**Troubleshooting a DB2 Crash with Entropy Issues**

**By** [Greg Sorensen](https://community.ibm.com/community/user/people/greg-sorensen)

**Introduction**

The strength of cryptographic keys, essential for robust data security, hinges on high entropy. Within a DB2 database environment, both database and network connections commonly utilize SSL/TLS for secure communication. Nevertheless, a degradation of entropy at the operating system layer can critically impair GSKit, resulting in disruptions to network connectivity and potentially causing a DB2 crash. The dynamics of this issue were examined by a team of DBAs, system admins and program developers. This blog post will detail the process of identifying the root causes the team found, such as low entropy issues and outline the diagnostic approaches and effective resolution strategies for scenarios where a DB2 crash occurred due to a GSKit malfunction.

**Problem Description**

In a very rare case, a DB2 system crash halted with the following message.

<Date> Instance:<instance> Node:000

PID:44585(db2agent (<DB>) 0) TID:2768234240 Appid:<IP>.37370.270505022405

base sys utilities sqleDoForceDBShutdownFODC Probe:10 Database:<DB>

ADM14001C An unexpected and critical error has occurred: "ForceDBShutdown".

The instance may have been shutdown as a result. "Automatic" FODC (First

Occurrence Data Capture) has been invoked and diagnostic information has been

recorded in directory

"/<dbpath/sqllib/db2dump/DIAG0000/FODC\_ForceDBShutdown\_<date>/".

Please look in this directory for detailed evidence about

what happened and contact IBM support if necessary to diagnose the problem.

<Date> Instance:<instance> Node:000

PID:44585(db2agent (<DB>) 0) TID:2768234240 Appid:<IP>.37370.270505022405

base sys utilities sqleDoForceDBShutdownFODC Probe:17395 Database:<DB>

ADM7518C The database manager has shut down the following database because a

severe error has occurred: "<DB\_name> ".

Superficially, this situation appeared quite alarming; however, it generally represents DB2's normal self-preservation protocols, necessitating further investigation. During the actual diagnosis of the failure, we encountered a highly unusual event: a complete collapse of the entire GSKit encryption library. The *db2diag.log* recorded the following message types:

<DATE>-240 I870310E1818 LEVEL: Error

PID : 44585 TID : 140149602117376 PROC : db2sysc 0

INSTANCE: <INSTNAME> NODE : 000 DB : <DB>

HOSTNAME: <FQDN>

EDUID : 48 EDUNAME: db2loggw (<DB>) 0

FUNCTION: DB2 Common, Cryptography, cryptEncryptInit, probe:20

MESSAGE : ECF=0x90000403=-1879047165=ECF\_CRYPT\_UNEXPECTED\_ERROR

Unexpected cryptographic error

DATA #1 : Hex integer, 8 bytes

0xFFFFFFFFFFFFFFFE

DATA #2 : String, 55 bytes

error:FFFFFFFFFFFFFFFE:lib(255):func(4095):reason(4094)

CALLSTCK: (Static functions may not be resolved correctly, as they are resolved to the nearest symbol)

[0] 0x00007F7732FDD87B /db2\_src/db2plug/sqllib/lib64/libdb2osse.so.1 + 0x24087B

[1] 0x00007F7732FDD6B1 ossLog + 0xA1

[2] 0x00007F7734A39931 cryptLogICCError + 0x171

[3] 0x00007F7734A37790 cryptEncryptDecryptInit + 0x290

[4] 0x00007F7734A342AA cryptDecryptBuffer + 0x3A

[5] 0x00007F7738FF3866 \_Z23sqlexRedeemCipherTicketP17sqlexCipherTicketPhmP5sqlca + 0xF6

[6] 0x00007F773AAA64E6 \_Z31sqlpgEncryptUserDataInNLogPagesPvS\_tP17sqlexCipherTicketm + 0x1C6

[7] 0x00007F773AC68862 \_Z15sqlpWriteNPagesP9SQLP\_DBCBP9sqeBsuEduP9SQLP\_LECBPcmmb + 0x202

[8] 0x00007F773AC64EBF \_Z16sqlpgWriteToDiskP9SQLP\_DBCBP9SQLP\_LFPBmbbm + 0xD4F

[9] 0x00007F773AC63E36 \_Z13sqlpgPingPongP9SQLP\_DBCBP9SQLP\_LFPBmbm + 0x76

[10] 0x00007F773AC6BDCD \_ZN11sqpLoggwEdu8sqlpgwlpEmmPK9SQLP\_LSN8m + 0xF3D

[11] 0x00007F773AB956A3 \_ZN11sqpLoggwEdu13sqlpLoggwMainEv + 0x16A3

[12] 0x00007F773AE07C17 \_ZN11sqpLoggwEdu6RunEDUEv + 0x27

[13] 0x00007F773C52748E \_ZN9sqzEDUObj9EDUDriverEv + 0x1BE

[14] 0x00007F773AA3CD2A sqloEDUEntry + 0x57A

[15] 0x00007F7742694EA5 /lib64/libpthread.so.0 + 0x7EA5

[16] 0x00007F773200DB0D clone + 0x6D

<DATE>-240 I881028E519 LEVEL: Error

PID : 44585 TID : 140149602117376 PROC : db2sysc 0

INSTANCE: <INSTNAME> NODE : 000 DB : <DB>

HOSTNAME: <FQDN>

EDUID : 48 EDUNAME: db2loggw (<DB>) 0

FUNCTION: DB2 UDB, data protection services, sqlpEncrypt, probe:100

MESSAGE : ZRC=0x875C00CD=-2024013619=SQLEX\_UNEXPECTED\_SYSERR

"Unexpected System Error"

DATA #1 : String, 30 bytes

Error redeeming cipher ticket!

<DATE>-240 I880112E510 LEVEL: Severe

PID : 44585 TID : 140149602117376 PROC : db2sysc 0

INSTANCE: <INSTNAME> NODE : 000 DB : WEB

HOSTNAME: <FQDN>

EDUID : 48 EDUNAME: db2loggw (<DB>) 0

FUNCTION: DB2 UDB, bsu security, sqlexRedeemCipherTicket, probe:1519

MESSAGE : ZRC=0x875C00CD=-2024013619=SQLEX\_UNEXPECTED\_SYSERR

"Unexpected System Error"

<DATE>-240 I881548E19008 LEVEL: Error

PID : 44585 TID : 140149602117376 PROC : db2sysc 0

INSTANCE: <INSTNAME> NODE : 000 DB : <DB>

HOSTNAME: <FQDN>

EDUID : 48 EDUNAME: db2loggw (WEB) 0

FUNCTION: DB2 UDB, data protection services, sqlpgEncryptUserDataInNLogPages, probe:30

MESSAGE : Encountered error when encrypting log page.

DATA #1 : unsigned integer, 8 bytes

0

DATA #2 : SQLP\_LFPB, PD\_TYPE\_SQLP\_LFPB, 1 bytes

bytecount = 2542

firstIndex = 138

PHFlags = 0x0010

pageLso = 21194762240142 maps to 0000135F010789FD

CheckSum = 0x319467a1

Hex Dump of Log Data ------------------------------------

<Followed by a long set of HexDump values>

Upon analyzing this situation, it became clear that the encryption problems stemmed directly from the inability to encrypt the logs. This led to a DB2 crash, which invoked FODC, and brought all transactions to a halt. The necessary recovery procedure includes issuing a *db2kill* , performing an *ipcrm* to clear any lingering DB2 resources, and then executing a *db2start* to enable the database to undergo crash recovery.

“*Error redeeming cipher ticket and the associated ZRC=0x875C00CD”* indicates a problem with retrieving or validating the "cipher ticket". A cipher ticket is likely related to the key management and exchange process within the DB2 encryption mechanism.

The message “*Encountered error when encrypting log page from function sqlpgEncryptUserDataInNLogPages, probe:30”*, confirms the issue occurred while encrypting a log page.

Based on searches involving this we have all these likely sources.

**Key Management Issues**:

The core problem seems to be an inability to access or utilize the necessary encryption

keys or related credentials for encrypting log pages. According to IBM, DB2 relies on GSKit (IBM Global Security Kit)

for cryptographic requests, so problems with GSKit could prevent access to the required encryption keys.

**Keystore Accessibility/Corruption:**

There may have been issues accessing the keystore that stores the encryption keys for the DB2 instance. If the keystore was corrupt or unavailable, DB2 would not be able to retrieve the necessary key to encrypt the log page.

**Permission Issues:**

Improper permissions on the keystore or related files could have prevented DB2 from accessing them and subsequently encrypting the log page.

**GSKit Errors:**

The ZRC value 0x875C00CD indicates a "Unexpected System Error" which can be caused by various issues, including GSKit errors related to the encryption process.

**Low Entropy Environment:**

According to IBM, low entropy environments can lead to GSKit self-test failures, potentially impacting encryption operations.

**FIPS Compliance:**

If the DB2 instance is operating in FIPS-compliant mode, and there were issues related to FIPS-validated cryptography, this could have also contributed to the encryption failures.

**Diagnosis**

Our troubleshooting process involved a comprehensive examination of all encryption sources. Initial investigations readily dismissed issues concerning GSKit KDB file access, as its permissions and location had been stable for years. We also confirmed the integrity of the keystores for both DB2 encryption and SSL encryption by successfully viewing and verifying them. FIPS compliance mode was ruled out, validated by the absence of *STRICT\_FIPS* in the *DB2AUTH* variable and the lack of *fips=1* within */proc/cmdline* on the Linux operating system. The server did not exhibit signs of overload in terms of CPU or memory utilization. Network interfaces were fully functional, showing no indications of dropped packets or CRC errors. Nevertheless, monitoring data from Instana highlighted a sharp increase in *s3fs* CPU utilization immediately preceding the crash. Our Instana setup tracks critical processes such as DB2, s3fs, DB2 automation scripts, CDC activity, Tivoli Storage Manager backups, various Java processes, and the underlying OS, and it explicitly flagged s3fs for heavy CPU consumption during this period.

This essentially left us with a "Low Entropy Environment" issue.

To diagnose the issue, we meticulously reviewed DB2 audit logs for elevated transaction volumes and consulted */var/log/messages* for anomalies occurring concurrently. While a correlation with increased DB2 transactions from particular users was noted, it did not entirely exceed typical operational bounds. The messages file, conversely, provided more revealing insights. Enabling debug logging (*-o dbglevel=debug*) for *s3fs* (our cloud object storage filesystem client) exposed an unusually high number of HTTPS operations. Specifically, the *s3fs* process was generating numerous 'HTTP response code 404' entries, signifying that its underlying *curl* requests (for HTTPS GET and PUT operations) were failing to locate specific directories on the traversed network paths. Furthermore, we ascertained that rngd was not deployed to address potential low entropy conditions. Examination of */proc/sys/kernel/random/entropy\_avail* indicated erratic fluctuations in entropy levels, demonstrating insufficient replenishment to sustain demand.

We ran this command multiple times over a period of 10 minutes. A healthy system should show values in the thousands (preferably closer to the maximum pool size, which is typically

4096 bits on a standard system).

$ cat /proc/sys/kernel/random/entropy\_avail

Values below 200 bits are considered dangerously low and can cause processes to block, waiting for more entropy. Our values were smaller than 3200 bits and might also lead to issues like the system hanging while generating certificates.

We further used the rng-tools package provides a utility called rngtest that can perform tests on /dev/random to assess the quality of the randomness.

Install rngd with sudo yum install -y rng-tools (or equivalent for your distribution).

Then, run tests like:

# cat /dev/random | rngtest -c 100

This will perform 100 FIPS tests and report on the randomness of the data from /dev/random.

The results of these showed that FIPS testing failed randomly with the FIPS types with failure counts in the low single digits.

**Solution**

After evaluating all potential sources of encryption-related issues, we identified three key problems to resolve: first, the *rngd* service needed to be installed; second, the s3fs programs required recompilation and updating to improve stat cache utilization; and third, every directory in the cloud object storage bucket needed to be touched with an additional / appended.

The rngd service is easily configured. We use Red Hat 8.10 and installed the rng-tools package and then enabled it.

# yum install -y rng-tools

# systemctl enable rngd

# systemctl start rngd

# systemctl status rngd

● rngd.service - Hardware RNG Entropy Gatherer Daemon

Loaded: loaded (/usr/lib/systemd/system/rngd.service; enabled; vendor preset: enabled)

Active: active (running) since Mon ......

Main PID: 2451155 (rngd)

CGroup: /system.slice/rngd.service

└─2451155 /usr/sbin/rngd -f --rng-device=/dev/hwrng --fill-watermark=0 -x pkcs11 -x nist -x qrypt -D daemon:daemon

The compilation instructions for s3fs are located at <https://github.com/s3fs-fuse/s3fs-fuse/blob/master/COMPILATION.md>. Utilizing these guidelines, we updated our code to version v195 and subsequently applied the new binary. This update enabled the use of several new options, including *use\_path\_request\_style*, *enable\_noobj\_cache*, *stat\_cache\_expire=3600*, and *kernel\_cache*. The revised methods employed by this updated version modified the approach to utilizing target URLs for objects, thereby mitigating the occurrence of 404 errors and enhancing the efficiency of HTTPS communication for object storage.

Finally, we had to *mkdir* every directory on COS with the trailing / and non-slash.

$ cd <Cloud\_Object\_Base\_Mount\_Point>

$ find . -type d | while read dir; do mkdir $dir; mkdir $dir/; done

This ensured that all of the objects on the storage have both object types for curl to find. Cloud Object Storage is not a filesystem, and ensuring that both types exist help with the transactions used to POST and GET when storing data. This further reduces the 404 errors and the network activity needed when using an s3fs type storage device.

We then checked the entropy state after applying all of these with the same rngtest command and found all of the FIPS tests came back with 0 errors.

**Conclusion**

While our specific environment might be unique, the diagnostic approach is universally applicable. Any server experiencing similar problems requires a holistic examination of all encryption processes, both inside and outside DB2. Furthermore, a detailed review of major processes involving network communication is warranted, as they are often contributing factors. This encompasses, but is not limited to, analyzing network monitors, DB2 database encryption, DB2 remote transactions, system network backup utilities, NFS mounts, incoming network port scanners, system monitoring programs, and system log message agents.

The problem of low entropy can be exacerbated in virtualized environments, particularly on VMs that lack a dedicated hardware encryption module. Servers encountering such issues necessitate a comprehensive, system-wide analysis to identify the root causes. Nevertheless, installing the *rngd* service constitutes a valuable initial step. For Linux VMs operating on KVM/QEMU, the most widely adopted and recommended methodology involves the utilization of VirtIO RNG. Additionally, *haveged* serves as another daemon whose purpose is to replenish the kernel's entropy pool, thereby ensuring an adequate supply of randomness through */dev/random* and */dev/urandom*.

Keeping sufficient entropy available is critical for cryptographic operations and, consequently, vital for DB2 functionality in most production environments. DB2 utilizes SSL/TLS to secure both its database and network connections, making it highly dependent on a healthy entropy pool.

**About the Author**

Greg Sorensen is a DBA and Systems Admin for IBM Quoting with a Bachelors Degree in Electrical Engineering from UT Austin. He has worked primarily with AIX, Linux, DB2 and systems programming. He can be reached at gsorense@us.ibm.com.