What's new in Eloquence B.08.10

Overview

- Released July 2010 for the HP-UX and Linux platform
- Supported until June 2015

- Supports the same 32-bit and 64-bit platforms as B.08.00
  - HP-UX Itanium and PA-RISC
  - Linux x86, x64 and Itanium
  - Windows x86 and x64

- Installed in a separate location and may be used concurrently with previous Eloquence versions on HP-UX and Linux
- Windows release scheduled to be available later in 2010

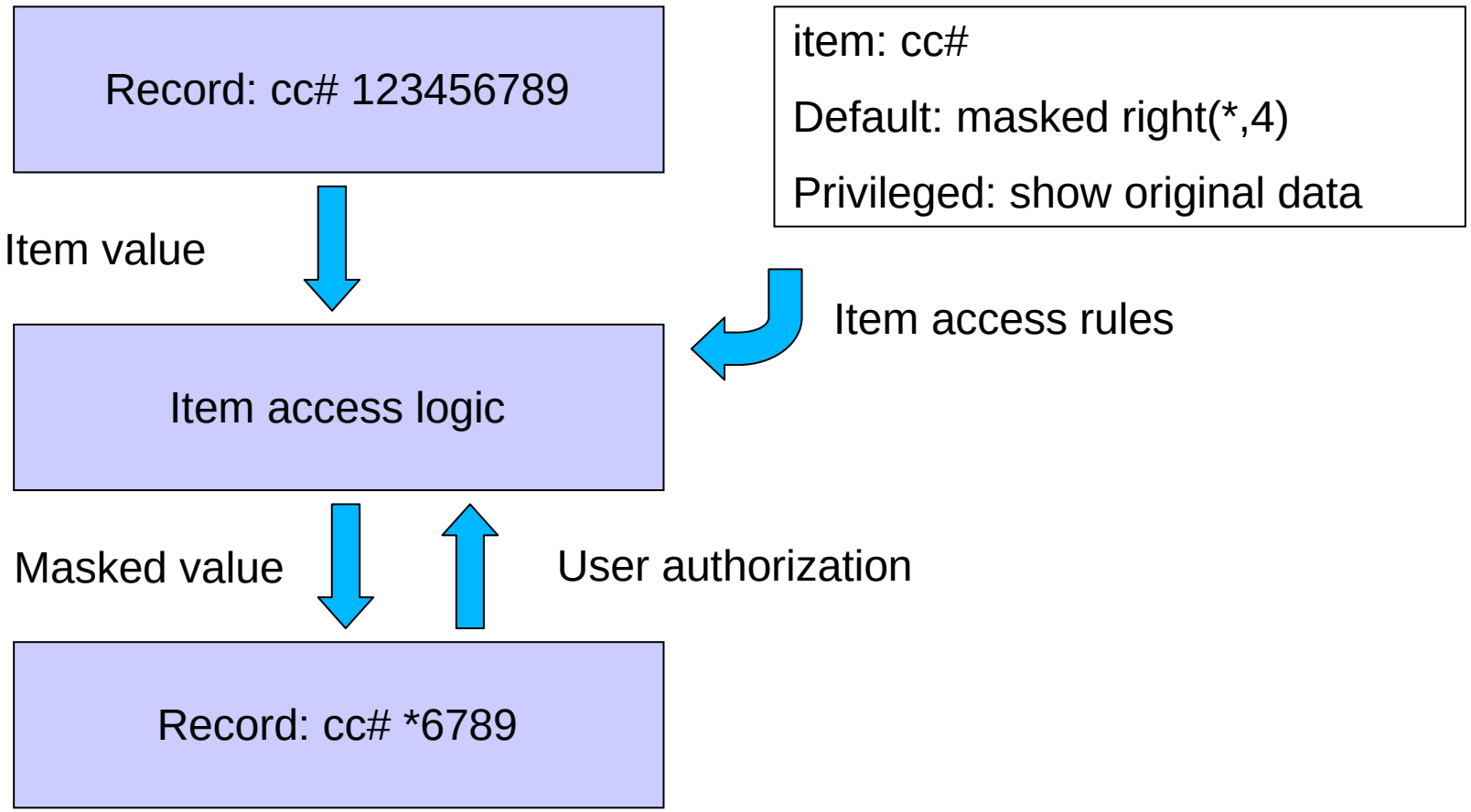
New functionality

- Major enhancements to help securing sensitive information
- Item Masking
  - Sensitive information may be masked or blanked upon retrieval, depending on the user authorization
- Database encryption
  - Sensitive information may be transparently encrypted in the database
- Allow enhancing the security of existing applications with no or minimal code changes
- Helps meeting PCI DSS requirements

Item masking

- Ensure sensitive information is not disclosed to unauthorized users
- Allow enhancing the security of existing applications with no or minimal code changes
- Option of hiding the field content on the fly or making only subset of information available
- Allows to specify defaults and exceptions for authorized users
- Extends on the Eloquence database security to define user access rights

Item masking



Item masking example

Database has a sensitive item called credit-card-no in several datasets

- Members of the billing department have full access
- Members of the "agents" group only see the rightmost 4 characters of the item but are able to change the item to a new value with DBUPDATE
- All other users only see the word “known", if the item contains a non-blank value, and should not be able to change the value

Item masking example

```
DATABASE "sampledb";

SET ITEM ACCESS ON credit-card-no
TO MASKED "set(known)", NOUPDATE;

SET ITEM ACCESS ON credit-card-no
TO MASKED "cover(*,0,4)" FOR "agents";

SET ITEM ACCESS ON credit-card-no
TO ALLOWED FOR "billing";
```

- default: show word “known” if credit-card-no is not empty, do not allow updates
- agents: show truncated credit-card-no
- billing: full access

dbutil syntax

```
UPGRADE DATABASE;


SET ITEM ACCESS ON ItemName [ IN SetName ]
TO MASKED [ MaskFunction ] [,NOUPDATE]
[ FOR GroupList ] ;


SET ITEM ACCESS ON ItemName [ IN SetName ]
TO ALLOWED [,NOUPDATE] [ FOR GroupList ] ;


UNSET ITEM ACCESS ON ItemName [ IN SetName ]
[ FOR { GroupList | ALL } ] ;
```

Item masking

- Database may impose rules for accessing sensitive information
  - Default access rules for sensitive data may be defined
  - Allow overriding defaults for authorized users
- Sensitive information may be masked or blanked upon retrieval, depending on the user authorization
  - Data is blanked (empty)
  - Data is masked (returned only partially)
  - Data may not be updated
- Extends on the Eloquence database security to define user access rights

Masking functions

Masking functions may be specified for string items. If no masking function is defined the content is blanked (or zero).

- **cover**(x,m,n) keeps the leading m and trailing n characters of the item contents and replaces each character in between with character x.
- **part**(x,m,n) keeps the leading m and trailing n characters of the item content and replaces the characters in between with a single character x.
- **left**(x,n) keeps the leading or all but the trailing part of the item content. The other content is replaced with a single character *x.*
- **right**(x,n) keeps the trailing or all but the leading part of the item content. The other content is replaced with a single character *x.*
- **set**(text) shows the string "text" if the item content is non-blank.

Masking functions: cover

cover(*x*,*m*,*n*) keeps the leading *m* and trailing *n* characters of the item contents visible and replaces each character in between with character *x*.

For example:

cover(X,1,2) on 123456 results in 1XXX56

- The item is considered left-justified and trailing blanks are ignored.

Masking functions: part

part(*x*,*m*,*n*) keeps the leading *m* and trailing *n* characters of the item contents visible and replaces the characters in between with a single character *x*.

For example:

part(*,1,2) on 123456 results in 1*56

- The item is considered left-justified and trailing blanks are ignored.

Masking functions: left

left(*x*, *n*) keeps the leading part or all but the trailing part of the item content.

- If *n* is positive, the *n* leftmost characters are kept.
- If *n* is negative, all but the trailing *n* characters are kept.
- The other content is replaced with the single character *x*.

For example:

left(*,2) on 123456 results in 12*
left(*,-2) on 123456 results in 1234*

- The item is considered left-justified and trailing blanks are ignored.

Masking functions: right

right(*x*, *n*) keeps the trailing part or all but the leading part of the item content.

- If *n* is positive, the *n* rightmost characters are kept.
- If *n* is negative, all but the leading *n* characters are kept.
- The other content is replaced with the single character *x*.

For example:

right(*,2) on 123456 results in *56
right(*,-2) on 123456 results in *3456

- The item is considered left-justified and trailing blanks are ignored.

Masking functions: set

set(*text*) shows the string "text" if the item content is non-blank.

For example:

set(PRESENT) on 123456 results in PRESENT
set(PRESENT) on an empty field results in an empty field

- The text is truncated as necessary if it exceeds the item size.

Database catalog

- The item access rules configured with dbutil script commands are maintained in the catalog table "sysitemproperty" for the respective database.
- They may be reviewed with dbdumpcat as needed (e.g. to help troubleshooting).

Database catalog

```
$ dbdumpcat -t 12 sampledb
-------------------------------------------------------
#12 sysitemproperty (5 entries)
-------------------------------------------------------
|gid |tableid|colid|type|flags   |data                 |
-------------------------------------------------------
|0   |0      |74   |1   |06000000|set(available)       |
|3   |0      |74   |1   |02000000|cover(X,0,4)         |
|4   |0      |74   |1   |00000000|                     |
|5   |101    |65   |1   |00000000|                     |
|6   |101    |65   |1   |02000000|                     |
-------------------------------------------------------
```

- The catalog tables "sysgroup", "systables" and "syscolumns" contain additional details on the values of the "gid", "tableid", and "colid" columns, for example the textual names of the respective entities
- Note that gid=0 denotes access rules not specific to a group and that tableid=0 denotes access rules not specific to a data set

Potential pitfalls

- Updating MASKED items
  - DBUPDATE has special logic to allow updating masked items
  - The masked value returned is silently ignored
  - A sequence of DBDELETE/DBPUT may corrupt masked items

- Users in multiple groups
  - If a user is member of more than one security group, the most restrictive access rule will apply, unless there is an "allow" rule that overrides the other access rules.
  - access rules sorted by precedence:
    ALLOWED > MASKED,NOUPDATE > MASKED > NOUPDATE
  - data set specific rules override unspecific rules

Potential pitfalls

- Using masked items for search and key items
  - DBGET mode 7 and DBFIND behave "as usual" when passing the (unmasked) item value as search argument
  - The application may compare the returned (masked) value

- Downgrading Eloquence
  - Item masking is ignored by older Eloquence versions
  - Remove item masks before using a previous Eloquence version

Database encryption

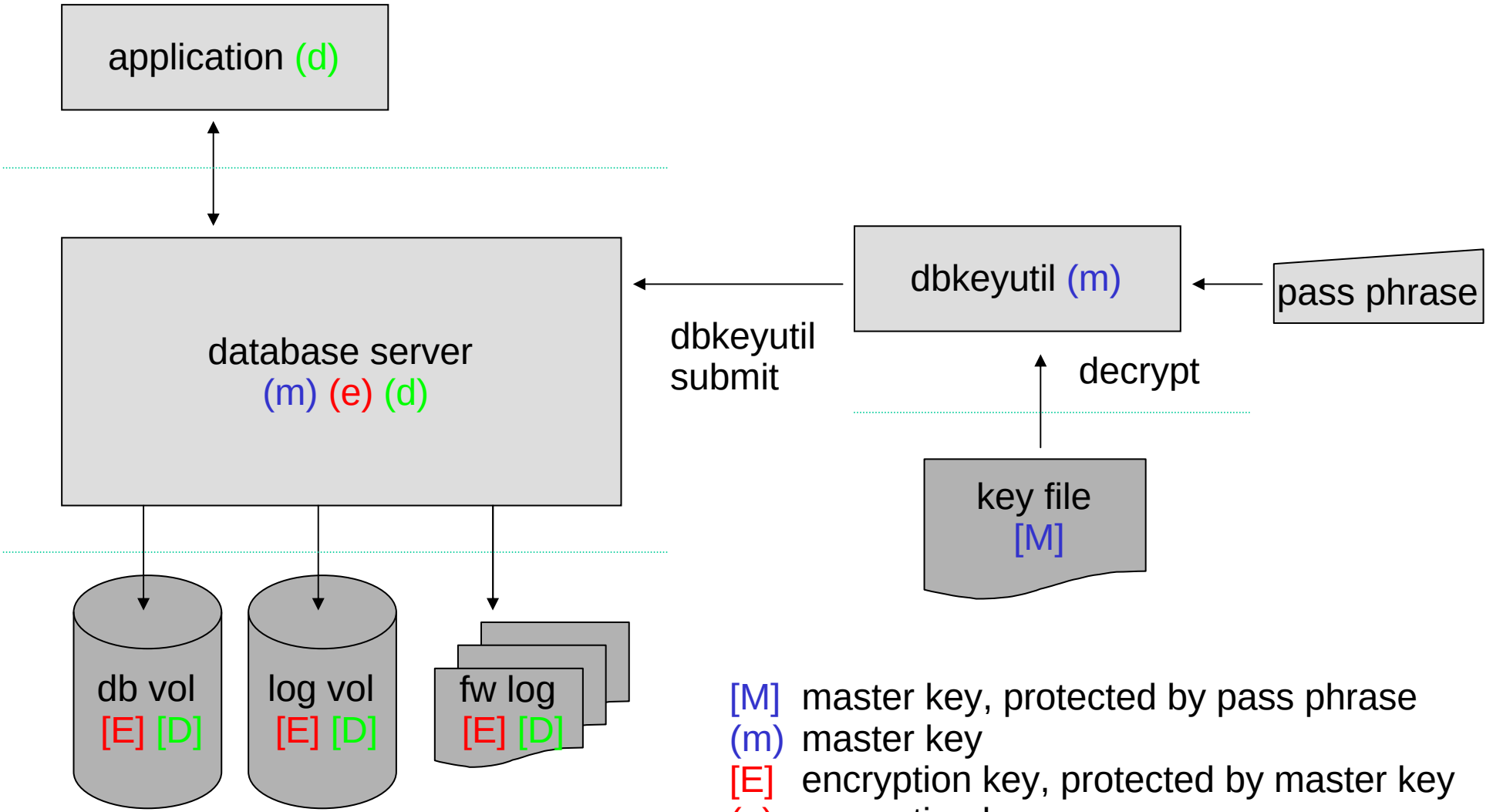
- The database encryption feature allows to designate dataset fields with sensitive contents for encrypted storage
- Helps to protect sensitive information in database volume files, forward logs, as well as dbstore files and backups against unauthorized access

- Using database encryption does not protect against weak database passwords or incorrect database security settings

Database encryption

- Data are encrypted using the standard AES algorithm
- Multiple data encryption keys supported
- Encryption keys may be updated periodically with no downtime
- Data encryption keys are secured by a separate “master key” that is maintained outside the database volume
- Master keys are protected by user pass phrase
- Supports “four-eyes principle” to enable access to encrypted data

The big picture



[M] master key, protected by pass phrase
(m) master key
[E] encryption key, protected by master key
(e) encryption key
[D] data, protected by encryption key
(d) data

Database encryption & keys

- The database server maintains a list of one or more data encryption keys per database
  - data encryption keys are used to encrypt the actual data
  - data encryption keys are kept in the syskey catalog table
- Each dataset entry containing encrypted items uses one data encryption key to protect the item contents
- New or updated entries always use the most recently created data encryption key
- The data encryption keys are stored (in encrypted form) inside the database volume files, protected by so-called "master keys"

Master keys

- Master keys are created outside the database server and stored in special text files, protected by individual pass phrases
- Master keys must be submitted to the database server to decrypt the data encryption keys to access encrypted dataset fields
- Master keys must be submitted after each start of the database server
- Without the master keys, the database server is unable to read or write encrypted items
  - A DBOPEN of an encrypted database returns status -812:0 if a master key is unavailable
  - A DBOPEN with mode 8 succeeds but encrypted items are unavailable (blanked/zeroed)

Master keys and utilities

- Any utility which accesses encrypted information needs the master keys to process encrypted fields

- Utilities like dbrecover or dbrepl do not need to know the master keys, as their processing does not involve "unpacking" of encrypted item content

Multi-part master keys

- Master keys may optionally be created as multi-part keys.
- With multi-part keys, all parts have to be submitted to the database server to be active.
- Using multi-part keys allows authorization schemes where keys are spread across independent key owners and all the involved parties are required for successful key submission.

Database catalog

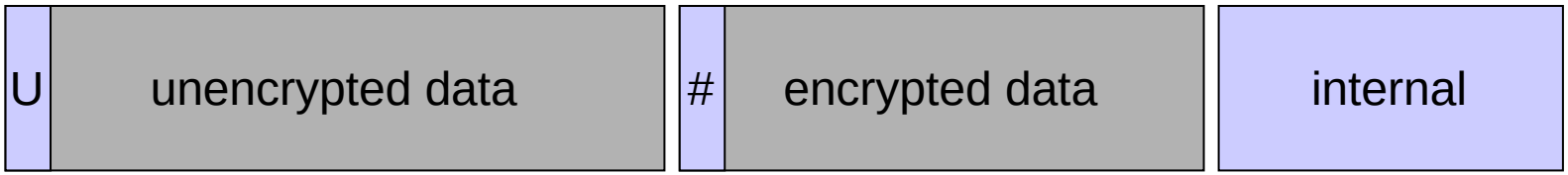
```
$ dbdumpcat -t 13 mydb
------------------------------------------------------------------------------
#13 syskey (1 entries)
------------------------------------------------------------------------------
|kid |tableid|type|tskey      |key              |kcheck          |mcheck         |
------------------------------------------------------------------------------
|1   |0      |2   |2009-11-25 |16:4b50922ca12f |16:e6da66d666f1 |16:9f452bd41b7d|
------------------------------------------------------------------------------
```

- The "kid" column shows the key ID
- The "tskey" column shows the data encryption key's creation timestamp (kept as seconds since 1970).
- The "mcheck" checksum shows the leading part of associated master key's checksum, which may be used to identify the respective master key in the keyfile.

Internal record layout

Unencrypted record

Encrypted record



- Encrypted fields are grouped for efficiency reasons
- Each record identifies the encryption key required to decrypt the data

Database encryption example

eloquence

Step 1: generate master key with dbkeyutil

- Before the database server can create and store any data encryption keys, it needs to be provided with a master key
- dbkeyutil is used to create a master key

```
$ dbkeyutil keygen mykey
Enter passphrase for mykey:
Confirm passphrase:
```

- The master key is generated, protected using the passphrase and stored in the local text file eqdb.key by default.

Database encryption example

Step 2: submit master key to the database server with dbkeyutil

- The dbkeyutil submit command is used to submit the master key to the running database server process.
- The utility prompts for the passphrase required to access the key in your keyring file (eqdb.key by default).
- It will require dba or operator privileges on the database server side.

```
$ dbkeyutil -u dba submit mykey
Enter passphrase for mykey:
Passphrase is valid
Master key submitted  successfully
```

Database encryption example

- The dbkeyutil status command may be used to review the list of active master keys on the database server

```
$ dbkeyutil status
idx master key checksum                stat type     ts
---- ------------------------------- ---- -------- -------------------
1   9f452bd41b7d669d1e2da1a4fdea8522  ACTV AES 128  2009-11-25 15:23:50
```

Database encryption example

Step 3: use dbutil to create data encryption keys

- The database server needs to have the internal data encryption keys for the respective databases
- Use dbutil to trigger the creation of data encryption keys.
- These keys are stored inside the database volume files, protected by a master key.
- You reference the desired master key in your dbutil command by specifying its checksum

```
database "mydb";
create encryption key using master key
"9f452bd41b7d669d1e2da1a4fdea8522";
```

Database encryption example

Step 4: use dbutil to configure dataset fields as encrypted

```
$ dbutil -v –
Processing script ...
database "mydb";
change set "invoices" set encrypted;
change item "order-date" in "orders" set encrypted;
change item "price" set encrypted;
exit;
Checking database consistency ...
Consistency check completed successfully

Database restructure analysis:
 PRODUCTS
  * Record reorganized due to encryption change
 INVOICES
  * Record reorganized due to encryption change
 ORDERS
  * Record reorganized due to encryption change
Data restructure process required.

Uploading modified schema ...
Restructuring database ...
done
```

dbutil script syntax

```
UPGRADE DATABASE;


CREATE ENCRYPTION KEY [ ("type") ]
USING [MASTER KEY] "master key";


CHANGE ITEM item-name [IN set-name] SET ENCRYPTED;
CHANGE ITEM item-name [IN set-name] UNSET ENCRYPTED;


CHANGE SET set-name SET ENCRYPTED;
CHANGE SET set-name UNSET ENCRYPTED;


CHANGE ALL ENCRYPTION KEYS
USING [MASTER KEY] "master key";

DELETE ALL ENCRYPTION KEYS;
```

dbkeyutil utility

- The dbkeyutil utility is used to create, maintain and upload database master keys
- A master key is used by the Eloquence database to encrypt/decrypt the actual data encryption keys
- Without the appropriate master key the database server is unable to access encrypted content.
- The master keys are maintained outside the database server volume files
- Master keys are saved in encrypted form in a text file that may hold multiple keys
- A master key may be created as a partial key where all parts must be submitted to the server separately to enable access to encrypted content

dbkeyutil utility

dbkeyutil supports the following commands

- keygen - create new master key
- chpass - change pass phrase
- check - test master keys
- submit - submit master key to server process
- revoke - revoke use of master key
- status - check status of master keys of server process

dbkeyutil example

- Creating a new master key “test”

```
$ dbkeyutil keygen test
Enter passphrase for test:
Confirm passphrase:
```

- Resulting key file

```
[test]
cksum = 19719711117059b5416af25a09ad8bb7
keyt = AES
key = dfeb22349e7b4ad284a18083c9620cc9
cipher = hmac-sha1:des-ede3-cbc:b250b72b0bc60461:1000
comment = AES-128 key, created 2010-02-01 15:57:24 by mike@dl385
```

- Submitting master key “test” to database server

```
$ dbkeyutil -u dba submit test
Enter passphrase for test:
Passphrase is valid Master key submitted successfully
```

dbctl dbkeyupdate

- The dbkeyupdate command in dbctl is used to retire old data encryption keys from a database
- It triggers a scan for the specified database covering all datasets and indexes using encrypted items
- It searches for data encrypted with an encryption key older than a given key ID or key creation date.
- If necessary it re-encrypts any entries found (using the currently active key)
- It finally discards the old and now unused data encryption keys from the catalog

dbctl dbkeyupdate

- dbkeyupdate is used to update data encryption keys (“rolling key update”)

For example:

- A new data encryption key is created every 6 months
- Data encryption keys older than a year are retired
- Eloquence only uses the most recent data encryption key when adding or updating data
- dbkeyupdate is then used to re-encrypt all data that was not updated and still depends on an older data encryption key

dbctl dbkeyupdate

- The scanning and encryption process may take quite some time and use considerable CPU or I/O resources
- dbkeyupdate provides "throttling" parameters for limiting its resource consumption and impact on concurrent users
- dbkeyupdate maintains internal status information in the database server.
- dbkeyupdate may be interrupted and resumed, as long as the server process is not restarted

dbctl dbkeyupdate utility

```
$ dbctl help dbkeyupdate
usage: dbkeyupdate [/v /speed pct /delay count ms]
       database {date|keyid}
```

- Specify the database name and a data encryption key ID or key creation date
- The dbkeyutil processing will then retire any older data encryption keys

dbkeyupdate example

- The following example uses dbctl dbkeyupdate to retire all encryption keys older than key id 3
- dbdumpcat on catalog table syskey is used to look up the most current key id
- A throttling with /delay to pause for 1000 msec every 10 entries is used
- Results in an upper bound of 10 entries per second to be scanned or re-encrypted.

dbkeyupdate example


```
$ dbdumpcat -t 13 toydb

-------------------------------------------------------------------------------
#13 syskey (2 entries)
-------------------------------------------------------------------------------
|kid |tableid |type |tskey      |key              |kcheck          |mcheck          |
-------------------------------------------------------------------------------
|2   |0       |2    |2010-03-24 |16:94d3e048fa5d |16:327f9fc27527 |16:b53bff99afbb |
|3   |0       |2    |2010-03-24 |16:ba0a59d86bf9 |16:a61551ecd9eb |16:8cdf7f0818f2 |
-------------------------------------------------------------------------------
```

dbkeyupdate example

```
$ dbctl -u dba dbkeyupdate /delay 10 1000 toydb 3

P0: server: Processing index ORDER-DETAILS.ORDER-IX
P0: server: Completed index ORDER-DETAILS.ORDER-IX - updated 1 records (0 already
current)
P0: server: Processing data set ORDER-DETAILS
P0: server: Completed data set ORDER-DETAILS - updated 15 records (0 already current)
P0: server: Processing primary index ORDER-MASTER.ORDER-NO
P0: server: Completed primary index ORDER-MASTER.ORDER-NO - updated 1 records (0
already current)
P0: server: Processing data set ORDER-MASTER
P0: server: Completed data set ORDER-MASTER - updated 15 records (0 already current)
P0: server: Processing primary index PRODUCTS.PRODUCT-NO
P0: server: Completed primary index PRODUCTS.PRODUCT-NO - updated 1 records (0
already current)
P0: server: Processing data set PRODUCTS
P0: server: Completed data set PRODUCTS - updated 10 records (1 already current)
dbkeyupdate: Database "toydb" successfully updated.
```

dbkeyupdate example


```
$ dbdumpcat -t 13 toydb
-------------------------------------------------------------------------------
#13 syskey (2 entries)
-------------------------------------------------------------------------------
|kid |tableid |type |tskey      |key              |kcheck           |mcheck          |
-------------------------------------------------------------------------------
|3   |0       |2    |2010-03-24 |16:ba0a59d86bf9 |16:a61551ecd9eb |16:8cdf7f0818f2 |
-------------------------------------------------------------------------------
```

Potential pitfalls

- Loss of the master keys or their pass phrases will result in the irrecoverable loss of the associated encrypted database (and forward log) contents
- Enabling encryption requires additional CPU resources, depending on the amount of encrypted items and the frequency records holding encrypted data are accessed
- Database server requires submitting the master key(s) after each restart
- Encrypted databases cannot be accessed by previous Eloquence versions

Password change timestamp

Database server maintains timestamp on last password change

- Timestamp is updated on account creation and when password is changed
- Displayed by dbutil in the user properties dialog
- Displayed by dbcumpcat

Operator user property

The "operator" user property may be used to allow some operational tasks previously limited to dba.

- Allows to be more restrictive with dba accounts
- Use in batch jobs
- Separation of roles
- Operator may not change data or permissions

Operator user property

Operator may

- dbctl backup – on-line backup
- dbctl forwardlog restart – New log generation
- dbctl shutdown - shutdown server process
- dbctl cancelthread / killthread – abort db session
- dbctl logfile – change server message log
- dbctl replication stop – abort replication on slave server
- dbctl statfile / session stat file – change statfile
- dbctl encryption revoke key - revoke encryption key
- dbctl dbkeyupdate – update encryption keys
- dbctl dbrestore /ALL – restore database
- purge a database

Refined dba privileges

- The server was enhanced to implicitly grant a DBA user administrative capabilities on all databases

- With previous Eloquence versions only the “creator” had administrative access
- Additional administrative users had to be added explicitly

dbctl list filters

- The dbctl list command was enhanced to support filter expressions
- A filter expression may be specified and the /count option may be used to only obtain the number of matching entries

For example:

```
$ dbctl list session "pname=*query3k*“
$ dbctl list lock /count "status=blocked"
```

DBINFO mode 114

- DBINFO mode 114 is similar to DBINFO mode 104 but returns field specific status information rather than item numbers
  - field is stored on disk in encrypted format.
  - encryption key for the database is not available. If this affects actual record, the field is blanked (if a string item) or zeroed when read.
  - item mask exists for this item, even if it does not apply for the current user
  - item mask affects information in this field (eg. information is truncated or blanked)
- May be used by application to identify sensitive information
- DBINFO mode 114 is available in both the image3k and the native client library

dbutil syntax enhancements

- CHANGE PATH may be used to change an existing path definition in a detail set
- CHANGE SET TYPE may be used to change set type between automatic and manual master
- Added support to order item list

Database utility changes

- dbutil has been enhanced to support item masking and item encryption
- dbctl has been enhanced for encryption status and key management
- The dbkeyutil utility was added to support item encryption.
- dbbexp, dbcfix, and fwaudit have been enhanced for item encryption
- dbutil and dbdumpcat were enhanced for password change timestamp
- Added a warning message to dbexport if data is masked or not available due to a missing master key

More information

Detailed information is available on the Eloquence web site

http://eloquence.marxmeier.com

Get in contact

info@marxmeier.com