Encrypting MySQL data at Google

Jonas Oreland

and

Jeremy Cole

bit.ly/google_innodb_encryption

Jonas Oreland

Software Engineer at Google
Has worked on/with MySQL since 2003
Has a current crush on Taylor Swift
Loves the German language and pretends to speak it fluently

Jeremy Cole
@jeremycole

Making MySQL Awesome at Google
Worked at MySQL 2000-2004
Contributor since 3.23
16 years in the MySQL community
Code, documentation, research, bug reports
Yahoo!, Proven Scaling, Gazillion, Twitter

Ceiling Cat

Reading your plaintext data since 2003
Grandmother (tabby) was owned by George Orwell
Worked for unnamed agencies for unknown governments
R.I.P. (2010)

The threat

Diverse set of threats to protect against

Access through network APIs (mysql client, etc.)

Access within from a running server (ptrace, memory dumping, etc.)

Lost or misplaced disk

Backups

Not all threats are feasible to mitigate

A dedicated attacker with root access to a running instance and unlimited time to attack it

An attacker with unlimited network access — we have to assume that the network and shared services are reasonably secure

The alternatives

Column encryption in the application

Encrypt individual column data from the application or ORM system

Direct access via SQL no longer possible

May not be feasible with many diverse users of applications

Completely incompatible with any 3rd party applications

Column encryption by middleware

Column encryption from a middleman — MyDiamo wraps InnoDB and MyISAM storage engines

Column encryption via a connection proxy — CryptDB uses a proxy in between client and server

Full system encryption / disk encryption

Block device / full disk encryption – dm-crypt encrypts the device on which the filesystem is built. While mounted, files are accessed as normal.

Our solution

Design goals and principles

Encrypt all user data that may touch the disk — InnoDB data, InnoDB logs, binary logs, temporary tables, temporary files

Make drive-by data exfiltration impossible

An attacker has to steal data *and* have access to internal key management systems

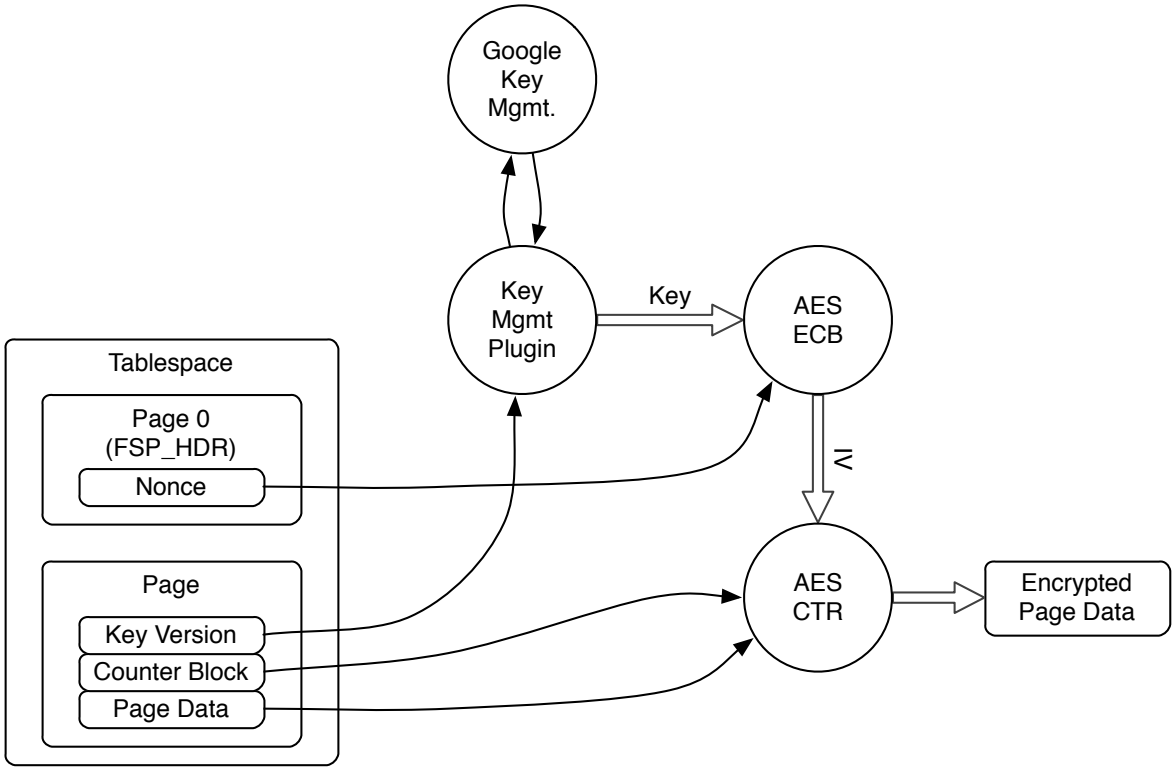
Technology and Terminology

Definitions

AES — Advanced Encryption Standard — a standard with a set of encryption primitives
IV — Initialization Vector — a starting point for an algorithm
Nonce — Random data used to randomize IV inputs
AES CTR — AES Counter Mode — block encryption, data stays the same size, input to the counter must be unique
AES GCM — AES Galois/Counter Mode — authenticated block encryption, data grows (new “tag” added)
Counter block — Input to AES CTR to guarantee unique/secure output of repeated or known patterns

InnoDB data and redo logs

InnoDB data and redo logs

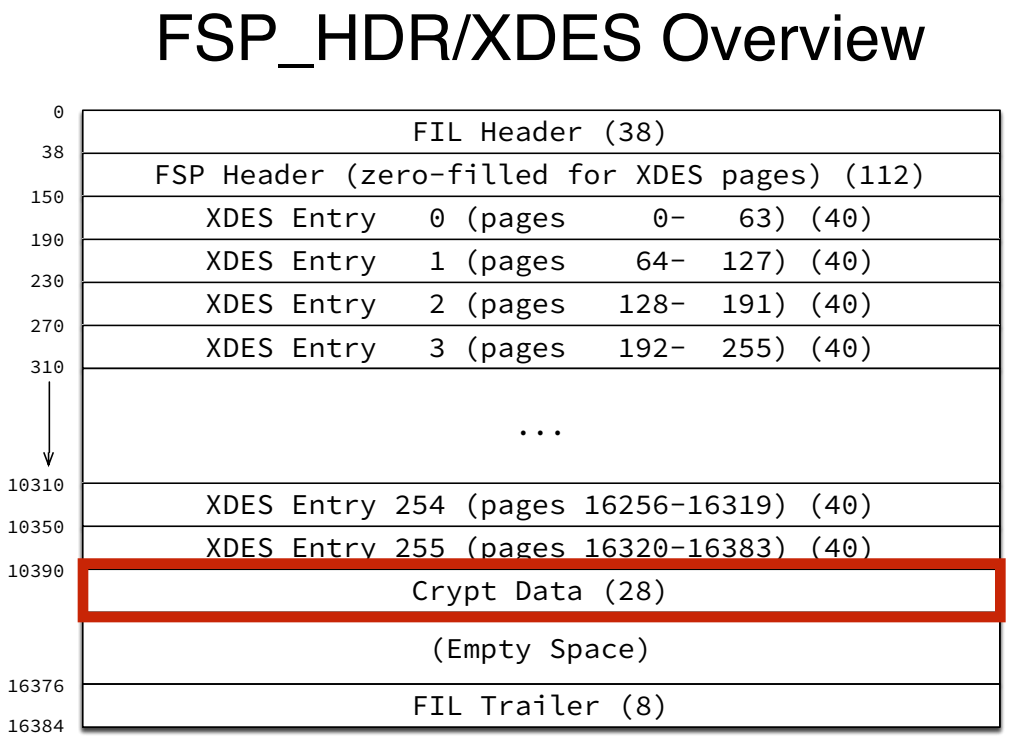


InnoDB tablespaces (table and index data)

- InnoDB organizes data into (default) 16 KiB pages
- All pages are encrypted except for page 0 (FSP_HDR: header and tablespace bookkeeping)
- Page 0 is augmented with Crypt Data, data need to perform encryption/decryption
- The page header of all other pages are augmented with key version
- The page header and trailer are not encrypted

InnoDB tablespaces (table and index data)

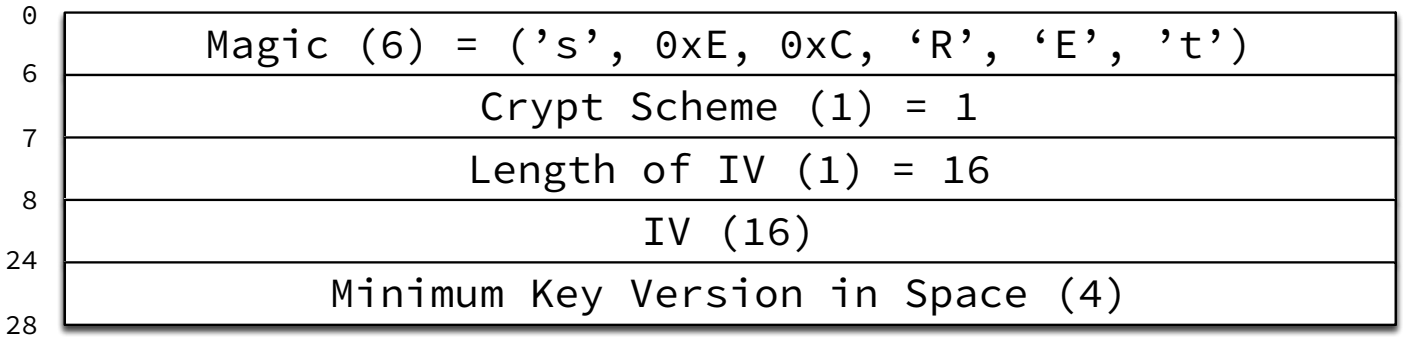
- PAGE 0 is not encrypted and is augmented with Crypt Data Header



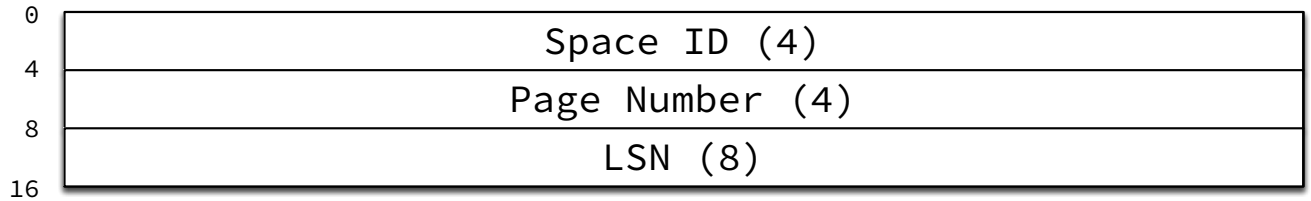
InnoDB tablespaces (table and index data)

- PAGE 0 is not encrypted and is augmented with Crypt Data Header

Crypt Data Header

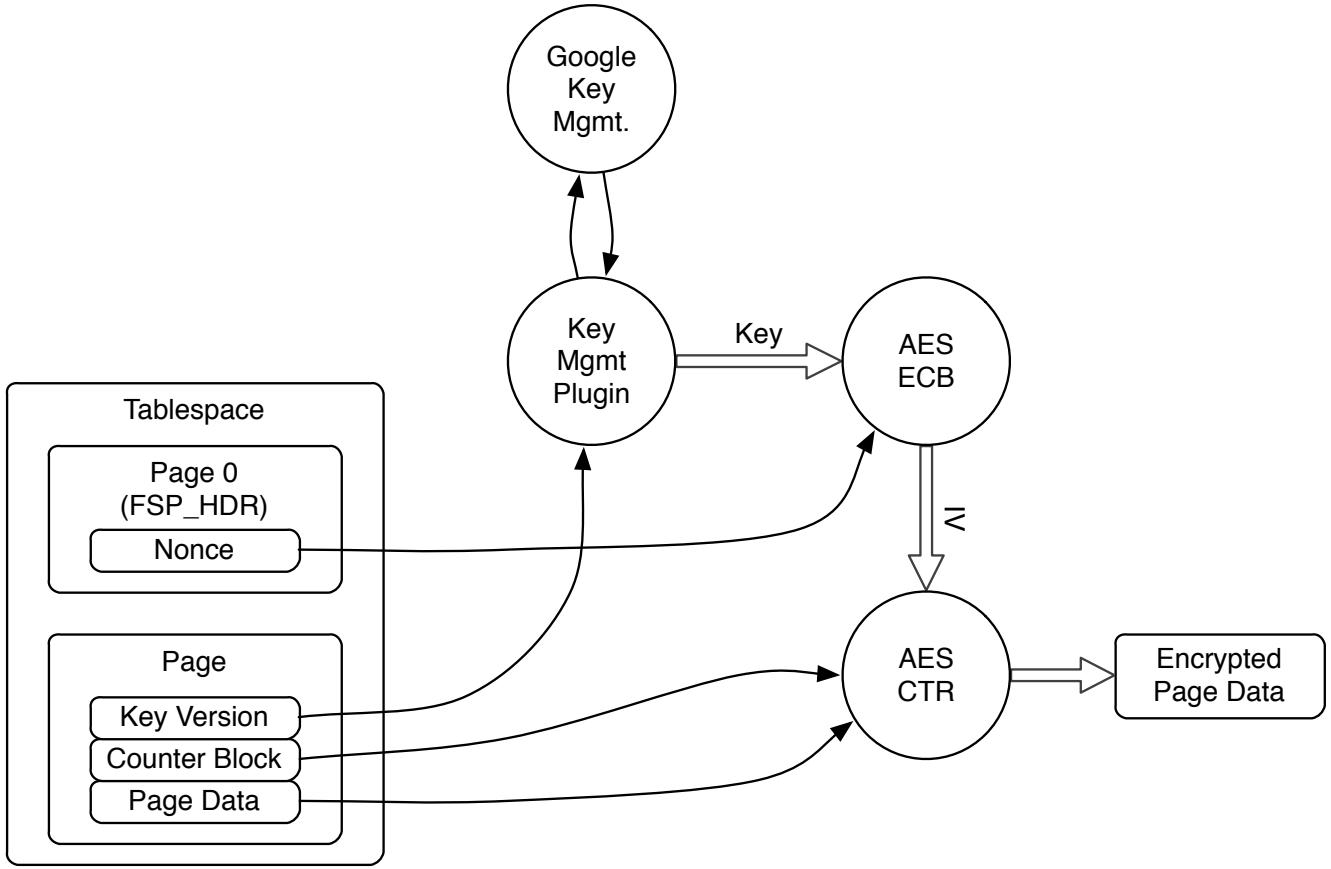


Page Counter



InnoDB tablespaces (table and index data)

InnoDB Space Encryption

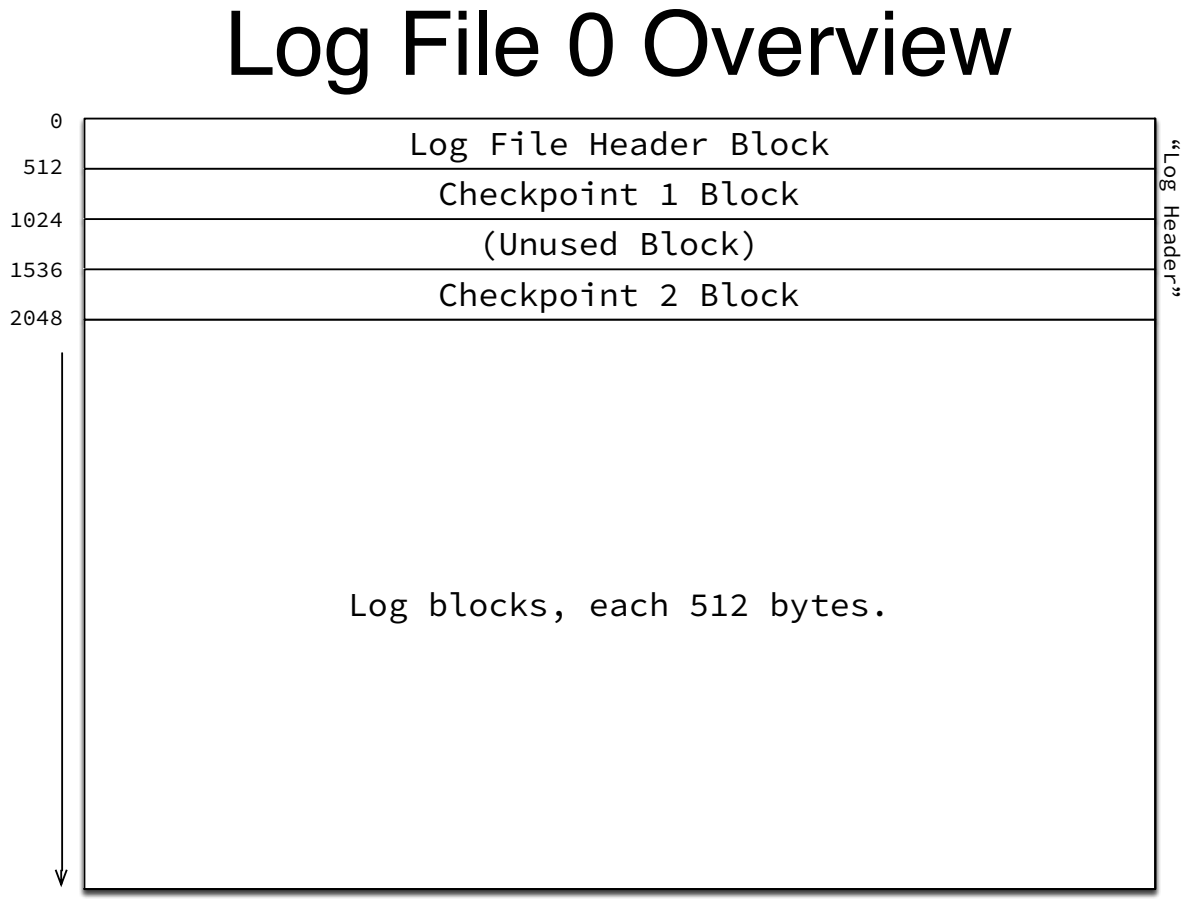


InnoDB redo logs (ib_logfileX)

- A redo log file is a series of log blocks
- The first 4 in logfile0 are not encrypted
- The checkpoint log blocks (2 & 4) are augmented with Crypt Data
- All other log blocks are encrypted individually
- Each log block contains a checkpoint number
- Each checkpoint number has a Crypt Data Entry

InnoDB redo logs (ib_logfileX)

- A redo log file is a series of log blocks
- The first 4 in logfile0 are not encrypted



InnoDB redo logs (ib_logfileX)

- The checkpoint log blocks (2 & 4) are augmented with Crypt Data

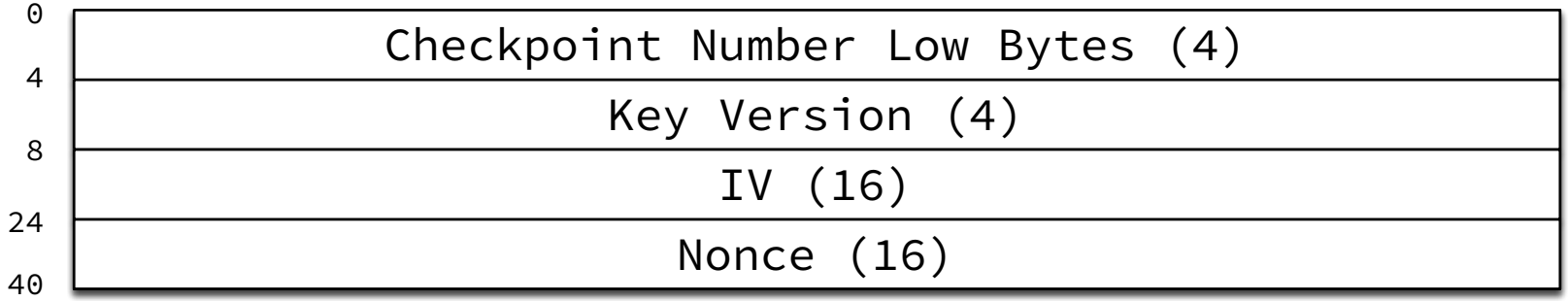
Log Checkpoint

0	Checkpoint Number (8)
8	Checkpoint LSN (8)
16	Checkpoint Offset Low Bytes (4)
20	Buffer Size (4)
28	Archived LSN (8)
36	Log Group Array (unused) (256)
292	Checksum 1 (4)
296	Checksum 2 (4)
300	FSP Free Limit (4)
304	FSP Magic Number (4)
308	Checkpoint Offset High Bytes (4)
312	Crypt Scheme (1) = 2
313	Number of Crypt Checkpoint Entries (1)
314	Crypt Checkpoint Entries (TBD)
512	

InnoDB redo logs (ib_logfileX)

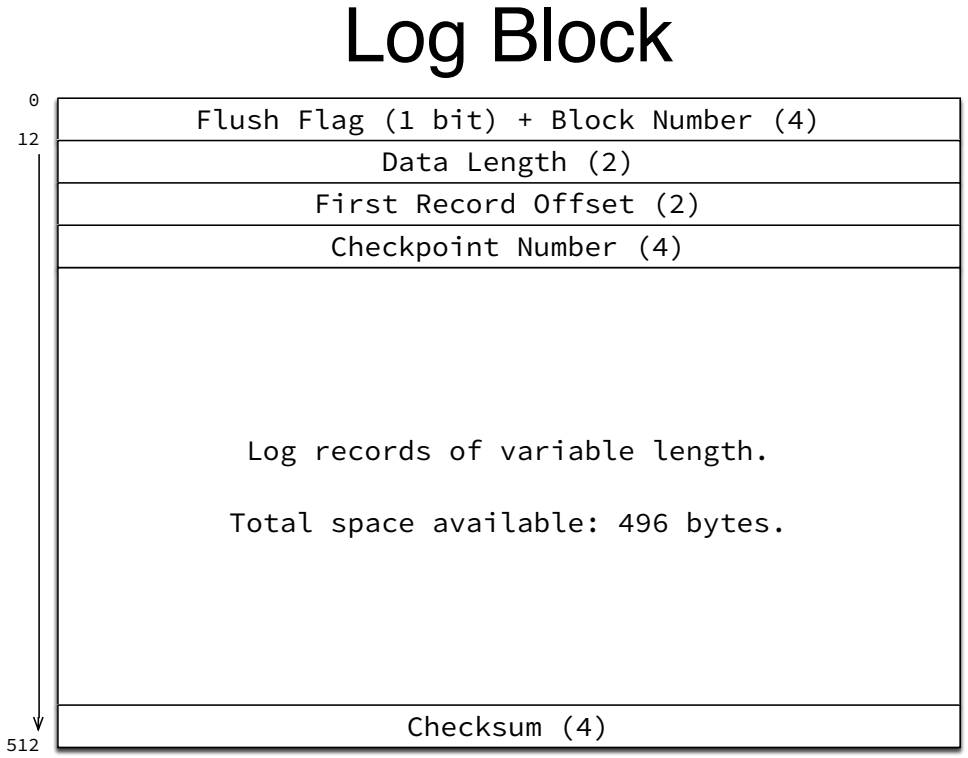
- Each checkpoint number has a Crypt Data Entry
- Each Log Checkpoint block stores information about the last 5 checkpoints

Crypt Checkpoint Entry



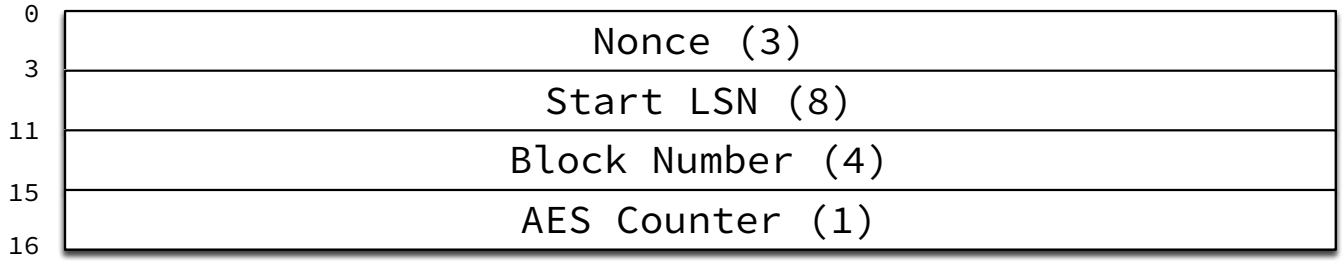
InnoDB redo logs (ib_logfileX)

- All other log blocks are encrypted individually
- Each log block contains a checkpoint number



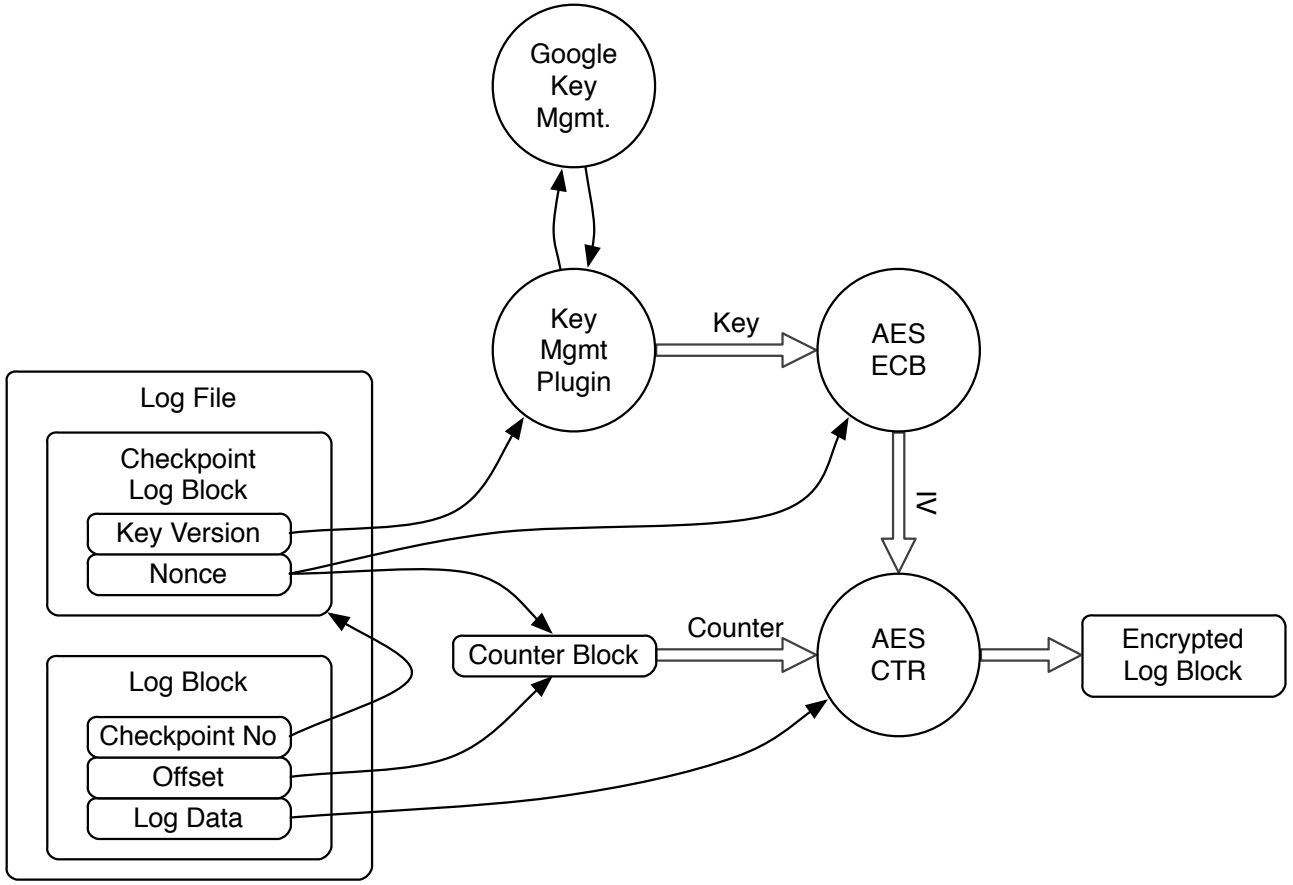
InnoDB redo logs (ib_logfileX)

Log Block Counter



InnoDB redo logs (ib_logfileX)

InnoDB Redo Encryption



InnoDB undo logs (record versions for MVCC)

Undo logs are stored in regular pages and are encrypted as well

Nothing special to make this work!

Key Management

An exercise for the reader...

We provide an example of the key management API

You have to write your own to be really useful/secure...

Google’s key management

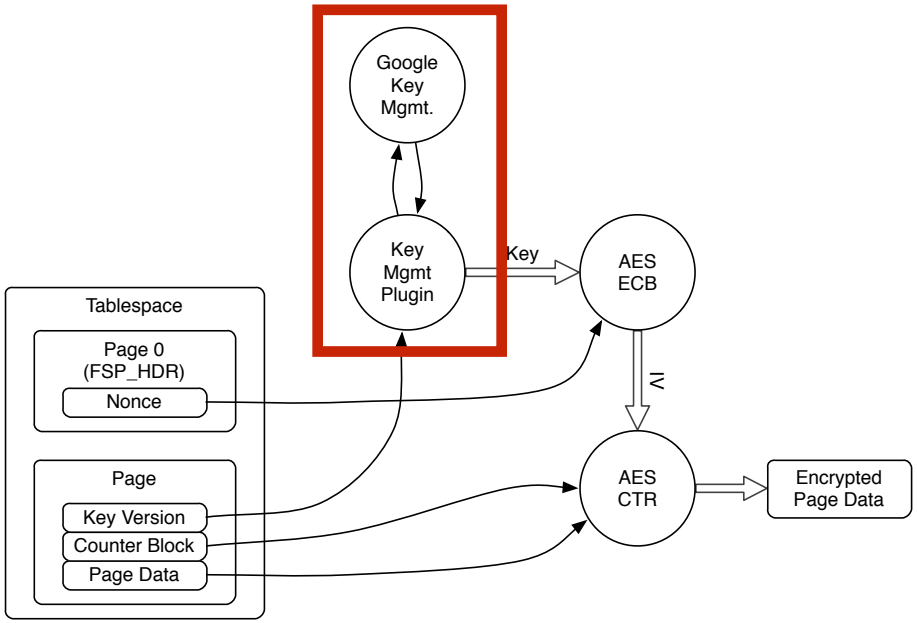
Google has a proprietary key management plugin (which is not in our public patch)

Keys are stored on other machines, fetched over the network (using Google-proprietary authenticated and encrypted RPCs)

Keys are cached in RAM in mysqld

Key management plugin interface

```
GetLatestKeyVersion()
GetCryptoKey(version, *key, key_size)
```



Key Rotation
(and scrubbing)

Goals

Set an upper bound on how long a key can be in use — each key has a lifespan after which it is no longer used

How is key rotation done

Redo logs — dummy log entries are written regularly to ensure that the log will be overwritten in a bounded time period

Temporary tables and files — always encrypted with the latest key and have a bounded lifetime

Binary logs and relay logs — encrypted using the latest key, log rotation ensures a bounded lifetime

InnoDB data

Each page has the key version it was encrypted with stored in the header
Each tablespace has the oldest (minimum) key version stored in the tablespace header
If the tablespace has any pages with a key older than the minimum acceptable key version, key rotation is started on the tablespace

Key rotation for InnoDB data

Ensure tablespace is marked as encrypted

Using N threads:

- Read a page
- Check f the key version is too old
- If yes, mark the page as dirty to cause a flush
- Flushed pages always use the newest key

After all pages are checked:

- Make sure all modified pages are flushed
- Update tablespace header in page 0 with new minimum key version
- Flush page 0

Cleaning up data to limit attack surface

InnoDB tends to collect “garbage”:

- Deleted rows
- Old versions of modified rows
- Redo log space which hasn’t been overwritten
- Undo log space which hasn’t been reused

Binary Logs

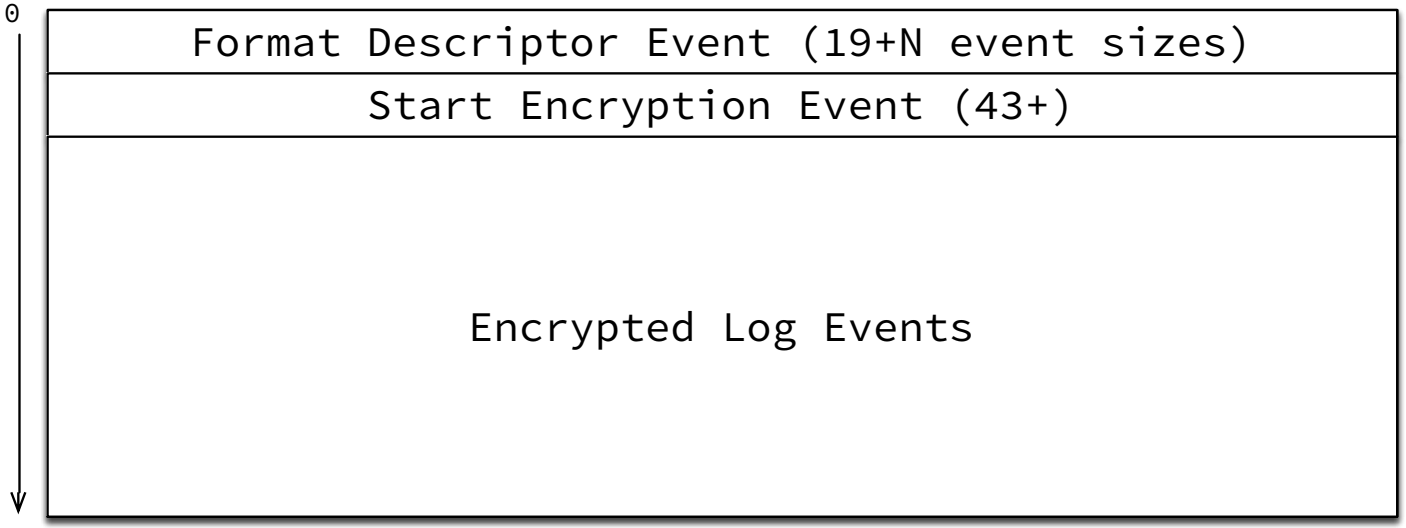
MySQL binary logs (for replication)

- New event type, Start_encryption, containing IV and key version
- Encryption begins for all events after Start_encryption
- Each event is augmented with 20 bytes to hold length and crypt tag
- Authenticated encryption AES-GCM

binary/relay logs

- Encryption begins for all events after Start_encryption

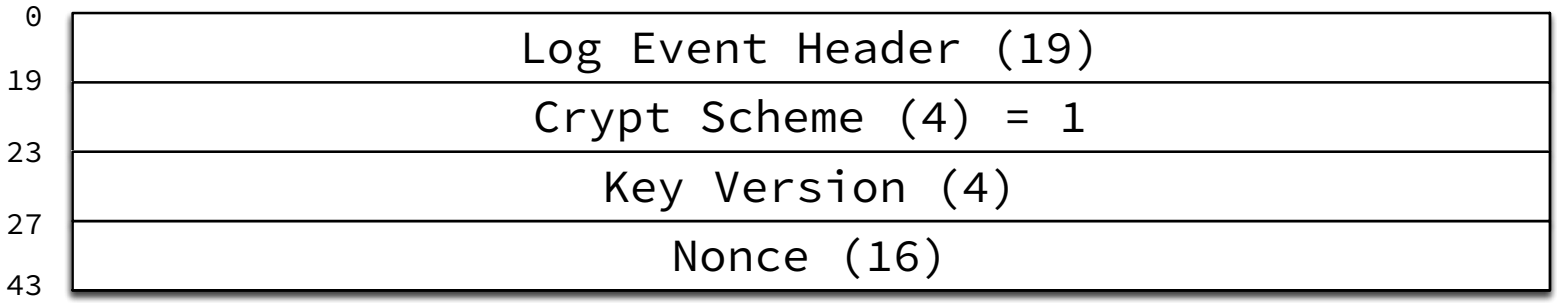
Binary Log Overview



binary/relay logs

- New event type, Start_encryption

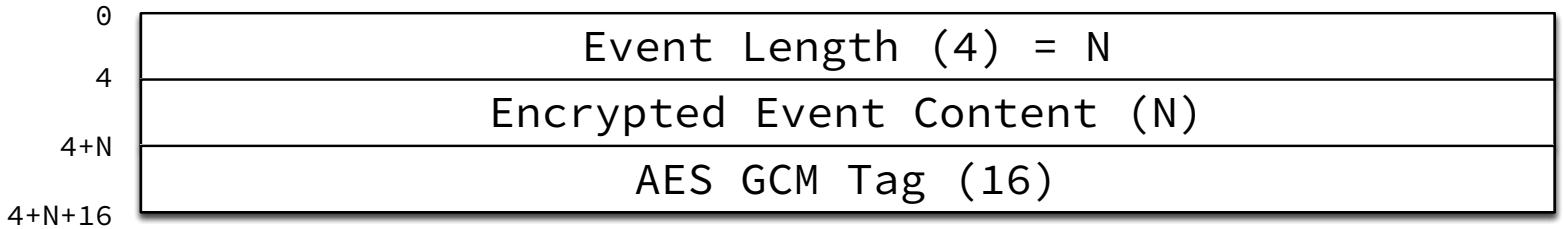
Start Encryption Event



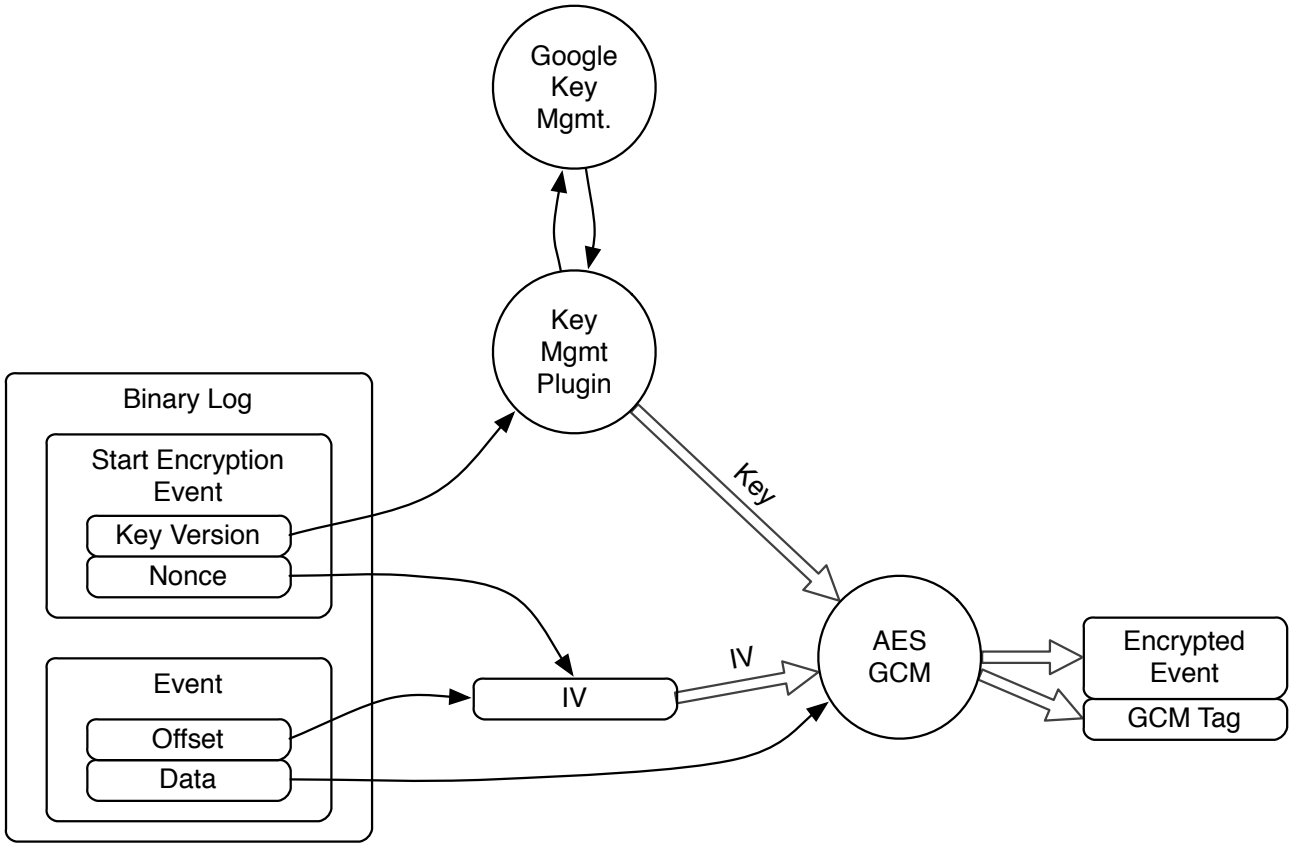
binary/relay logs

- Each event is augmented with 20 bytes to hold length and crypt tag

Encrypted Log Event



Binary Log Encryption



Temporary Tables (Aria)

Temporary tables during query execution (Aria)

- Aria supports many different file formats
- We encrypted only BLOCK_RECORD
- When encryption is enabled, the format is forced to BLOCK_RECORD
- When encryption is enabled, checksumming is always turned on
- We only use aria for temporary tables, so upgrade is not supported.
- Encryption is performed using AES_CTR

Temporary tables (Aria)

- First N-blocks of a data-file contains table meta data
- Table meta-data is augmented with Crypt Data Header
- Crypt Data Header contains crypt scheme and 20 bytes nonce (total 22 bytes)
- Header of data pages and blob pages are augmented with 4 bytes key version
- Header of transactional index pages are augmented with 4 byte key version
- Header of non-transactional index pages are augmented with 4 byte key version and 8 byte nonce (instead of LSN)

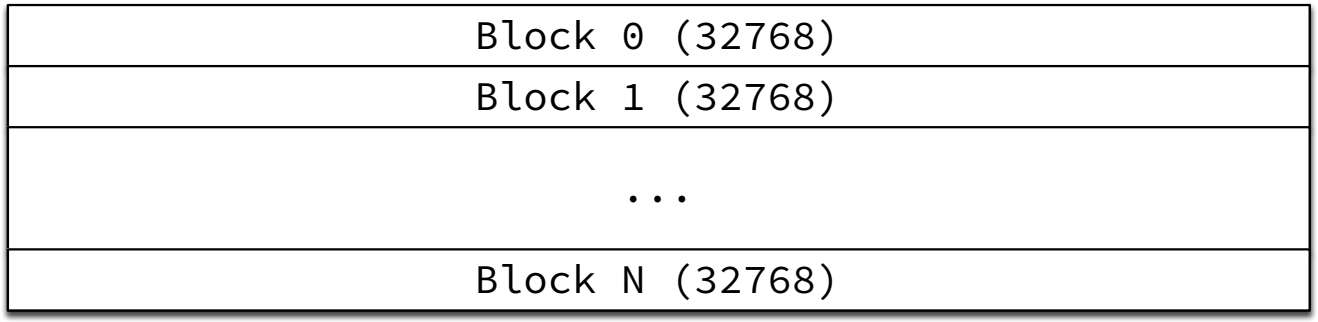
Temporary Files
(sorting buffers mostly)

Temporary files (e.g filesort buffers)

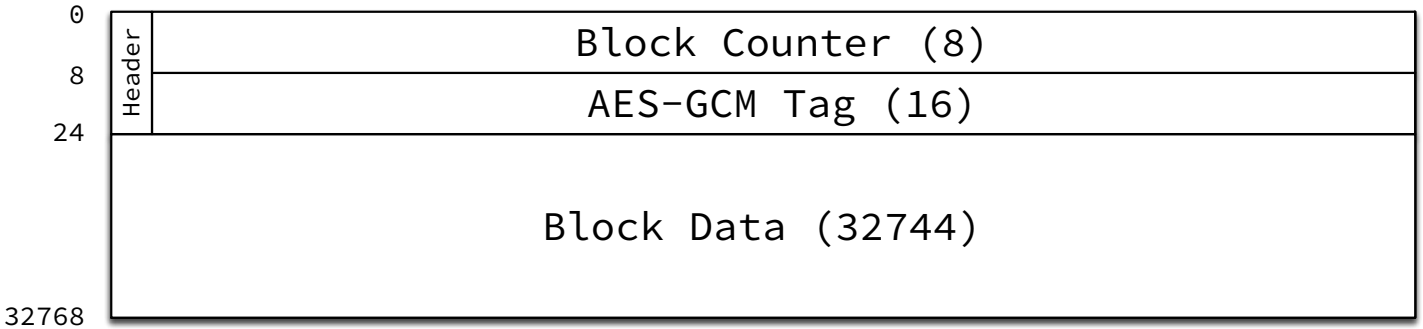
- Encryption key generated and stored in memory only
- Data in encrypted in blocks
- Block size is configurable using `temp_file_encryption_block_size` (32768)
- Each block is augmented with a 24 byte header

Temporary files (e.g filesort buffers)

Temporary Files



Temporary File Block



Where to find the code…

Code

innodb data	storage/fil/fil0crypt.cc
innodb redo log	storage/log/log0crypt.cc
binary/relay logs	sql/log_event.cc sql/log.cc
aria	storage/maria/ma_crypt.c
temp files	mysys/block_encrypted_io.cc

Google MySQL

See: code.google.com/p/google-mysql

Branch: mariadb-10.0.12/16-encryption
Branch: mariadb-10.0.12/17-scrubbing

MariaDB 10.1

MariaDB added encryption features to MariaDB 10.1 based somewhat loosely on our encryption work

This was done very recently and we haven’t reviewed it

We’re discussing things...

Conclusion

Conclusions...

Google takes security very seriously

Security presentations are great for people with sleeping problems — recording will be available for later use

Q&A