as a straight copy of the DATA folder onto a CD or DVD), then serious consideration should be given to using *Strong Encryption*. In that case, it is essential that you carefully think through the entire process. Precautions must be taken to assure that only the most trusted employees are able to log into the server to encrypt the database. Losing the only employee who knows the encryption key could be disastrous.

How To Change Your Current Encryption Level

To change the current encryption level of your database you must rebuild it using the appropriate options. Once you have reviewed the information below and decided on an encryption level, have all users exit all SOS modules, backup your database as you normally do, then start the rebuild as follows:

1. **On the computer where the database is located** start SOSLogin.
2. Enter an account ID and password of a user with SOS security administrator privileges.
3. Click the “Admin” icon (the keys).
4. Select Database Tools.
5. On the list of utilities, select and run “Database Rebuild Utility”

For detailed instructions about running the Rebuild Utility, see document 127 in the SOS web site document library (Support > Document Library) or use the direct link below:

<http://www.sosoft.com/fod/doc127-dbrebuild.pdf>

Starting an Encrypted Database

Once the database has been encrypted, whenever the database is started, the encryption key must be provided on the command line when the database is started, or the database engine must be instructed to prompt for the encryption key as it starts the database.

By default, SOS includes database engine startup parameters in a file in the SOS folder. In standalone implementations, this file is called STDALONE.PRM; in network implementations it is called SERVER.PRM. This file contains options for the database *engine*, which could, potentially run more than one database at a time. Options for each database run by the engine cannot be added to the engine options file. They must be placed *after* each database file on the startup command line. The following examples reference the network engine, DBSRV9.EXE. Just substitute DBENG9.EXE for standalone installations. All the following examples would be on a single line:

Default command line (simple or no encryption options):

```
c:\sos\asa\win32\dbsrv9.exe @c:\sos\server.prm c:\sos\data\sosdata.db
```

Command line to prompt the user for the encryption key (-ep) every time the database is started:

```
c:\sos\asa\win32\dbsrv9.exe @c:\sos\server.prm c:\sos\data\sosdata.db -ep
```

Command line with encryption key specified explicitly:

```
c:\sos\asa\win32\dbsrv9.exe @c:\sos\server.prm c:\sos\data\sosdata.db -ek secretkey
```

The last option is obviously less than ideal because the encryption key appears in plain text in the startup line invoked by your shortcut, batch file, etc. There is, however, an alternative. That part of the command line can be hidden using a provided utility that will hide the text from all but expert snoopers. When you select the *Strong Encryption* option during the rebuild, your encryption key will automatically be saved in an unreadable file called OPTION2.PRM, which you will find in the SOS folder as well as the DATA folder. You can therefore use this variation of the command above (entire command is typed on a single line):

```
c:\sos\asa\win32\dbsrv9.exe @c:\sos\server.prm c:\sos\data\sosdata.db  
@c:\sos\option2.prm
```

If you wanted to create the unreadable option2.prm file manually, here is how it would be done:

1. Create a plain text file using Notepad or another text editor.
2. Enter -ek and the encryption key with which your database was encrypted.
3. Save the file as TEMP.TMP in your SOS folder.

4. Now apply simple encryption to the file, saving the encrypted copy as OPTION2.PRM with the command:

```
c:\sos\asa\win32\dbfhide temp.tmp option2.prm
```

5. You will now have the original TEMP.TMP and the encrypted OPTION2.PRM in your SOS folder. Once you are sure that the encryption key value is safely recorded in case you should ever need it (SOS recommends that you call SOS and have the value stored in your account records at SOS), delete the TEMP.TMP file. Hold down the shift key while deleting to be sure the file is deleted instead of just being moved to the recycle folder. If you have a utility for secure deletion of files, that would be even better.

Now you can start the database using the following command, which does not expose the key:

```
c:\sos\asa\win32\dbsrv9.exe @c:\sos\server.prm c:\sos\data\sosdata.db @option2.prm
```

A strongly encrypted database cannot be opened without its matching encryption key. For that reason it is essential that you either backup the option2.prm file containing your key, or securely record the key in a log that will document changes in the key over time. If you ever have to restore your data, you will need the key that was in use when that database was backed up.

Including Your Encryption Key in the ODBC Configuration

In standalone installations, the database is rarely, if ever, started directly. Instead, it is automatically started whenever needed using a start command specified in the ODBC configuration named SOSDATA (located by default on the System DSN tab). As mentioned above, you should not simply type the encryption key in the indicated field on the Database tab because it will appear in clear text in the Windows Registry. Instead, follow the instructions in the section above to create an OPTION2.PRM file that contains an encrypted version of the database encryption option and your key. You must then modify the ODBC configuration, specifically the Database tab entries as follows, assuming default SOS folders and a standalone installation:

Server name	SOSDATA
Start line (typed all on one line)	c:\sos\asa\win32\dbeng9.exe @c:\sos\stdalone.prm c:\sos\data\sosdata.db @option2.prm
Database name	SOSDATA
Database file	<leave this field empty>
Encryption key	<leave this field empty>
Start database automatically	checked
Stop database after last disconnect	checked

On a network server, you can use a similar configuration. Just replace dbeng9.exe with dbsrv9.exe and stdalone.prm with server.prm in the Start line field.

Running an Encrypted Database as a Windows Service

If prompting for a password on startup (the -ep parameter), be sure to use the -i parameter when creating your service so the service can interact with the console. As the service starts, a window will appear in which you must type the encryption key.

The following example (typed all on one line) creates a service that prompts for the encryption key:

```
c:\sos\asa\win32\dsbvc -as -i -t network -w mysos c:\sos\asa\win32\dsbrv9.exe
@c:\sos\server.prm c:\sos\data\sosdata.db -ep
```

You must include the -i (allow service to interact with the desktop) option when you use the -ep (prompt for encryption key) option. When configured in this fashion, someone must be present at the server console to type the encryption key when the service starts.

Although this option would be the most secure, it may well not be practical. The alternative would be to pass the encryption key value on the service startup line automatically. The problem is that if you type the encryption key in your startup command, it is not secure. To avoid having this value exposed, the -ek parameter, along with the encryption key, may itself be encrypted in a form that the database engine will be able to decrypt. When you rebuild the database the utility will create the appropriate, ready-to-use, encrypted key parameter in a file called option2.prm. You can also create that file manually as follows:

1. Create a plain text file using Notepad or another text editor.
2. Enter -ek and the encryption key with which your database was encrypted.
3. Save the file as TEMP.TMP in your SOS folder.

4. Now encrypt the file, saving the encrypted copy as OPTION2.PRM with the command:
`c:\sos\asa\win32\dbfhide temp.tmp option2.prm`
5. You will now have the original TEMP.TMP and the encrypted OPTION2.PRM in your SOS folder. Once you are sure that the encryption key value is safely recorded in case you should ever need it (SOS recommends that you call SOS and have the value stored in your account records at SOS), delete the TEMP.TMP file. Hold down the shift key while deleting to be sure the file is deleted instead of simply moved to the recycle folder. If you have a utility for secure deletion of files, that would be even better.
6. Now create your service using the following format (typed all on one line, with the appropriate changes if your installation is not in C:\SOS):
`c:\sos\asa\win32\dsbvc -as -i -t network -w mysos c:\sos\asa\win32\dsbrv9.exe
@c:\sos\server.prm c:\sos\data\sosdata.db @c:\sos\option2.prm`

Note: For additional information and description of all available command line options for service creation, see: <http://www.sosoft.com/fod/doc430-startingdbasservice.pdf>

Transport Layer Encryption

In addition to encryption of the database itself, a customer might be concerned about protecting the data while it is in transit between the server and the client workstations. Various network protection approaches are available, including VPN, wireless encryption, and use of Windows Terminal Services or Citrix.

As with database file encryption discussed previously, there are simple and strong encryption options available for transport-layer security. An easy way to avoid sending easily read data is to add `-ec simple` to your server startup options in the SERVER.PRM file in your SOS folder. On the client side, open the ODBC system data source called SOSDATA. Select the Network tab. Set the option at the bottom (“Select the method for encryption of network packets”) to “Simple”. Doing so provides the same kind of obfuscation for your database communication packets that simple encryption does for your database files. It will not secure the data from an encryption expert or “hacker” who is determined to eavesdrop on your communications, but it will prevent less sophisticated individuals from reading the contents of your transmission packets by simply hooking up a packet sniffer tool.

This sort of simple encryption has a minimal impact on performance.

Sophisticated, strong, transport-layer encryption options for your database communications are also available, using RSA or ECC certificates. As stated previously, if you want that level of protection, you should probably investigate strategies that will protect *all* your network communications, not just database packets. In addition, this kind of database-specific encryption requires purchase of an additional component, which is priced by number of user connections.