Lustre Client Encryption

09/2019

sbuisson@whamcloud.com

Lustre Client Encryption

- What is encryption for Lustre?
- Recap of last year's approach
- Alternative approach: fscrypt
- Current development status
- Remaining work

What is encryption for Lustre?

Use case:

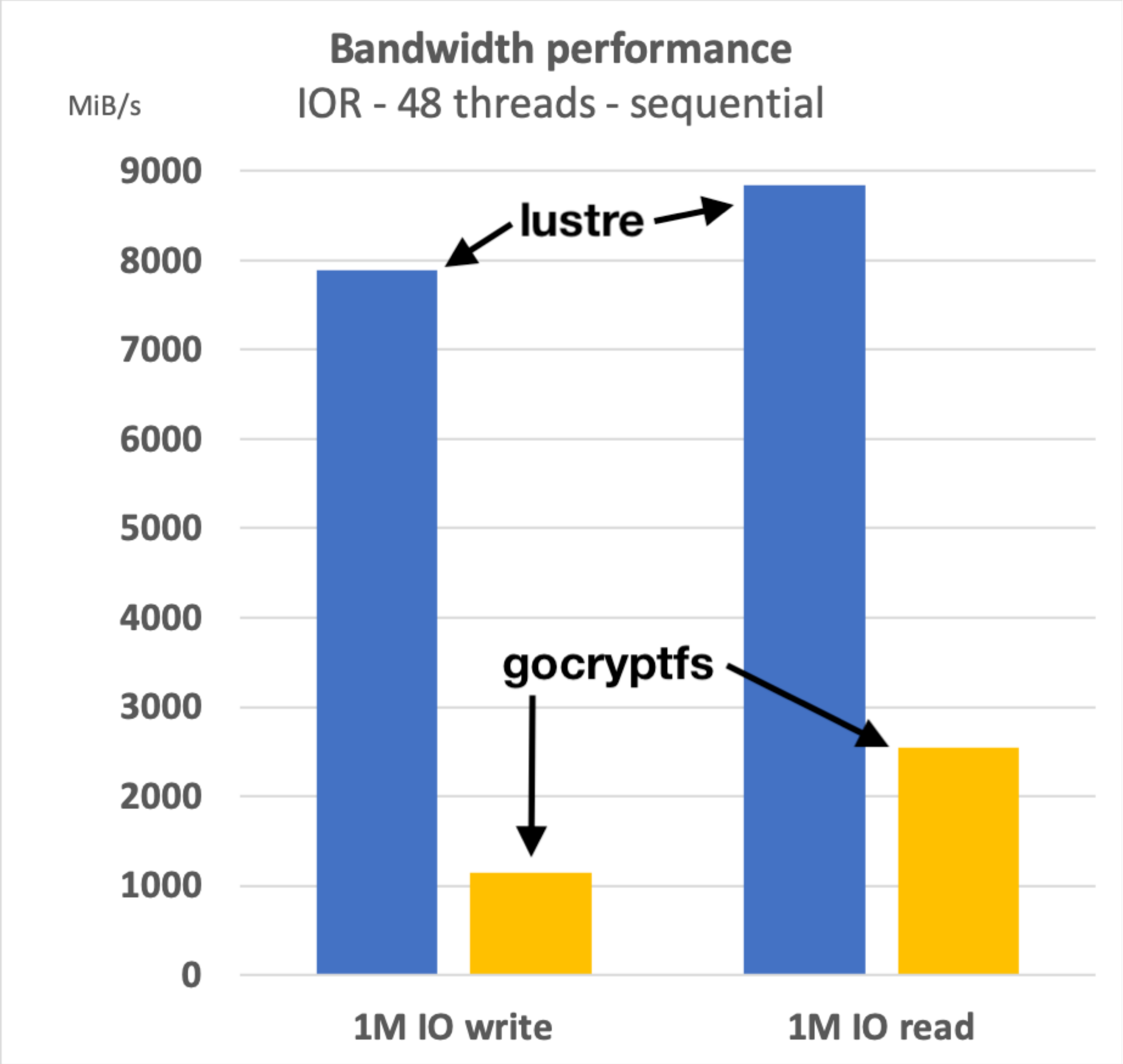
- Provide special directory for each user, to safely store sensitive files

Goals:

- Protect data in transit between clients and servers
- Protect data at rest

Last year recap: encryption on top of Lustre with Gocryptfs

- **Gocryptfs** stacked file system, written in GO, user space: FUSE
- Mount gocryptfs on top of Lustre client
  - Provides file content and file/directory name encryption
- Pros: immediately available and simple to implement
- Cons: performance penalty



Alternative approach: Lustre client encryption

- Implement encryption directly at the Lustre client level

- Requirements
  - Encrypt file content
  - Encrypt file/directory name
  - Have a master key for encryption
    - Per-file encryption key derived from master key
  - File data is no longer accessible after file is deleted (secure deletion)
  - End users provide their own encryption keys, and decide on dirs to encrypt
  - Deny access to encrypted data when master key is removed from memory
  - Able to change the user key without re-encrypting files
  - Access encrypted files from applications launched by a batch scheduler

Lustre Client Encryption – solution proposal

- Conform to fscrypt kernel API
  - Current users are ext4, F2FS, and UBIFS
  - Mature in 4.14 kernel
  - Usable implementation in Ubuntu 18.04 and RHEL8

- Reuse ext4 encryption principles
  - Encryption chunk size = system page size
  - encrypted page size = clear text page size
  - Encryption chunks are independent from each other
  - Pages in the page cache always contain clear text data

Lustre Client Encryption – solution proposal - continued

- ▶ Make use of fscrypt userspace tool
  - Manage encryption policies

  ⟹ Tell which directories to encrypt, and how

  - Need to use v2 encryption policies

- ▶ Ideally, share code infrastructure with client-side compression work
  - Same kind of operations, at same code locations

Lustre client encryption – addressing the requirements

fscrypt kernel API

- Encrypt file content
- Encrypt file/directory name
- Have a master key for encryption
  - ○ Per-file encryption key derived from master key
- File data is no longer accessible after file is deleted (secure deletion)

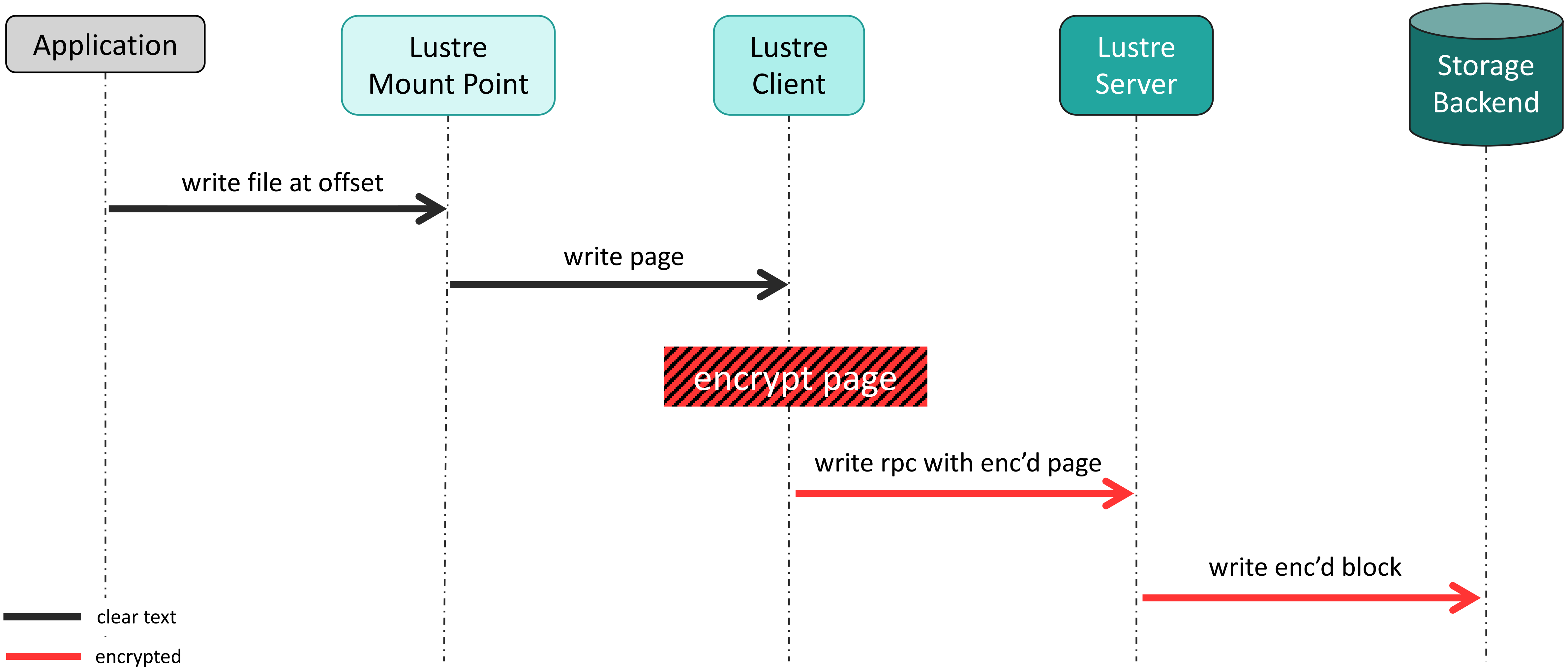
fscrypt userspace tool

- End users provide their own encryption keys, and decide on dirs to encrypt
- Deny access to encrypted data when master key is removed from memory
- Able to change the user key without re-encrypting files
- Work in “batch scheduler” mode

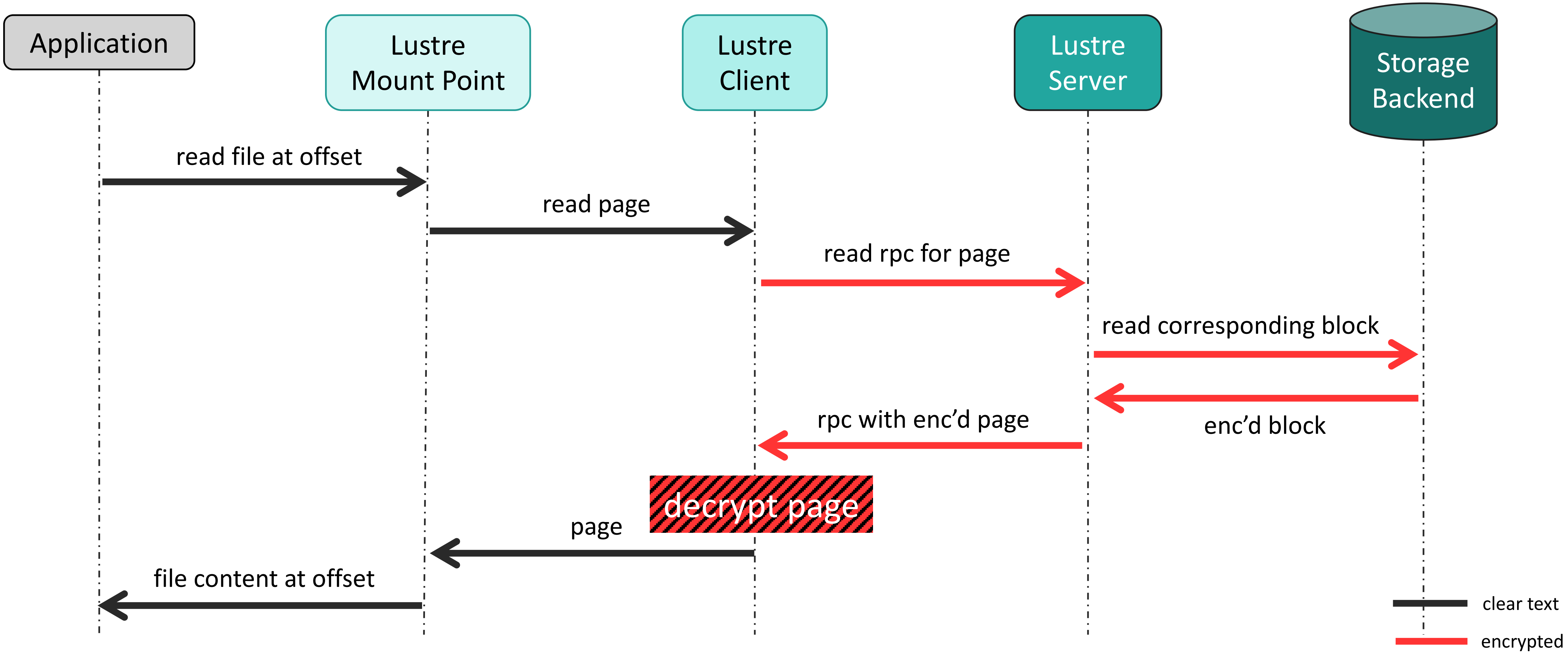
Lustre Client Encryption – data workflow

- Applications see clear text
- Data is encrypted before being sent to servers
  - Then remains untouched
- Data is decrypted upon receipt from servers
  - Untouched before that
- Servers only see encrypted data
  - But do not need to be aware of it
- Only client nodes have access to encryption keys

Lustre Client Encryption – write case



Lustre Client Encryption – read case



Lustre Client Encryption – threat model (details in LU-12275)

- Offline attacks
  - File contents and file names are protected
    - Confidentiality and integrity guaranteed if underlying encryption mechanism provides them
  - File metadata is not protected
    - e.g. file sizes, file permissions, file timestamps, and extended attributes
  - Existence and location of holes in files is not protected

Lustre Client Encryption – threat model - continued

- Online attacks
  - Vulnerable if the Linux Cryptographic API algorithms are…
  - Clear text file contents or filenames not hidden from other users on same client
    - UNIX rights, POSIX ACLs, or namespaces are here for that!
  - Lustre client kernel memory compromise can lead to encryption key compromise
    - Keys should be explicitly removed from memory after use
  - Lustre server kernel memory compromise has no effect
  - Per-file key compromise only impacts the associated file, not the master key

Lustre Client Encryption – development in progress

- Proof Of Concept quality code

- 5 patches pushed under LU-12275:
  - Common framework for flags, get/set encryption context
    - **dummy encryption mode** (fixed encryption key)
  - Implementation of encryption of file data on write path
  - Implementation of decryption of file data on read path
  - Proper file size handling
  - Non-regression tests to exercise encryption code

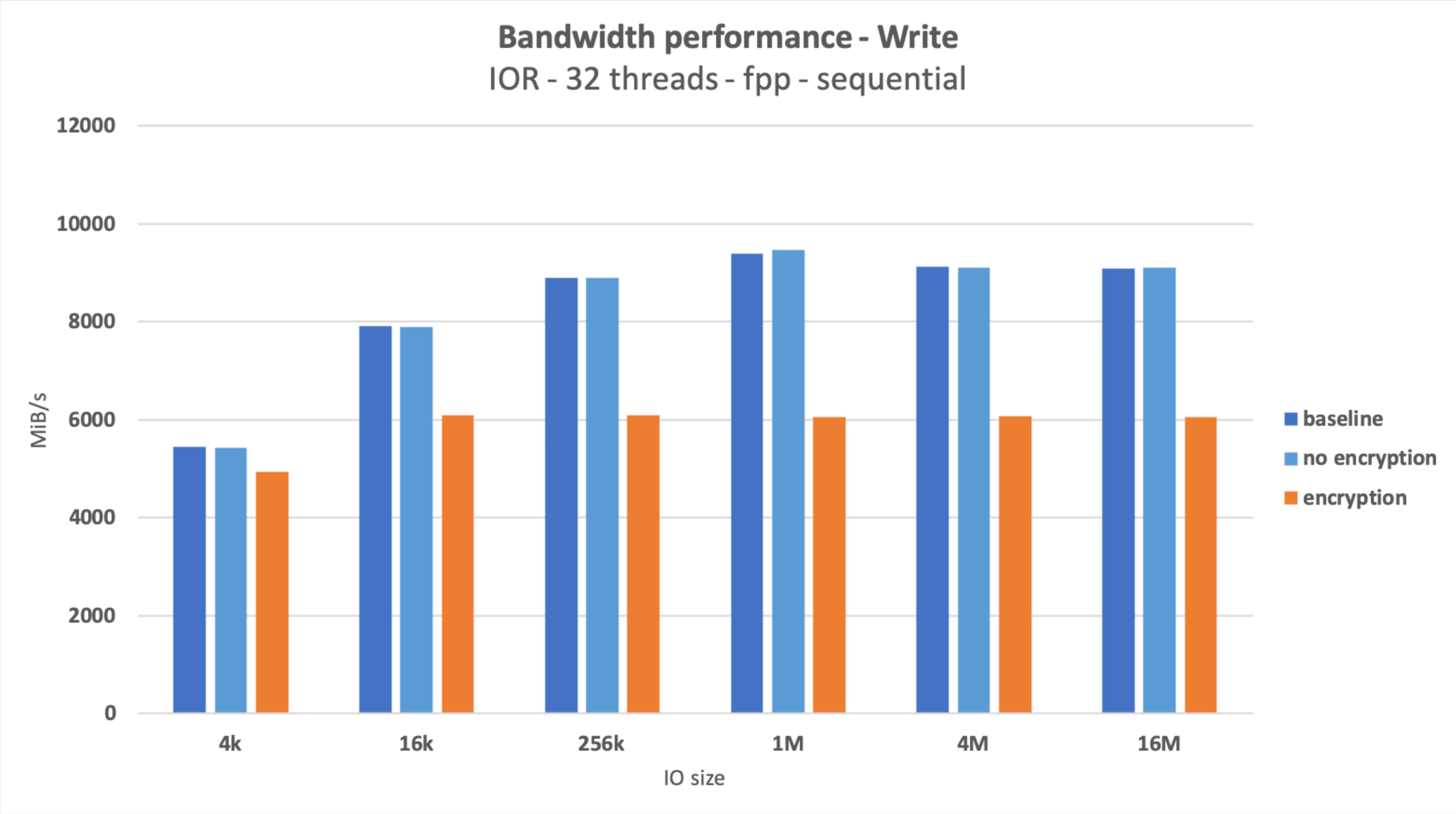
Lustre Client Encryption – Lustre subtleties

- Proper file size handling
  - Encryption chunk size is the page size
  - Ciphertext page is always full of data… even if clear text only contains one byte
  - But OSS assumes object size based on length of data received
    - Must carry on clear text length from client to server, and store along with object
- Checksum on request content
  - Client page cache contains clear text data
  - But ciphertext is sent to servers
    - Must not use pages in client cache for checksum calculation

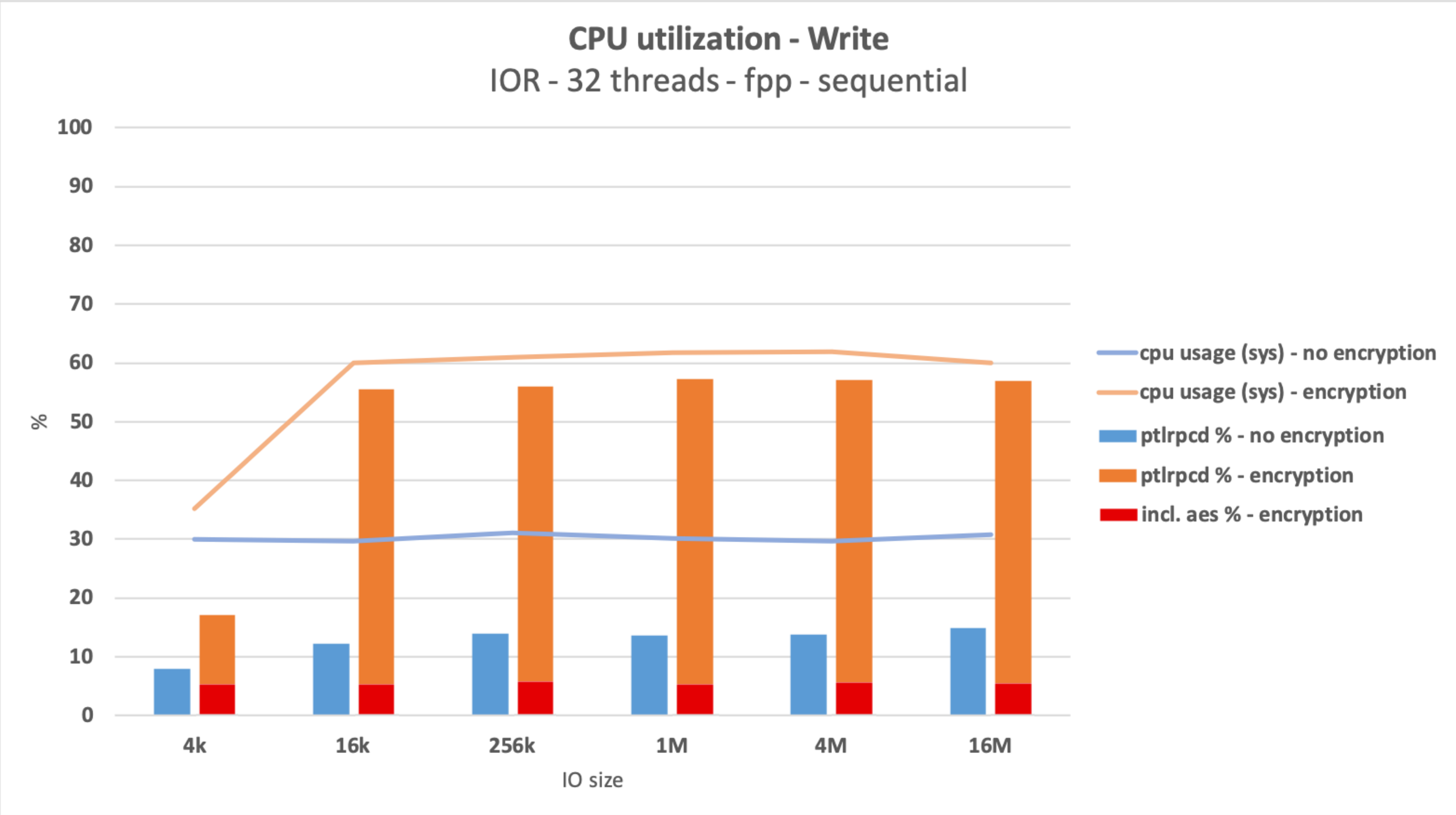
Lustre Client Encryption – performance evaluation

- POC code on top of `master`, **dummy encryption mode** (AES-256-XTS)
- Testbed
  - Client
    - Skylake 48 cores, Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz
    - 96 GB RAM
    - ConnectX-4 Infiniband adapter, EDR network
  - Storage
    - DDN ES200NV, 20 x NVMe HGST 1,7TB, 1 DCR pool
    - 4 OSTs, each 1/10th of pool
- Methodology
  - IOR, file per process, sequential IO
  - IOR, file per process, random IO

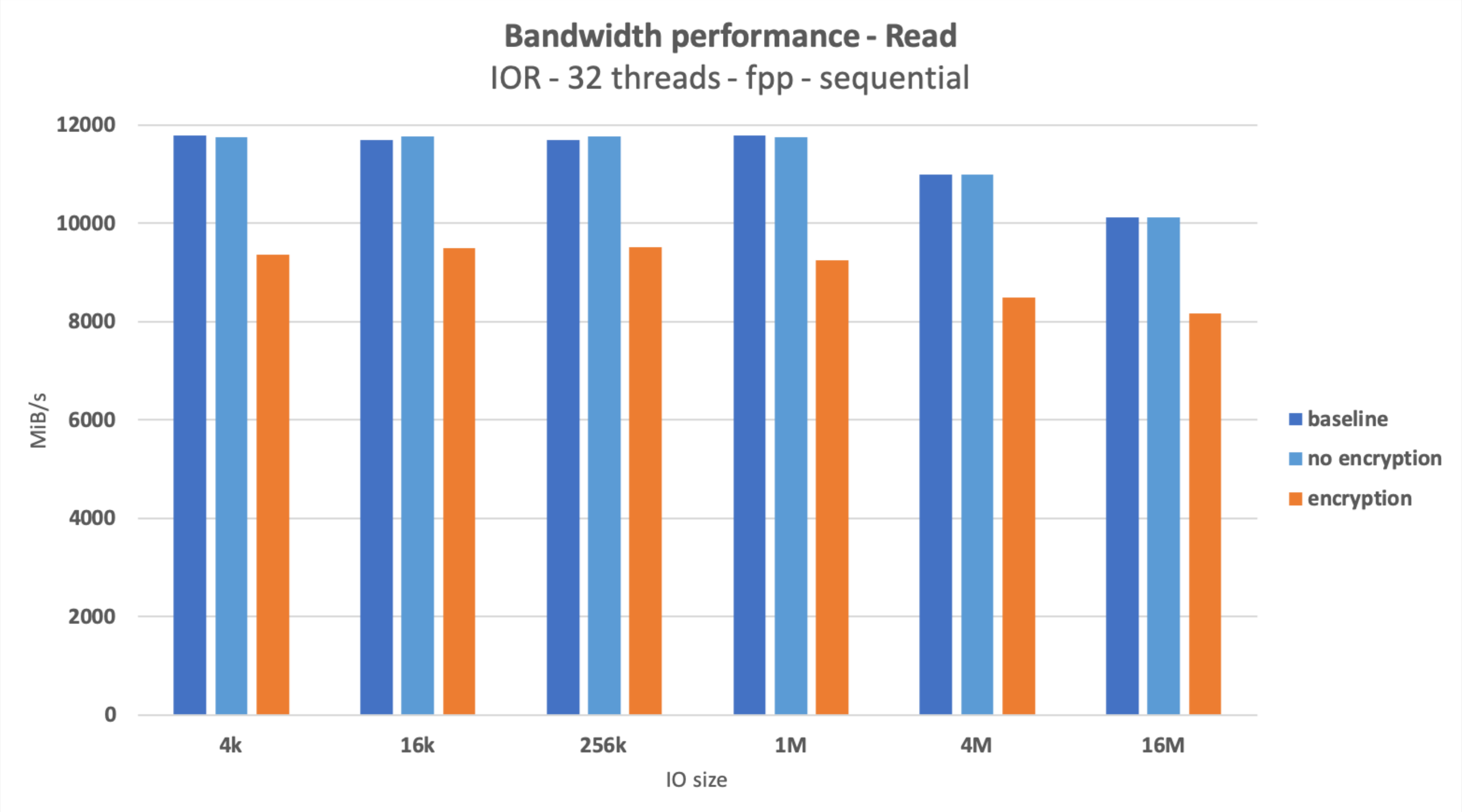
Lustre Client Encryption – early performance evaluation



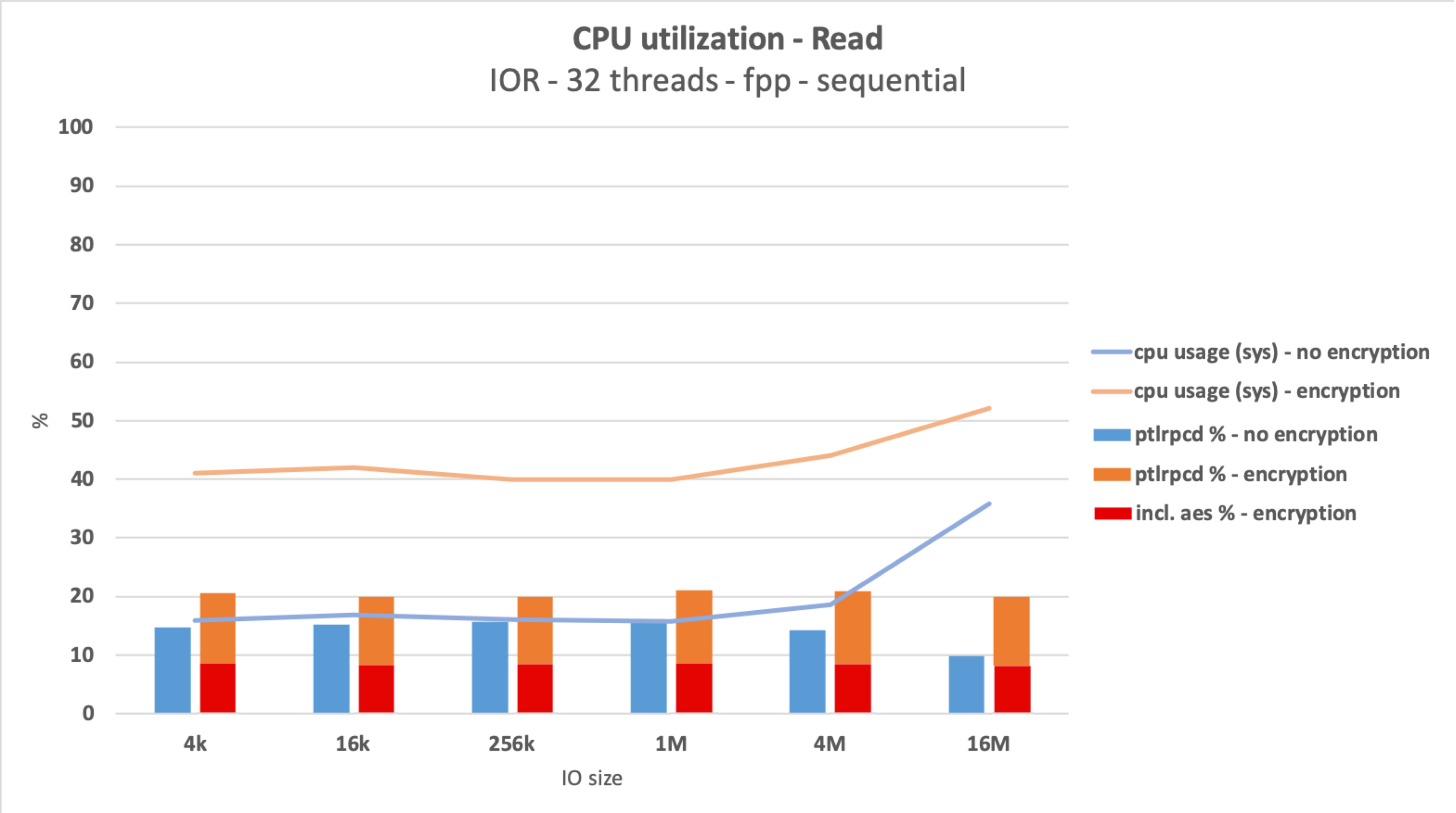
Lustre Client Encryption – early performance evaluation



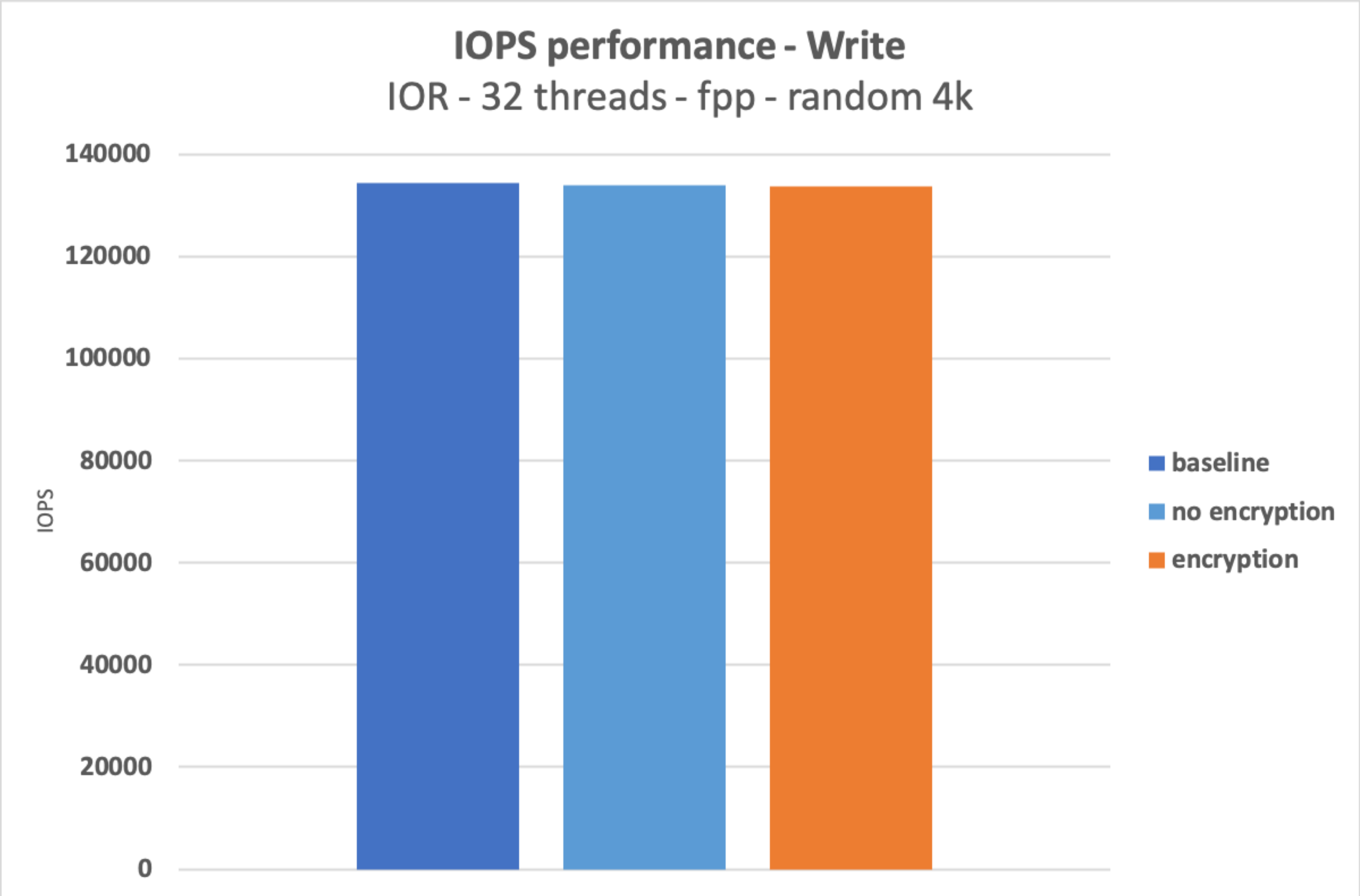
Lustre Client Encryption – early performance evaluation



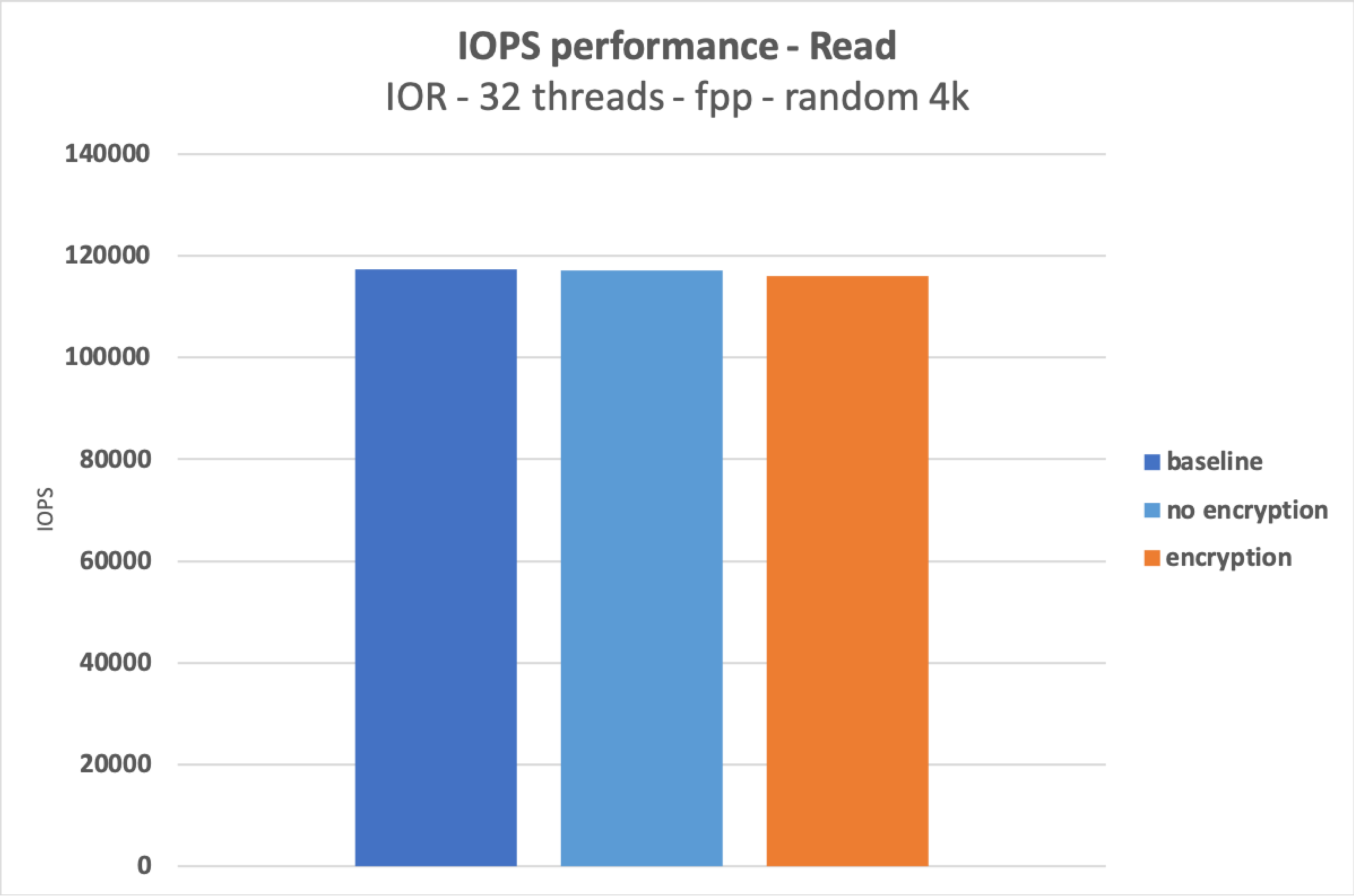
Lustre Client Encryption – early performance evaluation



Lustre Client Encryption – early performance evaluation



Lustre Client Encryption – early performance evaluation



Lustre Client Encryption – remaining development

- ▶ Encryption of file, symlink and directory names
  - Measure metadata performance impact
- ▶ Ability to set encryption policies on directories
  - Support new IOCTLs from fscrypt userspace tool
- ▶ Lustre specific optimizations: eg encryption context
  - Per-file encryption context is stored in an xattr
  - Getting/setting xattrs impacts performance by generating additional requests
  - Lustre must be able to
    - o Set encryption context directly with create request
    - o Fetch encryption context directly with open/lookup request

Lustre Client Encryption – challenges

- Distributed Namespace (DNE)
  - ⟹ Impact on file name encryption?

- File Level Redundancy (FLR)
- Data-on-MDT (DoM)
- File migration
- Request replay
  - ⟹ Impact on file content encryption?

- More generally, *the goal is for the performance penalty to only be the time spent on encryption and decryption.*

Conclusion

- This is just early stage of evaluation
  - Remaining development
  - Necessary optimizations
  - Metadata performance evaluation
- Encouraging bandwidth performance level
  - Good replacement for “Gocryptfs on top of Lustre” solution
- Advantage of simplicity once done
  - At the cost of development effort
- Key management is closely-related hot topic

Whamcloud

Thank you!

sbuisson@whamcloud.com

DDN® STORAGE