Discussion on *Space-Efficient Block Storage Integrity*

Moderated by Sam Small
600.624 Advanced Network Security
March 11th, 2005

with slides by Vishal Kher

Agenda

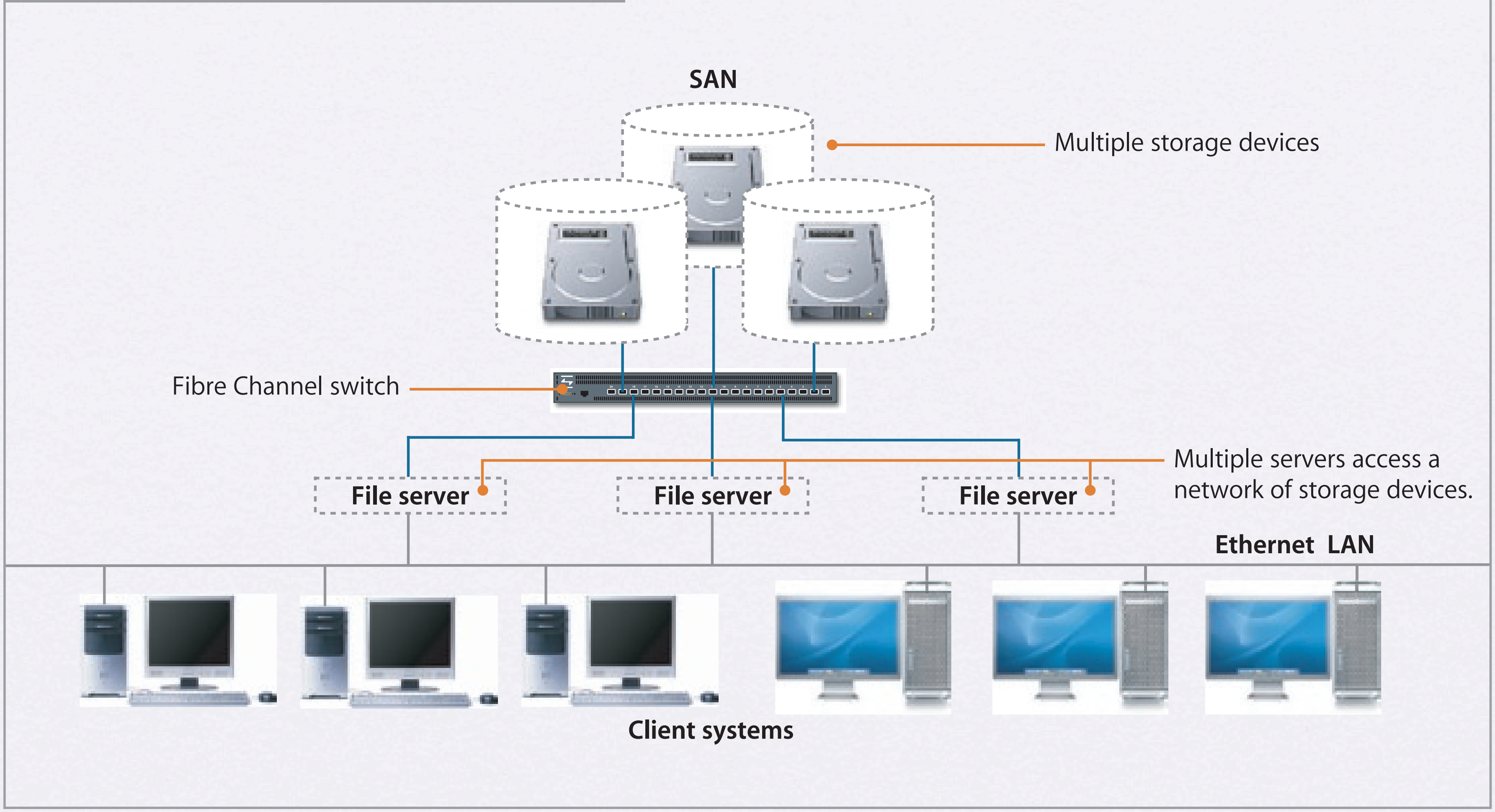
- More on the SAN model
- The Self-certifying File System (SFS)
- Provable Security
- Comments on the paper

Storage Area Networks (SAN)

- aggregates storage devices
- allows servers and client computers to access a single virtual storage entity
  - presents an interface to machines that is identical to that used by directly attached storage

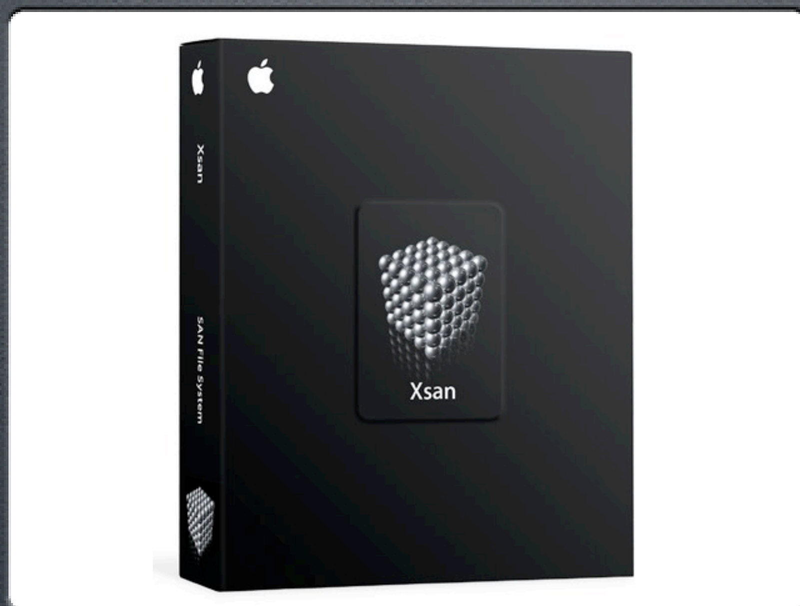
- Often use SCSI communication protocol
  - but not the SCSI low-level interface
- SAN: “Give me block 4000 from drive 5”
- NAS: “Give me /etc/passwd”

Storage Area Network



SAN Benefits

- Fast, concurrent file sharing
- Network-based storage management
- Eliminates single points of failure
- Topologies are flexible



Example: Xsan

Xsan

- Marketed towards:
  - professional video studios
  - data centers
  - high-performance clusters
- price point is significantly cheaper than similar products
  - has increased popularity of SANs

Self-certifying File System

- *Escaping the evils of centralized control with self-certifying pathnames*. SIGOPS, 1998. Mazieres, Kasshoek
- *Separating key management from file system security*. SOSP, 1999. Mazieres, Kasshoek, Kaminskv
- *Fast and secure read-only filesystem*. OSDI, 2000. Fu, Mazieres, Kasshoek

Motivation

- FS like NFS and AFS do span the Internet
  - They do not provide seamless file access
- Why is global file sharing (gfs) difficult?
  - Files are shared across administrative realms
  - Scale of Internet makes management a nightmare
  - Every realm might follow its own policy

SFS Goals

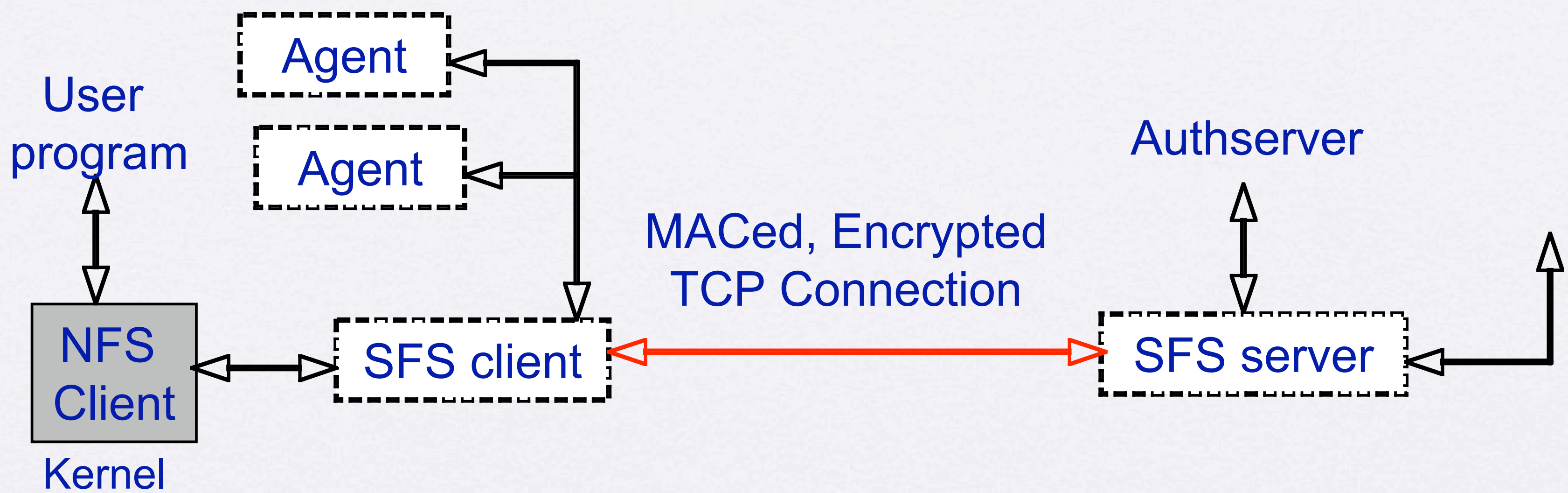
- Provide global file system image
- FS looks the same from every client machine
- No notion of administrative realm
- Servers grant access to users and not clients
- Separate key management from file system
- Various key management policies can co-exist

- Key management will not hinder setting up new servers
- Security Benefits
  - Authentication
  - Confidentiality and integrity of client-server communication
  - Versatility and modularity

Self-certifying Pathnames

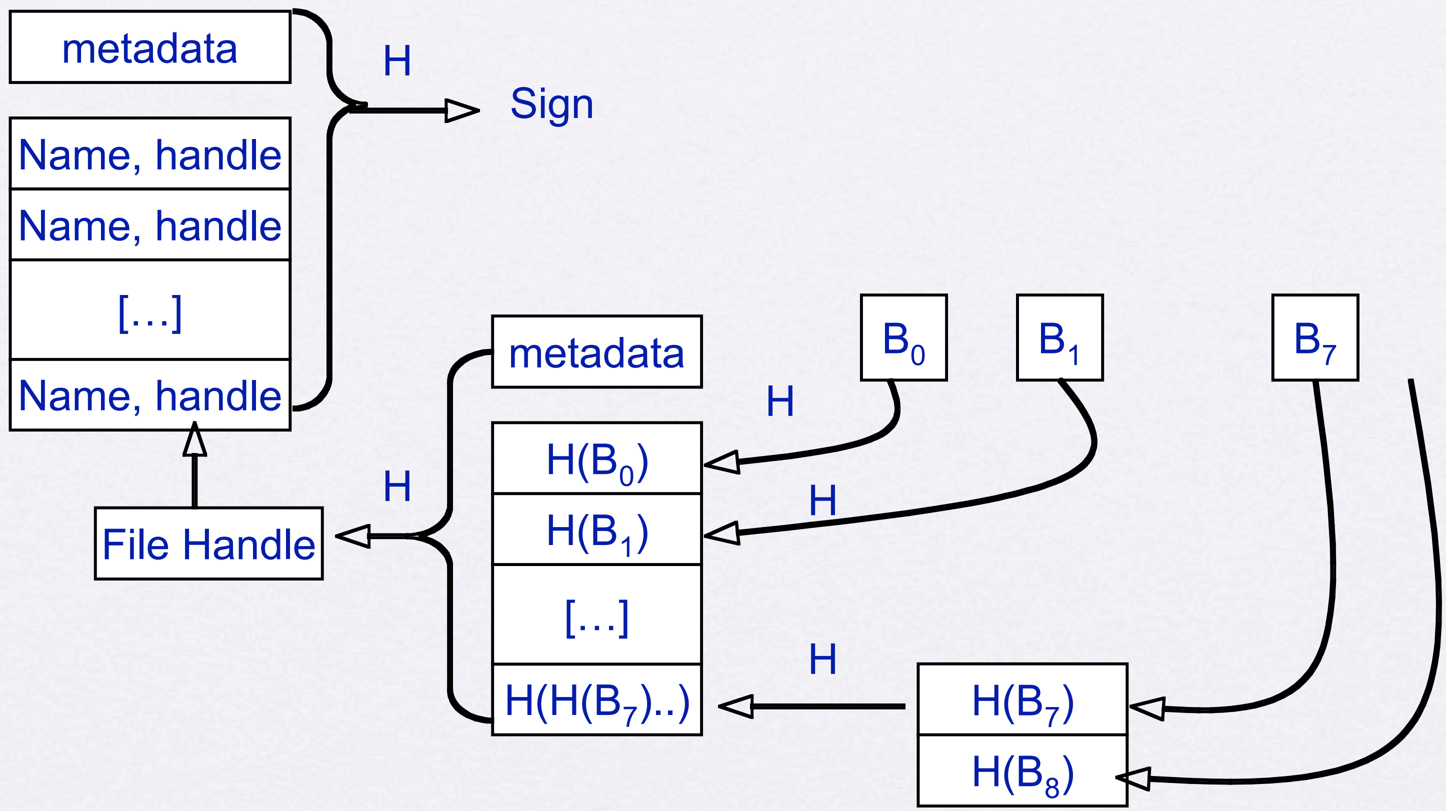
- Every SFS file system is accessible as:
  - /sfs/location:HostID
- HostID = ("Hostinfo", Location, PublicKey)
- Every pathname has a public key embedded in it

- /sfs/sfs.cs.jhu.edu:vefsdfa345474sfs35/foo
    - access file *foo* located on sfs.cs.jhu.edu
    - allows for automatic mounting


Agent
Agent
User
program
NFS
Client
Kernel
SFS client
MACed, Encrypted
TCP Connection
Authserver
SFS server

Recursive Hashing in SFS

- Each data block is hashed, becomes handle
- Handle used to lookup block in database
- Handles stored in file's inode
- Directories store <name, handle> pairs
- Directories and inodes hashed
- *rootfh* is hash of root directory's inode


metadata
H
Sign
Name, handle
Name, handle
[…]
Name, handle
File Handle
H
metadata
$H(B_0)$
$H(B_1)$
[…]
$H(H(B_7)..)$
H
B_0
B_1
B_7
H
H
$H(B_7)$
$H(B_8)$

Limitations

- Database update inefficient
  - Re-compute handles
  - Client must keep up with updates
- Verification
  - Traverse the tree to the root

Provable Security

- scheme constructions rely on cryptographic primitives
  - reduction argument: if A is secure and $A \Rightarrow B$, then B is secure. if B is not secure and $A \Rightarrow B$, then A is not secure
- the most ideal block cipher is a family of random permutations P, indexed by keys

Hazards

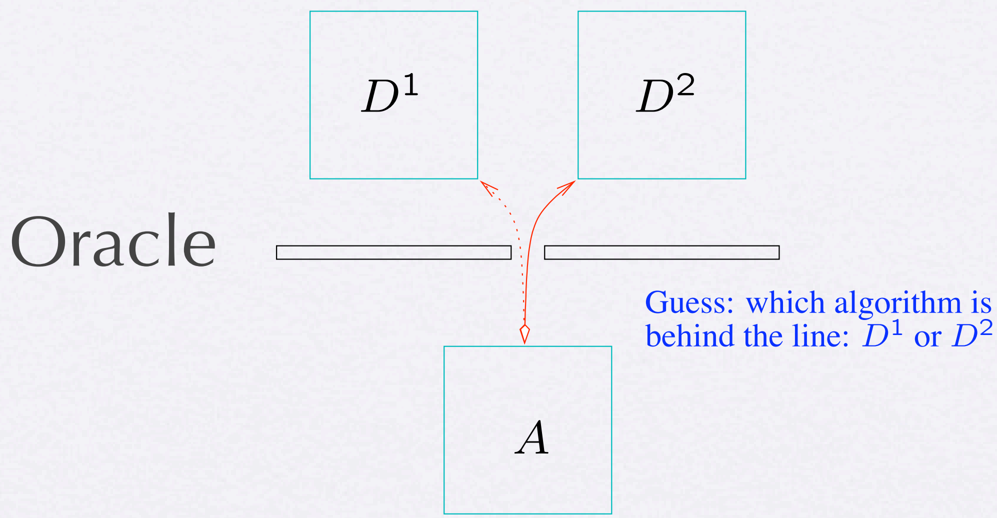
- Implementing P requires a database of $|P| \geq 2^{64}$
- Inefficient and impractical

Computational Security

- **unconditional security**: functions are random, bitstrings are random
- **computational security**: functions *seem* random, bitstrings *seems* random
  - to an adversary with limited resources
    - resources are usually bound by a polynomial Turing machine

- Instead of *P*, we use a pseudo-random permutation (PRP)
    - looks like a random permutation to a poly-bound adversary
    - what do we mean by saying that a PRP “looks” like a RP?

Oracle Model



PRP Definition

Definition. We say that E is an (q, t, ε)-*secure PRP* if for any algorithm that spends at most t steps (in some well-defined machine model), queries the oracle at most q times, has the success probability $\leq \varepsilon$ of distinguishing E:

$$\mathsf{Succ}_f^{\mathsf{PRP}}(A) \leq \varepsilon \text{ for all } (t, q)\text{-machines } A \ .$$

Provable Security in this week's paper

- Tweakable encryption scheme reduces to the security of the underlying block cipher
- The authors' integrity scheme S1 reduces to the security of second pre-image resistance in hash functions
- S2 reduces to the second pre-image resistance, tweakable encryption, and the guarantee of a low false positive rate

Comments on the Paper

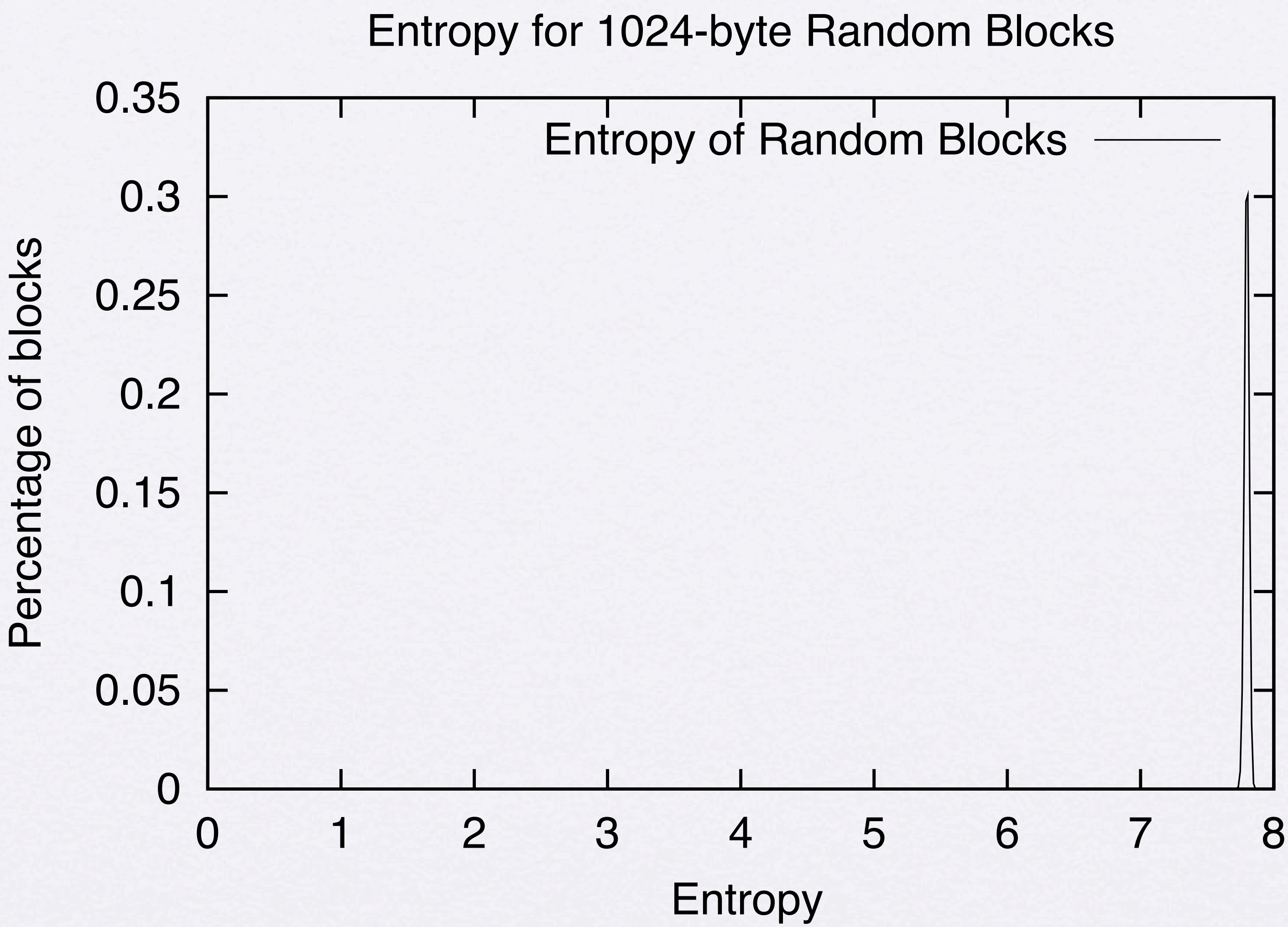


Figure 6. Entropy of 1024-byte Random Blocks

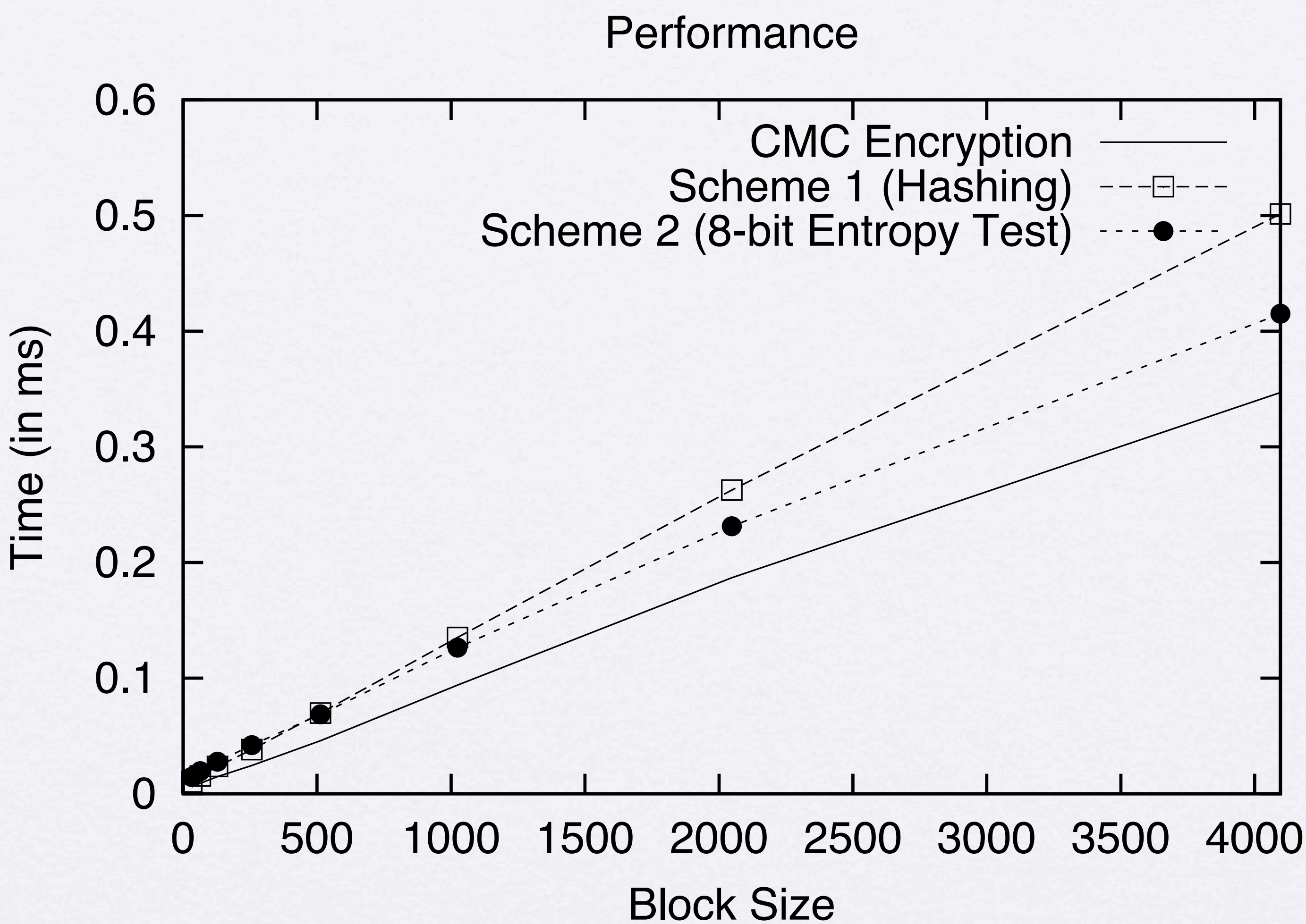


Figure 9. Performance Time for Different Storage Schemes

Storage for S_1	Storage for S_2	Storage for S_3
16.262 MB	0.022 MB	0.351 MB

Figure 11. Client Storage for the Three Schemes for One-Month Traces

Does Theorem 6.3 Hold?

- ... the frequency of any pattern in the sub-blocks of a single block should not exceed $p_i < 1/4$
  - is this assumption baseless? what is the justification?
  - this assumption is used to derive the formula for false negatives, the rate α

Skeptics

- “I don’t think this is an academic achievement as much as an exercise in performing an experiment for the sake of performing one”

Skeptics (2)

- Encryption does not always provide integrity

More on entropy

- Why do the authors consider two different lengths for their entropy tests? What are the advantages/disadvantages to using either?
- Is entropy the only metric that can be used to test for randomness in plaintext?

On test data

- Is this test set OK?
- Why don’t we use file access patterns from operational SANs?
- Shouldn’t we consider the entropy of file types rather than “all” files (e.g., WAV vs. MP3 vs. CPP)?

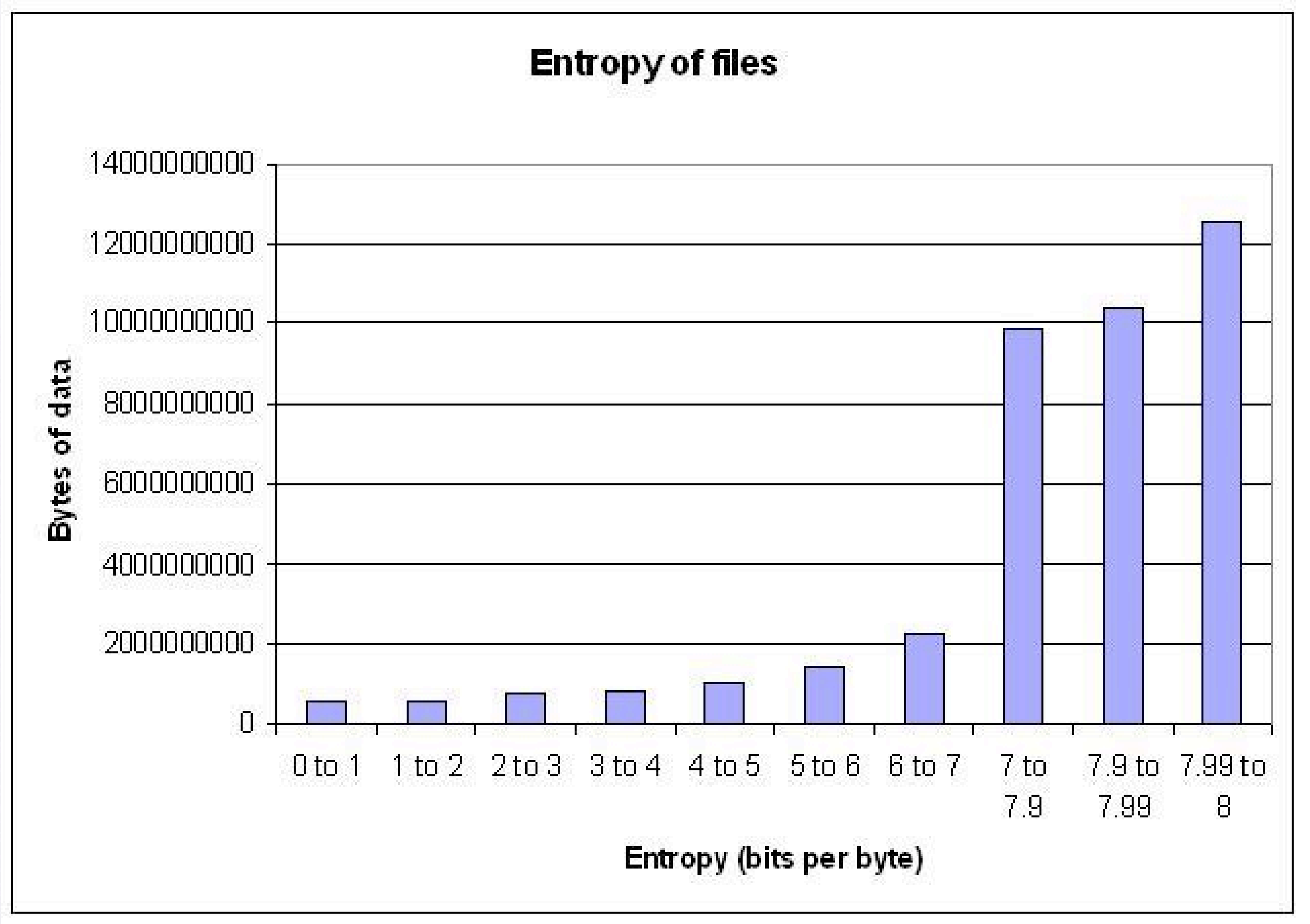
Entropy

- Looked at a bunch of files on my hard drive
  - Used ent at http://www.fourmilab.ch/random/
  - Analyzed 12.5 GB of files (24,897 files

Entropy by file format

- .c files: 5.06 (45,270,209 bytes / 2855 files)
- .h files: 4.69 (13,365,833 bytes / 1956 files)
- .vob files: 7.85 (7,384,492,032 bytes / 9 files)
- .php files: 5.12 (19,885,585 bytes / 1862 files)
- .java files: 5.00 (37,277,794 bytes / 1158 files)
- .mp3 files: 7.94 (487,454,293 bytes / 114 files)
- .wav files: 6.33 (271,408,960 bytes / 4 files)
- mis-decrypted file: 7.999658
- encrypted file (128-bit AES, CBC mode, base64 encoding removed): 7.999629

Cumulative distribution



Summary

- Lots of files have low entropy
- However, most of the larger files (hence, occupying more blocks) have higher entropy (mp3, vob, etc)
- My mis-decryption had an entropy of almost 8 - will they almost always be this high? Can the threshold be up around 7.99?
- What about chi square distribution?

Proposed Extensions

- Compression
- Message redundancy
- Multiple users