usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

FAST'19

Speculative Encryption on GPU Applied to Cryptographic File Systems

Vandeir Eduardo[1,2], Wagner M. Nunan Zola[1], and Luis C. Erpen de Bona[1]

[1] Federal University of Paraná
[2] University of Blumenau

Agenda

- Introduction and motivation
- Rationale:

  Cryptographic File Systems (CFSs), CBC and CTR encryption mode,

  file system EncFS (user space) and GPU library WAESlib
- CTR encryption mode applied to CFSs (in file system EncFS++)

  Generation and storage of nonces
- Spawning parallel encryption tasks in EncFS++

  (Challenges in organization and management of encryption contexts)
- Experimental Performance Analysis EncFS++
- Conclusions

Introduction and motivation

- Security in data storage:
  especially in the era of computing in the cloud.
- Natural evolution: integration of *encryption* in File Systems:
  FSs → CFSs
- Use of symmetric block ciphers (good security/speed ratio)
- Problems: Larger data volumes
  + faster media
  + alternative ciphers
  + larger keys

  = increase in CPU utilization

Motivation (cont.)

FAST'19

- Wanted: Using parallel processors for the task (e.g with GPUs)
  (or with multicore processors)

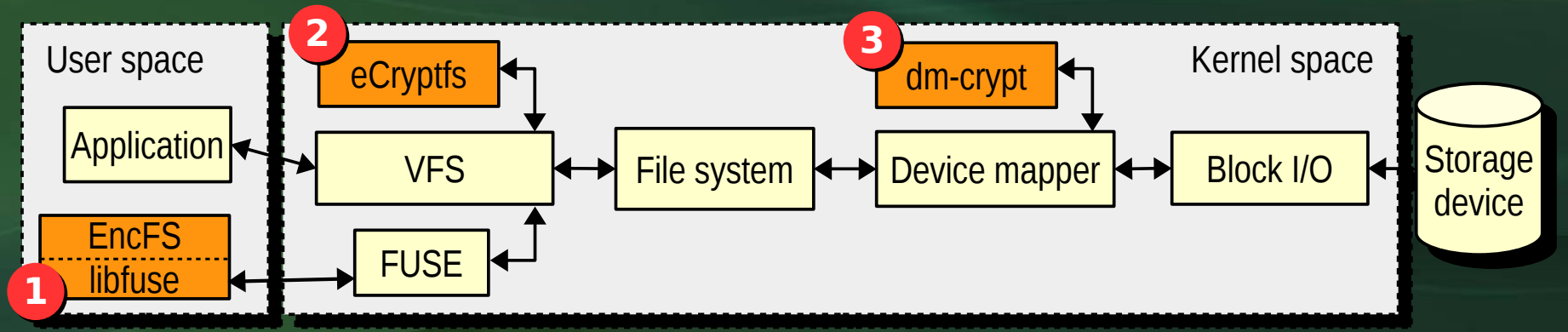
- Previous study of acceleration of AES in GPU
  GPU kernel WAES and WAESlib: exploring CTR mode
  → defines priorities for generation of *encryption masks*

- Current work: "Explore advantages of CTR mode in the context of CFSs", with parallel multicore or manycore processors:

  - using GPU cryptographic functions (current work)
  - get higher throughput with more efficient CPU usage
  - extend to other accelerators, multicore or heterogeneous (future work)

Cryptographic FILE Systems

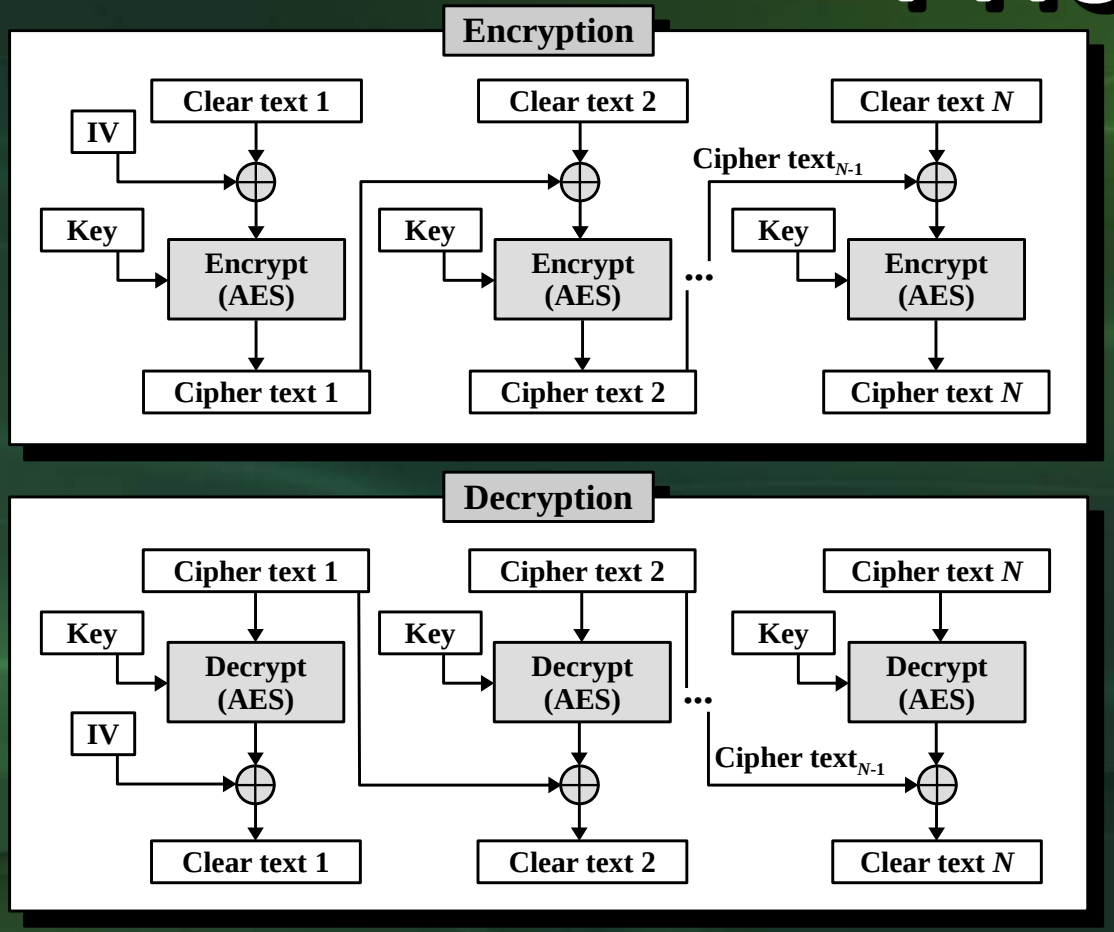
- Integrated at different system levels:
  1. User Space: FUSE-based CFSs
  2. Kernel Space: CFSs ↔ VFS
  3. Kernel Space: Cryptographic Systems ↔ I / O Blocks



usually: CBC mode of operation

- Detailed in NIST document SP 800-38A
- Sequential encryption (data dependency)
- Security requirement:

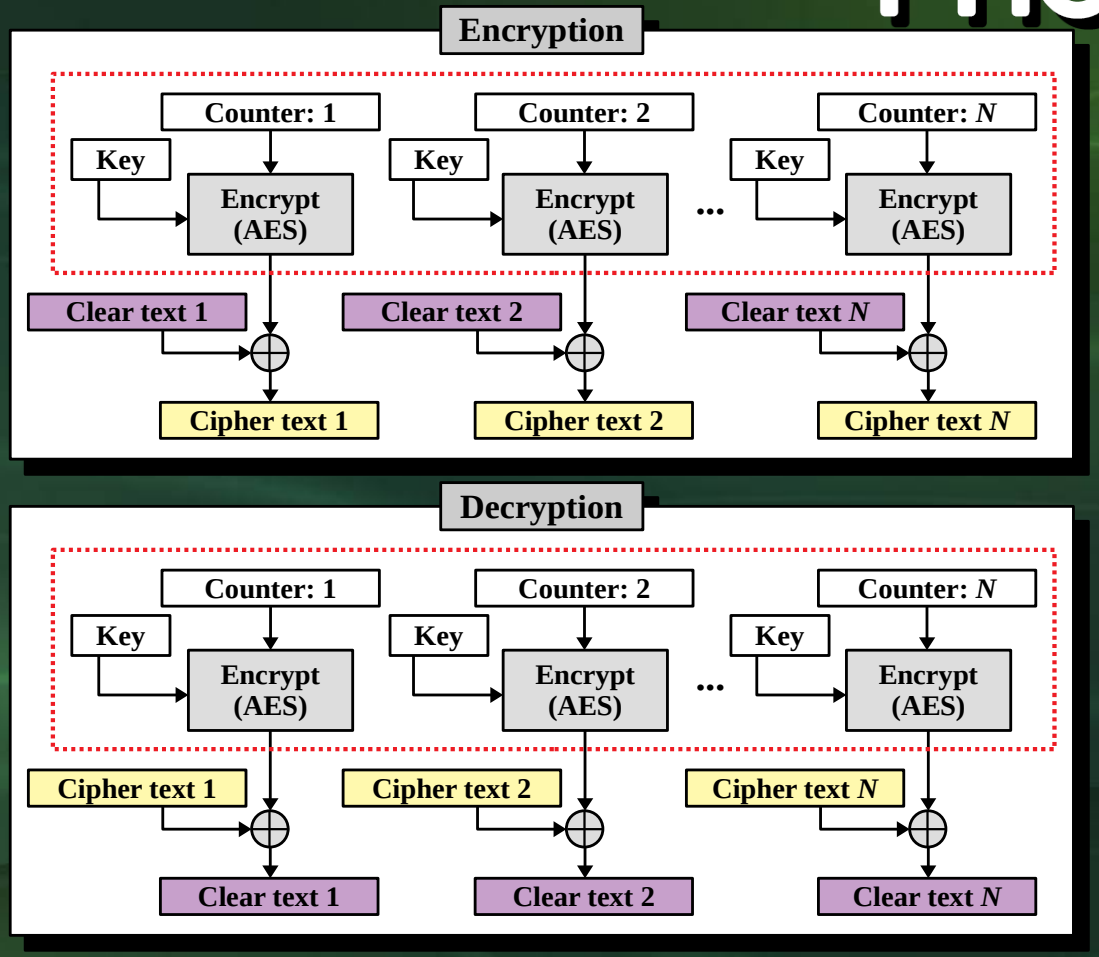
  necessary to use an "unpredictable" *Initialization Vector* (IV)



Wanted: work with CTR Mode

- Parallelizable
- Possibility of encryption Anticipation (of encryption masks)
- Security requirement: (uniqueness requirement)

  necessary to use a given (key, IV) pair only once at any encryption
- IV is called “Nonce”



EncFS file system: (some Features)

FAST'19

- based on FUSE
  - → works in user space → facilitates development / testing
  - → allows easier GPU library Integration in EncFS++
    - CUDA API and libfuse are in user space
    - IF using kernel space FS module:
      needed an intermediate process to use CUDA API
      (+ complexity, + latency)

EncFS Features

→ based on FUSE / space user facilitates development / testing

→ uses OpenSSL (CPU)

→ file content encrypted in data blocks

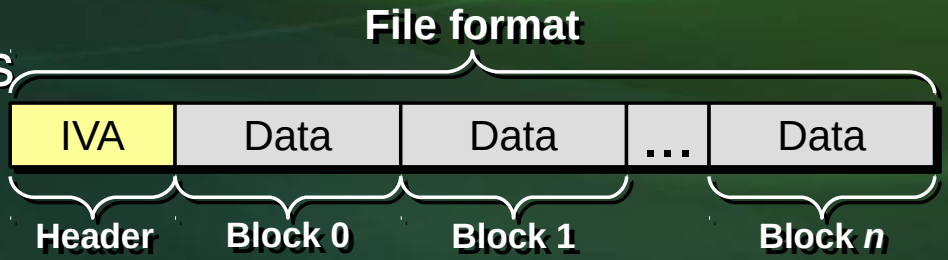
→ uses CBC for each data block

vK = volume Key

IVV = Volume IV

IVA = File IV

IVB = data Block IV



IV use unpredictability requirement:

* data block IV calculated dynamically with encryption hash (no need to store)
* reusable in block rewriting

data Block IV (IVB) = HMAC_CTX (vK, IVV || (NumBlock ⊕ IVA))

GPU Encryption Acceleration

FAST'19

- Extensively studied: for varous symmetric ciphers such as AES, Blowfish, IDEA, Camellia, etc.
- Related work: acceleration of cryptographic functions in some applications:
  → User space: Engine-CUDA, CrystalGPU, CRSFS

  → kernel space: OCF, Gdev, GPUStore
- Usually:
  using CBC+GPU → usually only compensates for larger requests (> 16 KiB)
- Applied to CFSs:
  no previous work have exploited the benefits of CTR mode

Why CTR?

FAST'19

- CTR Mode:
  → parallelizable
  → allows speculative encryption
  (creation encryption masks ahead of time)
  → XOR on CPU (avoids CPU → GPU data transfer)
  → As safe as CBC

- Previous library available in previous work: WAESlib
  → Reduces GPU processing complexity

  → Aggregation of small (4 KiB) contexts :

  - fewer WAES kernel activations
  - higher throughput (GPU → CPU)
  - more control in the order of production of masks (with priorities)

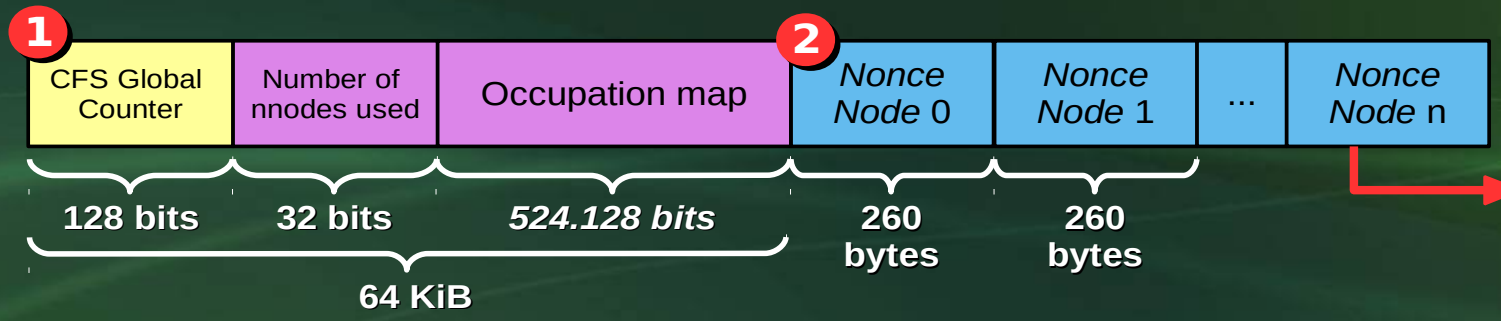
Challenges of using CTR in CFSs

FAST'19

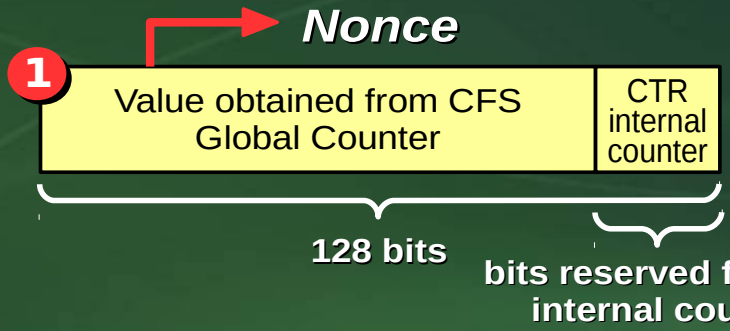
- Each recording and rewriting of a block requires a new nonce (due to: the uniqueness requirement)
- Problem: Necessary to store a nonce per block (same unique nonce used in encryption is necessary in decryption)
- Overhead of nonce storage could negatively impact CFS performance

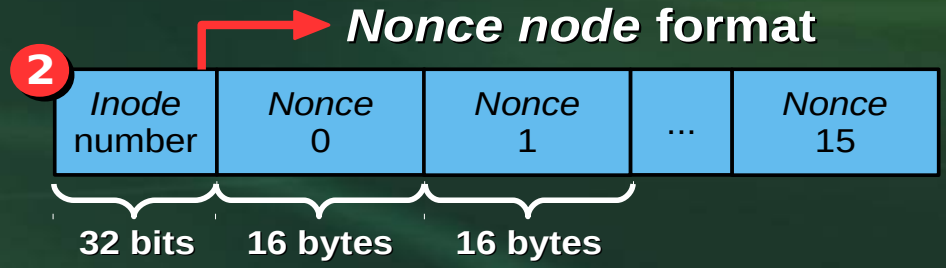
  Nonce storage format AND Access mechanism AND granularity are important for performance

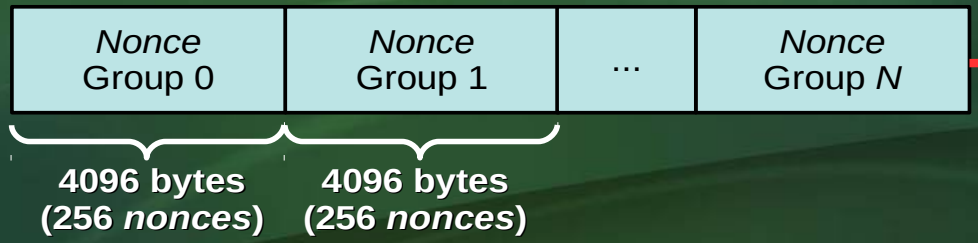
Nnodes: how nonce Nodes are stored in EncFS++



Nonce nodes file format
(loaded when CFS is mounted)





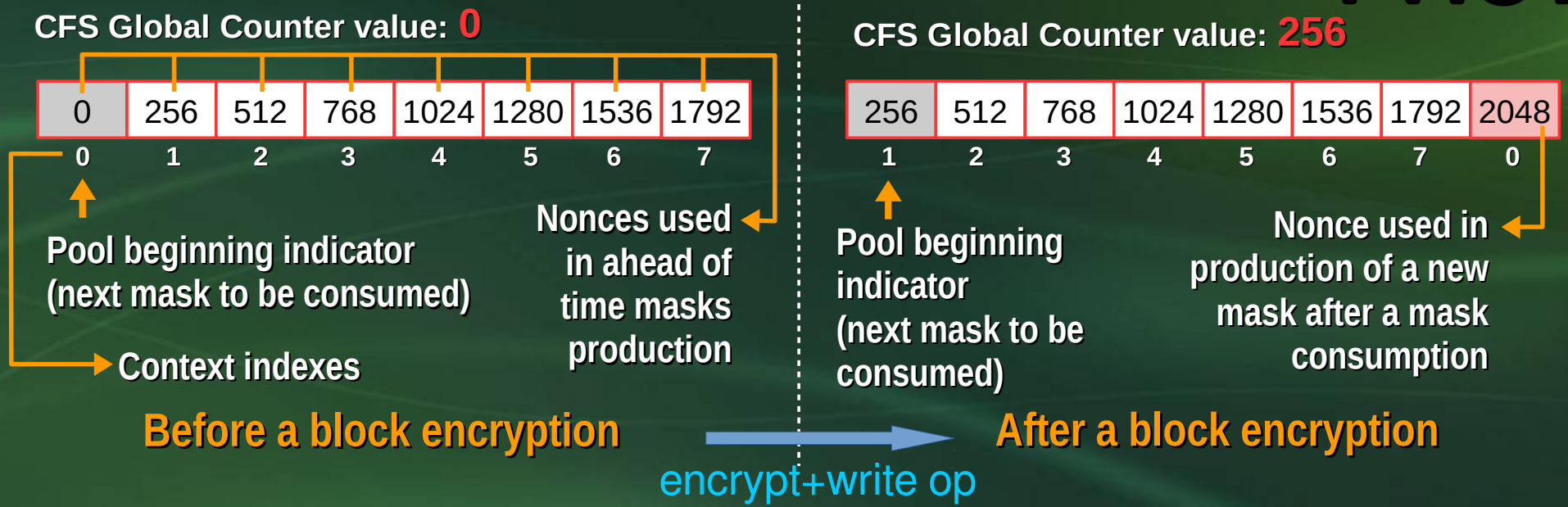


Exclusive *nonces* file format
Only for files > 64 KiB (16 * 4 KiB)
(loaded when file is opened)

Challenges in using speculative encryption in CFSs

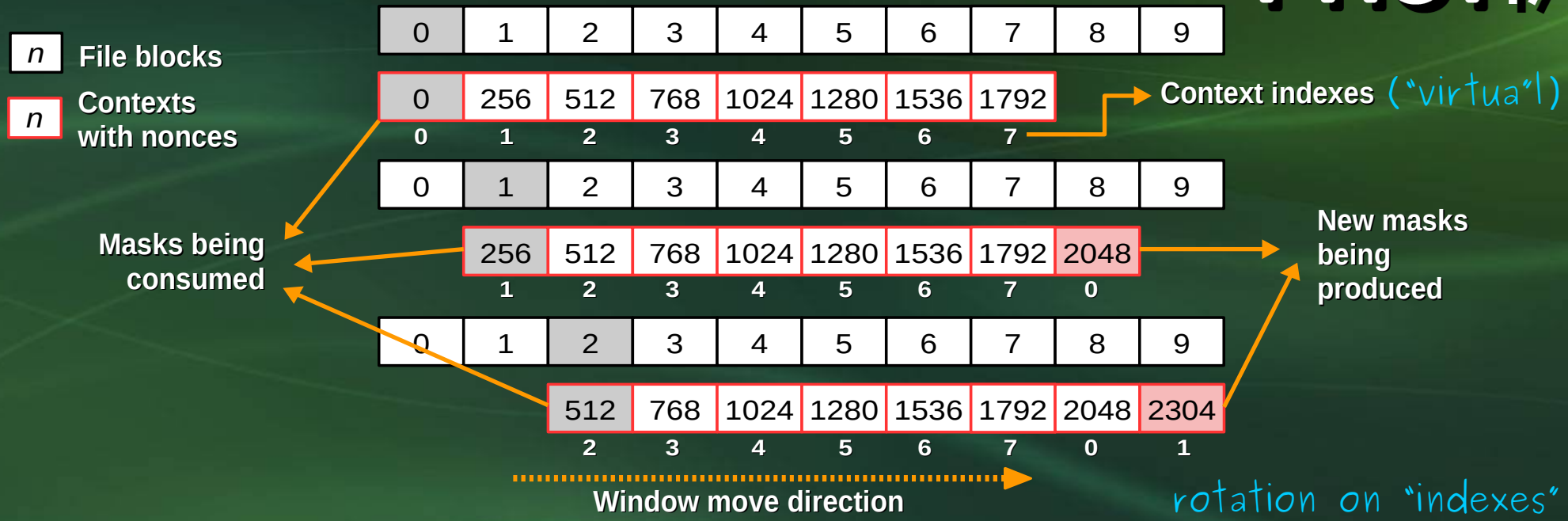
- Managing encryption contexts
  → How to organize the encryption contexts within the FS application?
  → How to use these contexts in the different CFS operations?
- When is the best time to trigger the generation of encryption masks (define contexts)?
- How to take advantage of the priority feature?

Write Context pool: maintained for encryption+writing



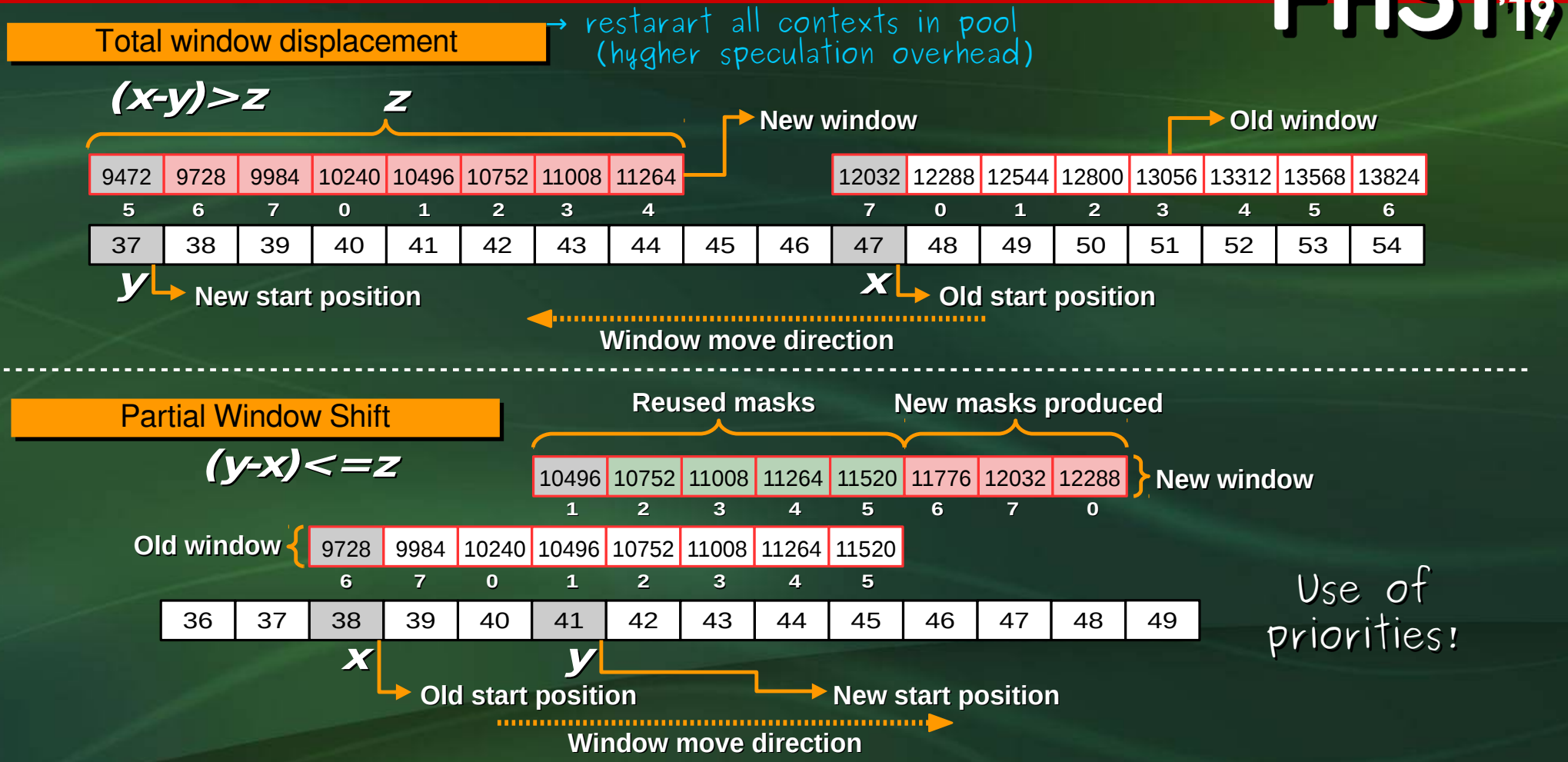
- Used for sequential and random writing (only one write context POOL needed per CFS)
- Contexts initially defined at CFS mount operation
- Contexts in this POOL are redefined as masks are consumed (uses lower priority)
- Implemented as a virtual circular queue (no storage → performance)

Context pool for decryption / read (seq.)



- Used for sequential and random reading (1 per file)
- Contexts initially defined in each file open operation (decreasing priority according to position)
- Contexts redefined as masks are consumed (uses lower priority)

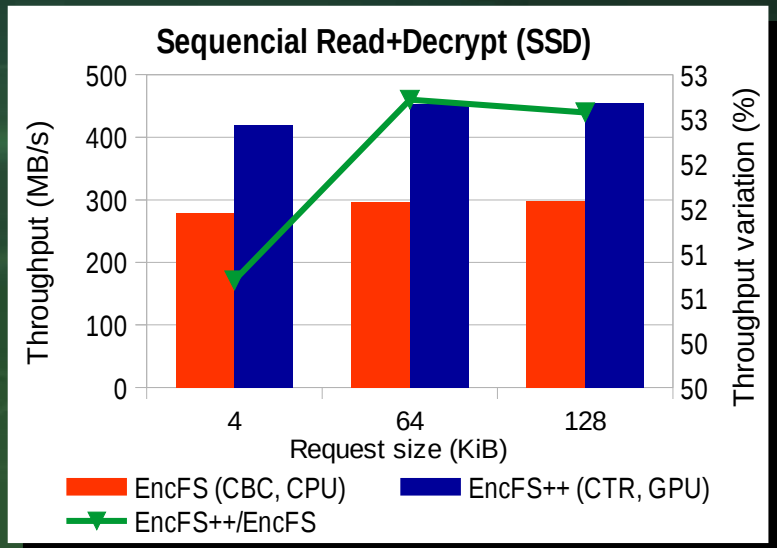
Context pool for decryption / read (random)

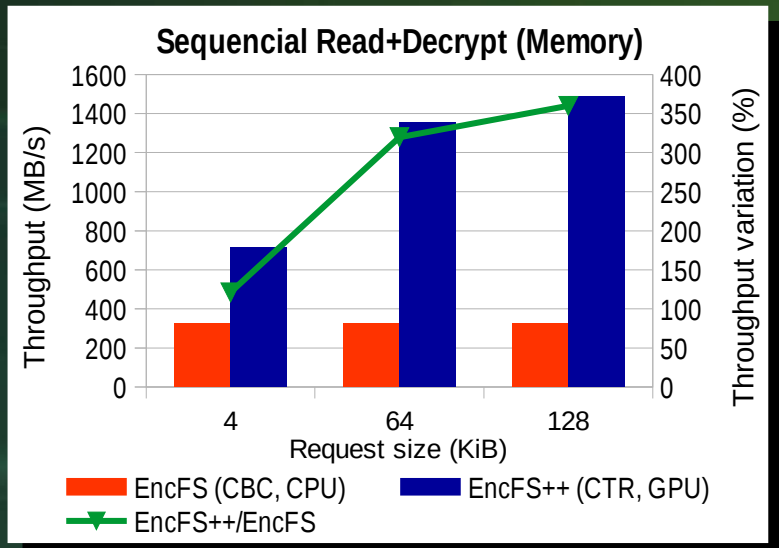


Performance Analysis

- Performance comparisons between EncFS (CBC), eCryptfs (CBC) and EncFS++ (CPU / GPU) and AESNI in eCryptFS (kernel mode)
- Microbenchmark with measuring flow in sequential and random read and write operations (requests: 4, 64 and 128 KiB in a big 16 GiB file)
- Macrobenchmark using filebench workloads with variation in number of threads (fileserver.f and webserver.f)
- Linux kernel 4.10.0, Intel Core i7-7700HQ at 2.8 GHz (fixed frequency), 32 GiB RAM, SSD disk (≈500 MB / s) and ramdisk (/ run / shm), libfuse 2.9.4, OpenSSL 1.0 .2g, WAESlib 2.01g0, ext4 base FS, NVIDIA GeForce GTX 1070 mobile (Pascal Architecture)

Microbenchmark (sequencial Read +Decrypt)

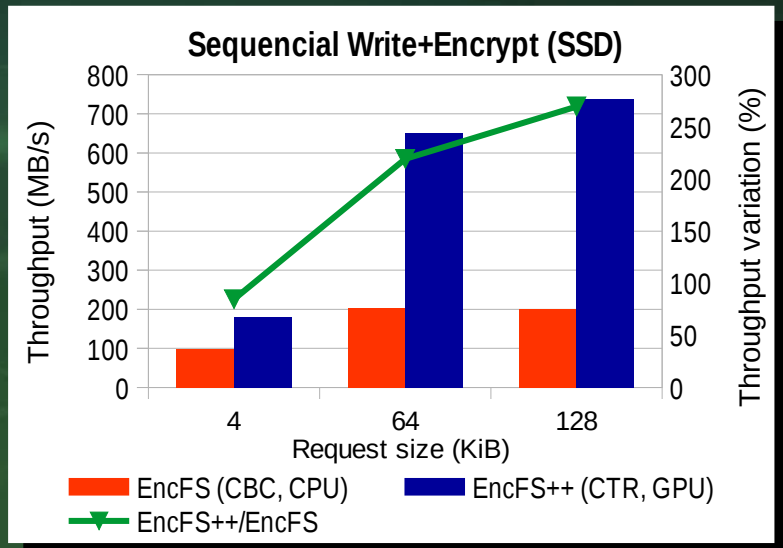


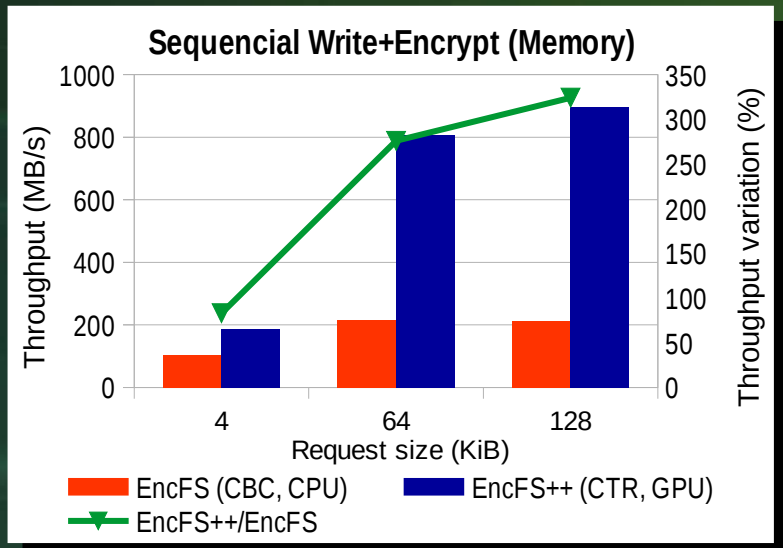


Sequencial Read+Decrypt (SSD)						
Req. Size (KiB)	EncFS (CBC, CPU)		EncFS++ (CTR, GPU)		EncFS++/EncFS	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput Var. (%)	CPU use efficiency
4	278.47	11.32	419.66	14.71	50.70	1.16
64	297.04	12.10	453.64	11.54	52.72	1.60
128	297.69	12.11	454.20	11.54	52.58	1.60

Sequencial Read+Decrypt (Memory)						
Req. Size (KiB)	EncFS (CBC, CPU)		EncFS++ (CTR, GPU)		EncFS++/EncFS	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput Var. (%)	CPU use efficiency
4	323.25	12.20	714.82	21.76	121.14	1.24
64	322.91	12.20	1,355.20	23.90	319.68	2.14
128	323.03	12.20	1,485.59	23.90	359.90	2.35

Microbenchmark (sequencial write +encrypt)

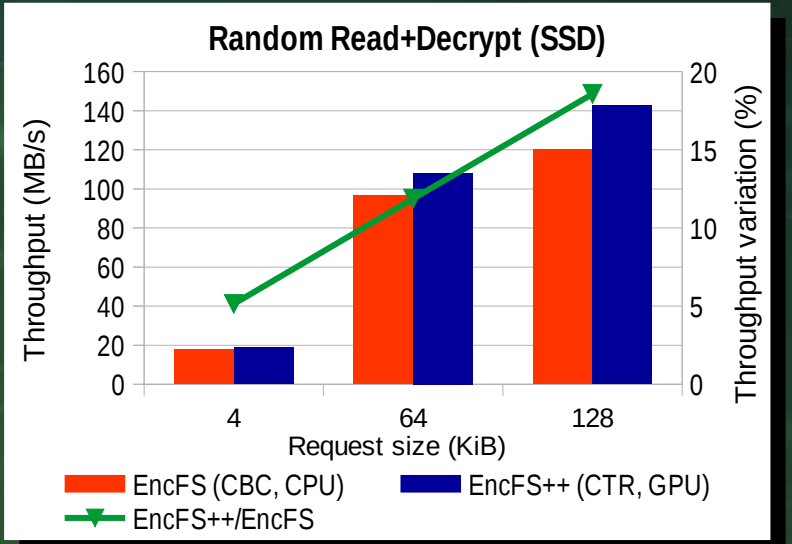


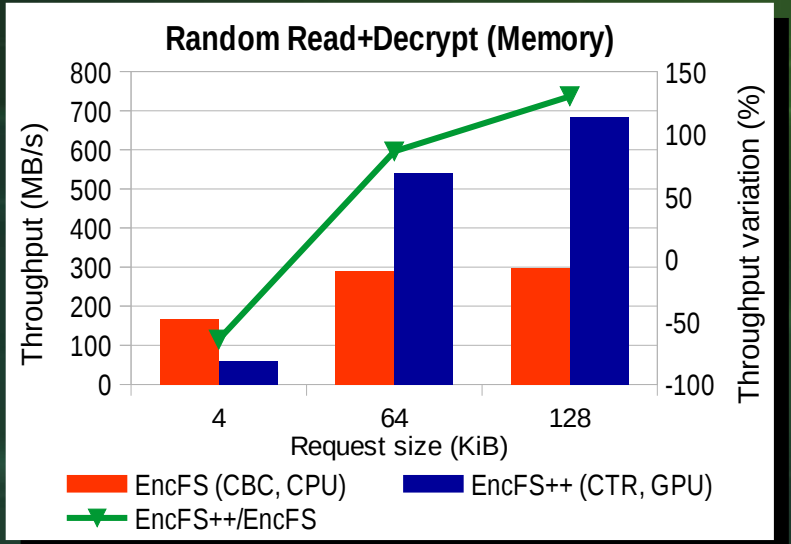


Sequencial Write+Encrypt (SSD)						
Req. Size (KiB)	EncFS (CBC, CPU)		EncFS++ (CTR, GPU)		EncFS++/EncFS	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput Var. (%)	CPU use efficiency
4	97.43	10.41	179.86	11.75	**84.61**	**1.64**
64	203.79	10.55	650.71	14.16	**219.30**	**2.38**
128	199.87	9.93	738.43	11.75	**269.44**	**3.12**

Sequencial Write+Encrypt (Memory)						
Req. Size (KiB)	EncFS (CBC, CPU)		EncFS++ (CTR, GPU)		EncFS++/EncFS	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput Var. (%)	CPU use efficiency
4	102.88	10.42	188.11	12.19	**82.83**	**1.56**
64	214.71	11.00	806.92	18.59	**275.82**	**2.22**
128	211.56	10.43	897.25	18.94	**324.12**	**2.33**

Microbenchmark (random read +decrypt)

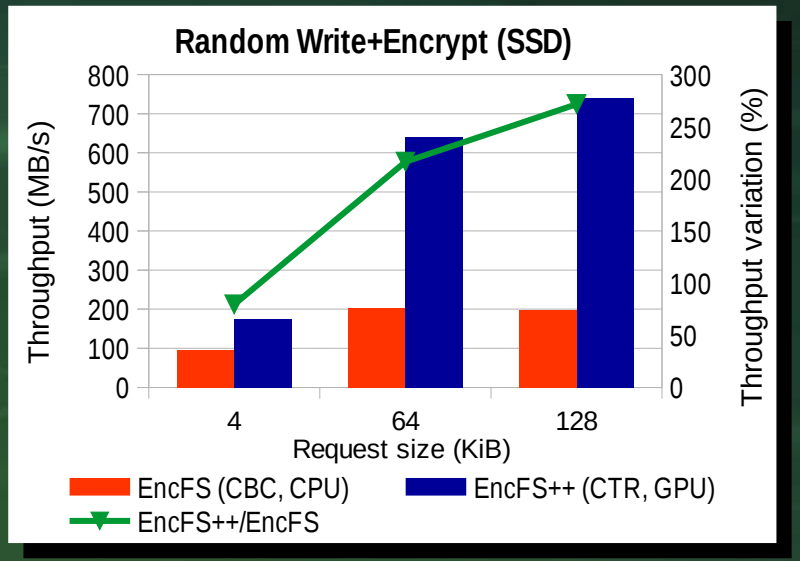


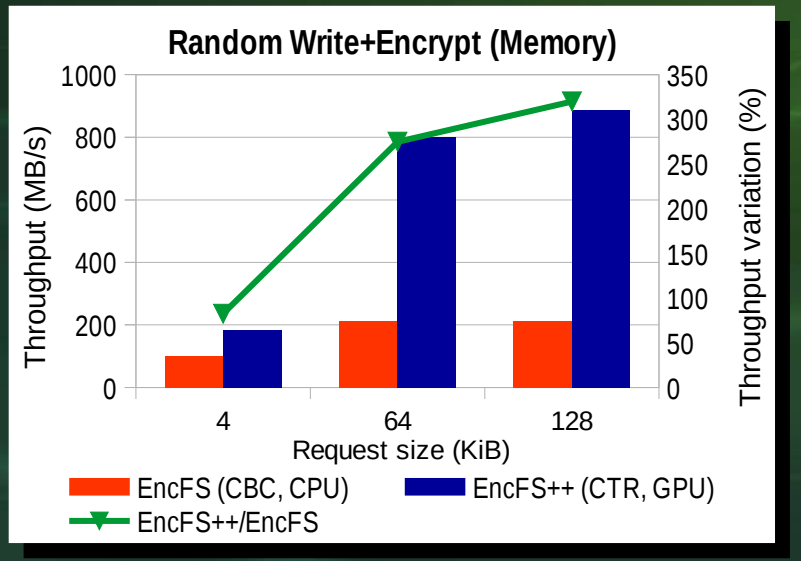


Random Read+Decrypt (SSD)						
Req. Size (KiB)	EncFS (CBC, CPU)		EncFS++ (CTR, GPU)		EncFS++/EncFS	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput Var. (%)	CPU use efficiency
4	17.80	1.93	18.71	3.75	5.14	0.54
64	96.48	4.37	107.95	5.35	11.88	0.91
128	120.17	5.34	142.50	5.38	18.57	1.18

Random Read+Decrypt (Memory)						
Req. Size (KiB)	EncFS (CBC, CPU)		EncFS++ (CTR, GPU)		EncFS++/EncFS	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput Var. (%)	CPU use efficiency
4	166.68	10.37	59.72	8.30	-64.17	0.45
64	290.74	11.40	541.54	17.12	86.26	1.24
128	297.11	11.45	684.10	17.37	130.25	1.52

Microbenchmark (random write +encrypt)





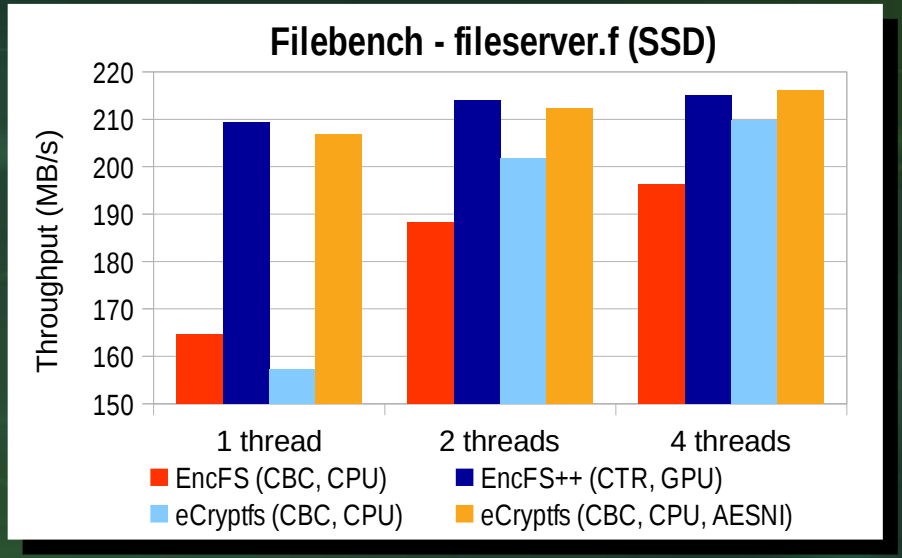
Random Write+Encrypt (SSD)

Req. Size (KiB)	EncFS (CBC, CPU)		EncFS++ (CTR, GPU)		EncFS++/EncFS	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput Var. (%)	CPU use efficiency
4	96.61	10.36	173.46	11.54	79.54	1.61
64	202.53	10.50	640.83	14.06	216.42	2.36
128	198.91	9.91	739.30	11.60	271.69	3.17

Random Write+Encrypt (Memory)

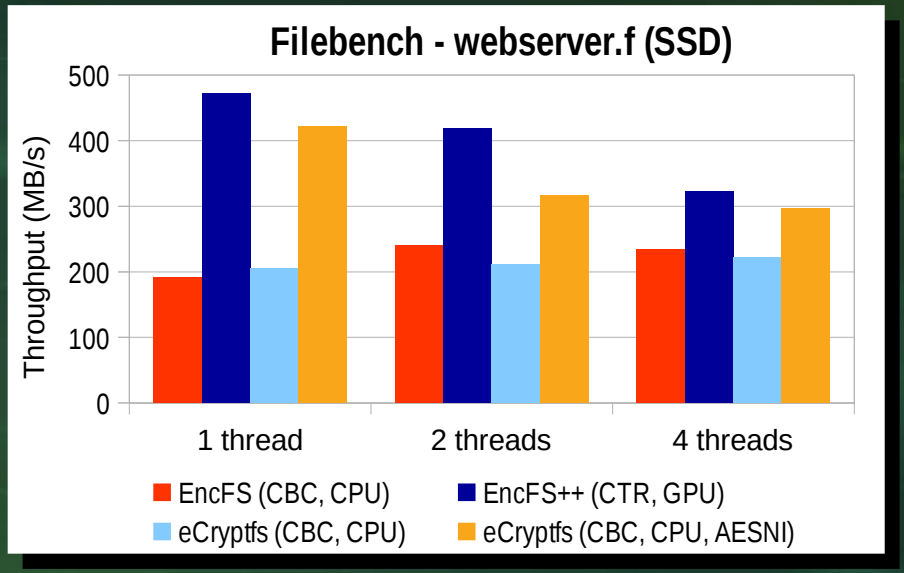
Req. Size (KiB)	EncFS (CBC, CPU)		EncFS++ (CTR, GPU)		EncFS++/EncFS	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput Var. (%)	CPU use efficiency
4	99.81	10.28	182.13	11.94	82.47	1.57
64	213.83	10.97	801.79	18.57	274.96	2.21
128	211.47	10.43	888.46	18.89	320.13	2.32

Macrobenchmark (fileserver.f)



Threads	EncFS (CBC, CPU)		eCryptfs (CBC, CPU)		eCryptfs (CBC, CPU, AESNI)		EncFS++ (CTR, GPU)			EncFS++/ EncFS (CBC)		EncFS++/ Ecryptfs (CPU)		EncFS++/eCryptfs (CPU, AESNI)	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	GPU (%)	Thrput Var. (%)	CPU Effic.	Thrput Var. (%)	CPU Effic.	Thrput Var. (%)	CPU Effic.
1	164.76	9.94	157.22	8.84	206.90	4.61	209.38	4.68	10.46	27.08	2.70	33.18	2.52	1.20	1.00
2	188.24	11.42	201.88	11.46	212.36	4.37	214.04	5.83	12.75	13.71	2.23	6.02	2.08	0.79	0.75
4	196.24	12.33	209.74	16.14	216.06	10.92	215.12	5.79	11.39	9.62	2.33	2.57	2.86	-0.44	1.88

Macrobenchmark (webserver.f)



Threads	EncFS (CBC, CPU)		eCryptfs (CBC, CPU)		eCryptfs (CBC, CPU, AESNI)		EncFS++ (CTR, GPU)			EncFS++/ EncFS (CBC)		EncFS++/ Ecryptfs (CPU)		EncFS++/eCryptfs (CPU, AESNI)	
	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	Thrput (MB/s)	CPU (%)	GPU (%)	Thrput Var. (%)	CPU Effic.	Thrput Var. (%)	CPU Effic.	Thrput Var. (%)	CPU Effic.
1	192.28	8.03	204.88	10.85	422.54	3.61	471.80	11.24	25.47	145.37	1.75	130.28	2.22	11.66	0.36
2	239.74	10.76	211.06	11.17	316.12	3.63	418.34	10.46	23.25	74.50	1.79	98.21	2.12	32.34	0.46
4	234.76	10.03	222.88	11.68	296.20	6.96	322.30	8.76	19.07	37.29	1.57	44.61	1.93	8.81	0.87

Conclusions

FAST'19

- Microbechmark, with FS in memory:

  Throughput: gains up to ≈360% (sequential read),
  ≈130% (random read), ≈320% (sequential and random writing).
  CPU Efficiency: gains up to ≈2.3x (sequential read and write and random write),
  ≈1.52x (random read)

- Macrobechmark (fileserver), with FS in SSD:

  Throughput: gains up to ≈27% (vs EncFS), ≈33% (vs eCryptfs).
  CPU Efficiency: ≈2.7x (vs EncFS), ≈2.9x (vs eCryptfs)

- Macrobechmark (webserver), with FS in SSD:

  Throughput: gains up to ≈145% (vs EncFS), ≈130% (vs eCryptfs).
  CPU Efficiency: ≈1.8x (vs EncFS), ≈2.2x (vs eCryptfs)

- Competitive even with AESNI, reaching up to ≈32% gain (vs eCryptfs, webserver). However, CPU usage: up to ≈0.4x (vs eCryptfs, webserver)

Conclusions (cont.)

FAST'19

- Main contributions:

  → advantages of applying CTR mode in CFSs (generation, storage and management of nonces)

  → explore additional advantages of CTR mode (parallelization, speculative encryption and Encryption Context Management)

- WAESlib applied to CFSs (abstracts GPU processing complexity, successfully exploits CTR mode, allows to create different techniques when using the encryption contexts)

Conclusions (cont.)

- GPU processing: significant increases in throughput (including small requests) and more efficient CPU utilization in environments where processors do not support the acceleration of cryptographic functions (or use of other ciphers)

- Future work:

  → performance analysis with actual loads
  (better testing / creating new techniques with context pools)

  → extend to other accelerators, multicore or heterogeneous

  → explore CTR / encryption in GPU (WAESlib) with kernel space client (e.g. dm-crypt, Crypto-API Linux FS client)

Thank you!

Questions?

Backup slide

Amount of time to CTR counter “wraparround”:

suppose a (current) 200 Gbps encryption capacity: $2 * 10^{11}$ bps
AES block size: 128 bits
Time for encrypting 1 AES block: $128 / 2 * 10^{11} = 6,4 * 10^{-10}$ s (0,64 ns)
Number of possible nonces: $2^{128} = 3,4 * 10^{38}$ blocos
Time to uniquely cypher all blocks: $(3,4 * 10^{38}) * (6,4 * 10^{-10}) = 2,17 * 10^{29}$ s

Years to “wraparround” = $2,17 * 10^{29} / 31.536.000 = 6,88 * 10^{21}$ years

Suppose a BILION times faster machine: $2 * 10^{20}$ bps
Time for encrypting 1 AES block: $128 / 2 * 10^{20} = 6,4 * 10^{-19}$ s
Time to uniquely cypher all blocks: $(3,4 * 10^{38}) * (6,4 * 10^{-19}) = 2,17 * 10^{20}$ s

Years to “wraparround” = $2,17 * 10^{20} / 31.536.000 = 6,88 * 10^{12}$ years
that is: 6,88 trilion years!