Secure Database Backups with SecureZIP

Approved procedures for insuring database recovery in the event of a disaster call for backing up the database and storing a copy of the backup offsite.

Given the sensitive nature of the data in many corporate databases, an additional layer of protection may be warranted as well—namely, to encrypt backups so that the data cannot be accessed by unauthorized persons. Data to be moved offsite is doubly vulnerable to this threat because it spends time in transit. For best security, data to be moved offsite should always be encrypted.

Integrate SecureZIP into Existing Backups

Companies invest considerable resources in the backup and archival systems they use to protect their data. The systems are tightly integrated with the DBMSs and are apt to be scheduled to run in tight time frames. Extending the time frames may have adverse effects on system availability, and making major changes to existing procedures may not be a viable option either.

SecureZIP provides the encryption necessary to secure your backups with minimal impact on either your existing backup routines or on the time it takes to execute them. SecureZIP supports both password-based and certificate-based encryption using strong, AES 256-bit algorithms for enterprise-strength security. SecureZIP is built on PKWARE's standard ZIP file format, long trusted for data backups on all major computing platforms.

This guide shows how simple it can be to incorporate SecureZIP with native DB2 and SQL Server backup methods, to gain securely encrypted backups with next to no overhead.

Contents

DB2 Backup on z/OS: Example

DB2 databases can be backed up in many ways. In this example, we focus on a common DB2 Image Copy scenario: We make full-image copies for a local site and a recovery site.

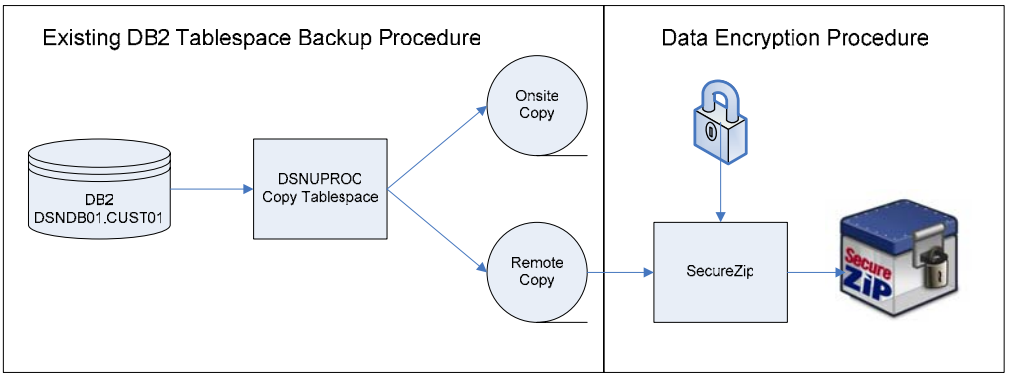

Figure 1

In the figure, the box on the left shows a DB2 full image copy procedure that creates two parallel copies, one for the local site and one for the recovery site. The box on the right represents an added step to secure the remote copy. In this step, SecureZIP compresses and encrypts the copy into a ZIP archive. This protects the copy both in transit and at the recovery site. Needless to say, the onsite copy can be encrypted as well.

Encryption can be inserted into the procedure anytime after the image copy is made. The resulting ZIP archive can be sent directly to a remote site, or another image copy can be added to it.

As shown in the sample code below, this example uses *asymmetric encryption*, also known as *public key encryption*. In contrast with password-based encryption, which uses the same key for both encryption and decryption, asymmetric encryption uses two different cryptographic keys, one private and one public. These keys are called a *key pair*. They are associated through an X.509 V3 digital certificate, which contains the public key, an expiration date and other information, and identifies the owner.

A backup encrypted with the public key can be decrypted only by applying the private key. The owner of the key pair (the DB2 administrator, in this case) keeps the private key in his own possession and publishes the public key (for example, to an LDAP server) so that it is available to anyone who may need to encrypt something for him. With the public key, anyone can encrypt database backups, but only the DB2 administrator (or a delegate having access to the DB2 administrator private key) can decrypt them.

The following sample jobstream encrypts a DB2 image copy of the CUST01 tablespace in the DSNDB01 database. The procedure has two steps: make the copies, and encrypt the copy for the remote site.

```
//CUSTIC EXEC DSNUPROC,UID='CUST.COPYTS',
//       UTPROC='',
//       SYSTEM='DSN',DB2LEV=DB2A
//COPY1 DD DSN=DB2.BACKUP.CUST01.LB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//COPY2 DD DSN=DB2.BACKUP.CUST01.RB,
//         SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//SYSIN DD *
OPTIONS EVENT(ITEMERROR,SKIP)
COPY TABLESPACE DSNDB01.CUST01
    COPYDDN(COPY1)
    RECOVERYDDN(COPY2)
    PARALLEL(2)
/*
//SZCUST EXEC PGM=SECZIP
//STEPLIB DD DISP=SHR,DSN=SECZIP.MVS.LOAD
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
-ENCRYPTION_METHOD(BSAFE_AES256)
-DATA_TYPE(BINARY)
-SAVE_LRECL(Y)
-INCLUDE_CMD(SECZIP.MVS.JCL(DBPROF))
-INCLUDE_CMD(SECZIP.MVS.JCL(LDAPPROF))
-RECIPIENT(LDAP:CN=REMOTESITE*)
-ARCHIVE_DSN(DB2.BACKUP.CUST01.RB.ZIP)
-ARCHIVE_DSORG(PS)
-ACTION(ADD)
DB2.BACKUP.CUST01.RB
/*
```

In this example, *–ENCRYPTION_METHOD(BSAFE_AES256* specifies the RSA BSAFE version of the AES 256-bit encryption algorithm. The Advanced Encryption Standard (AES) algorithm is the latest generation of encryption standards used by the US Government. SecureZIP supports three key sizes for the AES algorithm: 128, 192 and 256 bits. Longer key sizes make for greater security.

–RECIPIENT(LDAP:CN-REMOTESITE)* specifies a recipient search. A *recipient* is the owner of a key pair. We are searching for the digital certificate that contains the recipient's public key that we will use to encrypt the database backup. In this case, we are searching for the public key certificate that starts with a common name of *REMOTESITE*.

When the example is run, it first creates the image copy and then encrypts the database backup into the compressed archive DB2.BACKUP.CUST01.RB.ZIP. Only the recipient owner of the REMOTESITE private key can decrypt the backup.

DB2 Restore on z/OS: Example

This example assumes you are restoring in a recovery scenario. The example first shows how to get SecureZip up and running. Then SecureZIP is used to unzip and decrypt the DB2 backups. The example assumes that the operating system and DB2 have already been restored if necessary.

The example describes restoring SecureZip on the recovery site in a case where it is necessary to install SecureZip and import private keys needed to decrypt. If SecureZIP can simply be restored from backup, it is not necessary to reinstall it and import private keys.

Restore SecureZip Without a Backup

To restore SecureZIP without a backup, follow these steps:

1. See the SecureZIP installation instructions in the SecureZIP *System Administrators Guide* and follow the steps for either the SMP/E or non-SMP/E installation procedures. You must also initialize the license for SecureZip. The license key does not need to match the serial number of the CPU that you are running on for recovery. SecureZip provides an automatic five-day licensing grace period to accommodate disaster recovery scenarios.

If certificate-based decryption will be used, activate the ISPF interface. The steps are described in the *System Administrator Guide*. The ISPF interface makes certificate administration easier. If only password-based decryption is needed (and consequently there is no need to access a private key), it's not necessary to activate the ISPF interface, and you can skip the next step, below, of importing a private key.

2. Import the private key of a recipient for whom the database backups were encrypted.

   With certificate-based encryption, only a recipient for whom a file was encrypted can decrypt the file. A file encrypted for one or more recipients can be decrypted (only) with the private key of any one of the recipients.

   For SecureZIP to access a private key, the key must first be placed in the local certificate store. For instructions on importing a private key into the store, see the section "Add a Certificate to the Local Store" in Chapter 4 of the *System Administrator Guide*.

   Note that SecureZip has a MASTER_RECIPIENT option that allows for use of a contingency key. A contingency key enables an enterprise to decrypt and access file(s) when other, named recipients are no longer able or eligible. In other words, the option provides for contingency access to data. If database backups are encrypted with a MASTER_RECIPIENT contingency key, then only the corresponding MASTER_RECIPIENT private key needs to be imported to decrypt them.

SecureZip Unzip and Decrypt

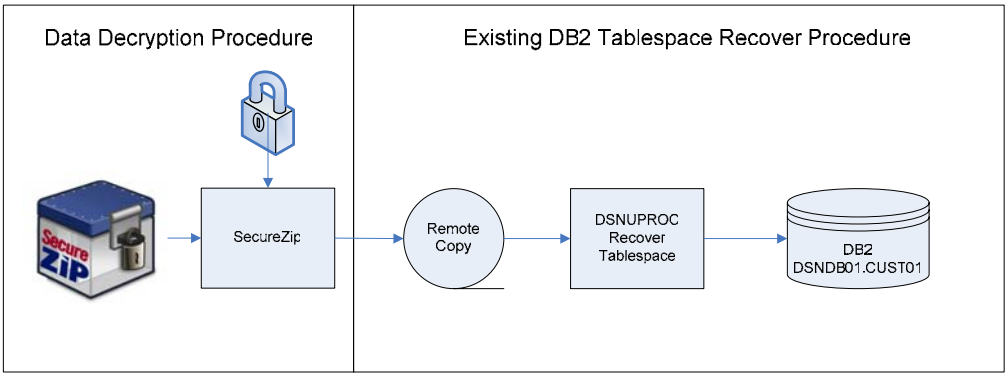

Figure 2

Once SecureZip is installed and the necessary private keys are imported, the backup can be decrypted and extracted from the ZIP archive. The operations of extracting and decrypting can be done in one step, as shown in the following code sample.

The sample JCL below runs the program SECUNZIP to decrypt and extract the backup. The archive name used in the example is *ARCHIVE_DSN DB2. BACKUP.CUST01.RB.ZIP*. The password in the PASSWORD parameter in the RECIPIENT option is used only to access the private key; it is not used to decrypt the backup itself. The private key is used for that.

```
//SZCUST EXEC PGM=SECUNZIP
//STEPLIB DD DISP=SHR,DSN=SECZIP.MVS.LOAD
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
-ARCHIVE_DSN(DB2.BACKUP.CUST01.RB.ZIP)
-DATA_TYPE(BINARY)
-INCLUDE_CMD(SECZIP.MVS.JCL(DBPROF))
-RECIPIENT(DB:CN=REMOTESITE,R,PASSWORD=$EcR3t P@$$w0rd)
-EXTRACT
-UNZIPPED_DSN(**,DB2.BACKUP.CUST01.RB)
/*
//*
//CUSTRT EXEC DSNUPROC,UID='CUST.COPYTS',
//       UTPROC='',
//       SYSTEM='DSN',DB2LEV=DB2A
//SYSIN DD *
RECOVER TABLESPACE DSNDB01.CUST01
/*
```

SQL Server Backup on Windows Server 2003: Example

This example shows a Transact-SQL procedure used to back up an SQL Server database to a disk device. SecureZip provides a command-line interface for easy process integration and can be used to backup SQL Server in a number of ways. The Transact-SQL procedure shown here illustrates a common case.

The example assumes that the disk backup devices have already been created using the following commands. These commands need to be executed only once.

```
USE master
EXEC sp_addumpdevice 'disk', 'PubsBackup',
   'c:\SQLData\MSSQL\BACKUP\PubsBackup.bak'

EXEC sp_addumpdevice 'disk', 'PubsLogBackup',
   'c:\SQLData\MSSQL\BACKUP\PubsLogBackup.bak'
```

Once the backup devices have been created, the Transact-SQL backup procedures can be executed as usual. These procedures can be stored procedures called by SQLAgent or batch processes executed by ISQL.

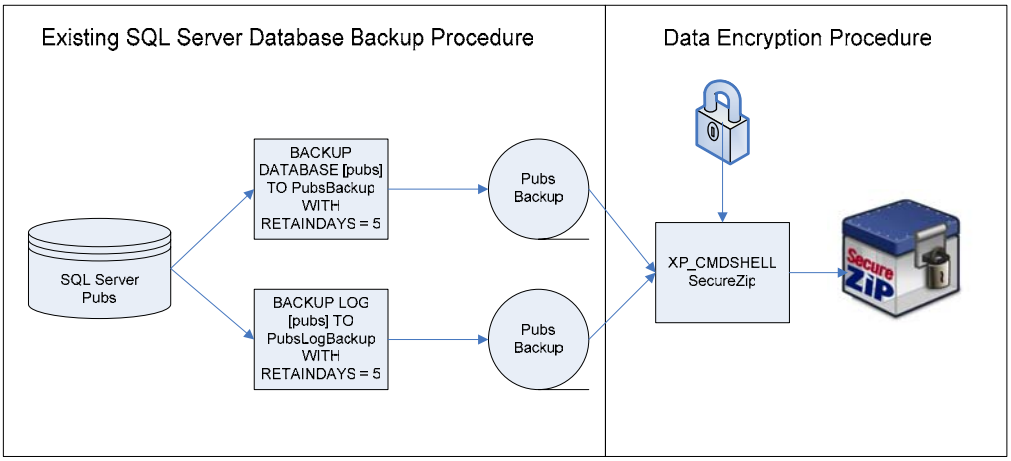

Figure 3

On the left, Figure 3 shows the backup procedures for the existing backup. The procedure on the right is added to compress and encrypt the database backups and transaction log backups. Compressing makes the encryption process more efficient by reducing the amount of data to be encrypted. This can reduce the overall time needed and thus minimize the impact on the backup window.

By default, SecureZip compresses the database backups before encrypting them. This is significant because database backups usually have a very high compression ratio, so compressing them first greatly reduces the amount of data to encrypt. For example, if a 1.5MB Pubs Database backup yields a not-unreasonable compression ratio of 89.7%, the resulting encrypted ZIP archive is only 155KB.

After the Data Encryption Procedure is complete, the encrypted archive can be sent to a remote site, or another database backup can be added to the archive.

This example, like the previous one, uses asymmetric, public key encryption. The owner of the key pair (the SQL Server administrator) holds the private key and publishes the public key so that anyone can use it to encrypt database backups. But only the holder of the corresponding private key can decrypt the backups.

The procedure to back up the Pubs database in the example and to compress and encrypt the subsequent backup device is given below.

```
BACKUP DATABASE [pubs]
       TO PubsBackup
       WITH RETAINDAYS = 5

EXEC xp_cmdshell 'pkzipc -add
-ldap=10.1.1.1:389/ou=abc_inc,
dc=abc,dc=com -recipient="backup@abc.com"
c:\SQLData\MSSQL\BACKUP\PubsBackup.zip
c:\SQLData\MSSQL\BACKUP\PubsBackup.bak'
```

In this example, we run SecureZip in the SQL Server XP_CMDSHELL extended stored procedure. We encrypt with the backup certificate located on the LDAP server located at 10.1.1.1, port 389. The resulting encrypted archive is located at `c:\SQLData\MSSQL\BACKUP\PubsBackup.zip`, identified by the backup device `c:\SQLData\MSSQL\BACKUP\PubsBackup.bak`.

The procedure to back up the Pubs transaction log and compress and encrypt the backup device is shown below.

```
BACKUP LOG [pubs]
       TO PubsLogBackup
       WITH RETAINDAYS = 5

EXEC xp_cmdshell 'pkzipc -add
-ldap= 10.1.1.1:389/ou=abc_inc,
dc=abc,dc=com -recipient="backup@abc.com"
c:\SQLData\MSSQL\BACKUP\PubsLogBackup.zip
c:\SQLData\MSSQL\BACKUP\PubsLogBackup.bak'
```

Like the database backup, the backup of the transaction log is executed using the SQL Server extended stored procedure XP_CMDSHELL and is encrypted with the `backup@abc.com` certificate.

SQL Server Restore on Windows Server 2003: Example

The preceding backup example used the recipient's public key to encrypt. A recipient is a person authorized to decrypt the encrypted backup data. Since a public key does not need to be protected, it can simply be placed on Active Directory. From there it can be retrieved using the `–ldap` command-line parameter.

In a restore operation, on the other hand, an encrypted database backup archive needs to be *decrypted*. Decrypting requires access to the private key, which *does* need to be protected. Because the private key is protected,

decryption requires an extra step. So doing a restore is a two-step process: first, decrypt the backup using the administrator private key; and then do a SQL Server restore.

To decrypt the database backups, the administrator with access to the private key runs a script to decrypt the database backup. The script includes the SecureZIP commands and options needed to extract and decrypt the backup from the archive. SecureZip automatically looks for the administrator's private key in the administrator's Windows Personal Store.

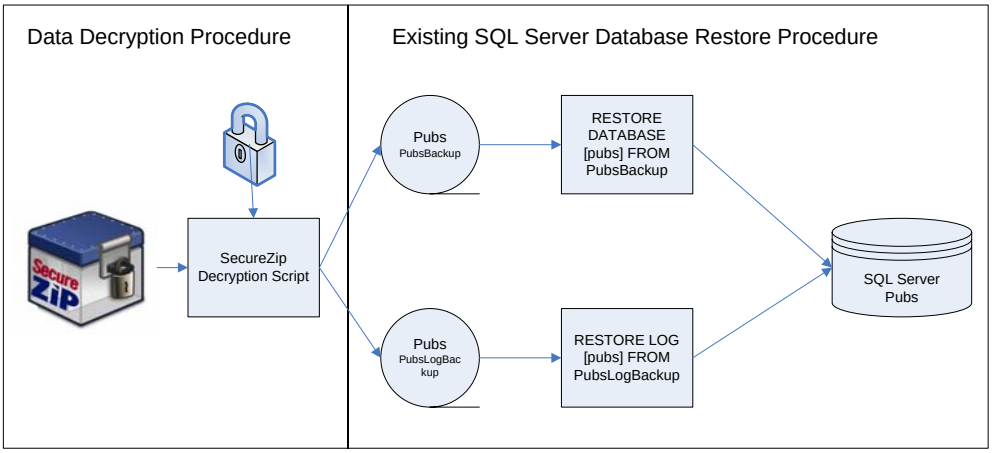

Figure 4

The SecureZIP command line below decrypts and uncompresses the database backup in the `PubsBackup.zip` archive to the directory `c:\SQLData\MSSQL\BACKUP\`. To decrypt, the administrator running the command line (which can be in a script) must have the private key for the `backup@abc.com` recipient in his Personal Store.

```
pkzipc –extract PubsBackup.zip c:\SQLData\MSSQL\BACKUP\
```

Once the SQL Server backup device has been decrypted and decompressed into the SQL Server backup directory, the backup device is now available for the RESTORE command to restore the Pubs database.

```
RESTORE DATABASE [pubs]
       FROM PubsBackup
```