Design for a Decentralized Security System for Network Attached Storage

William Freeman & Ethan Miller
University of Maryland Baltimore County
wef@lts.ncsc.mil elm@csee.umbc.edu

Goal: secure distributed file system

- Existing file systems lack strong security
  - Data travels unencrypted on the network
  - Data is stored unencrypted on the disk
    - Backups & superuser can read data!
    - Intruder can read files off disk
  - Users can't verify that data is valid
    - Intruder can falsify data on disk
    - Intruder can put false data on the network
- Cryptography has become feasible for performance-critical systems
  - Use cryptographic techniques to secure the file system
  - Run algorithms on client (if possible) and a server CPU associated with each disk

Secure Network-Attached Disk (SNAD)

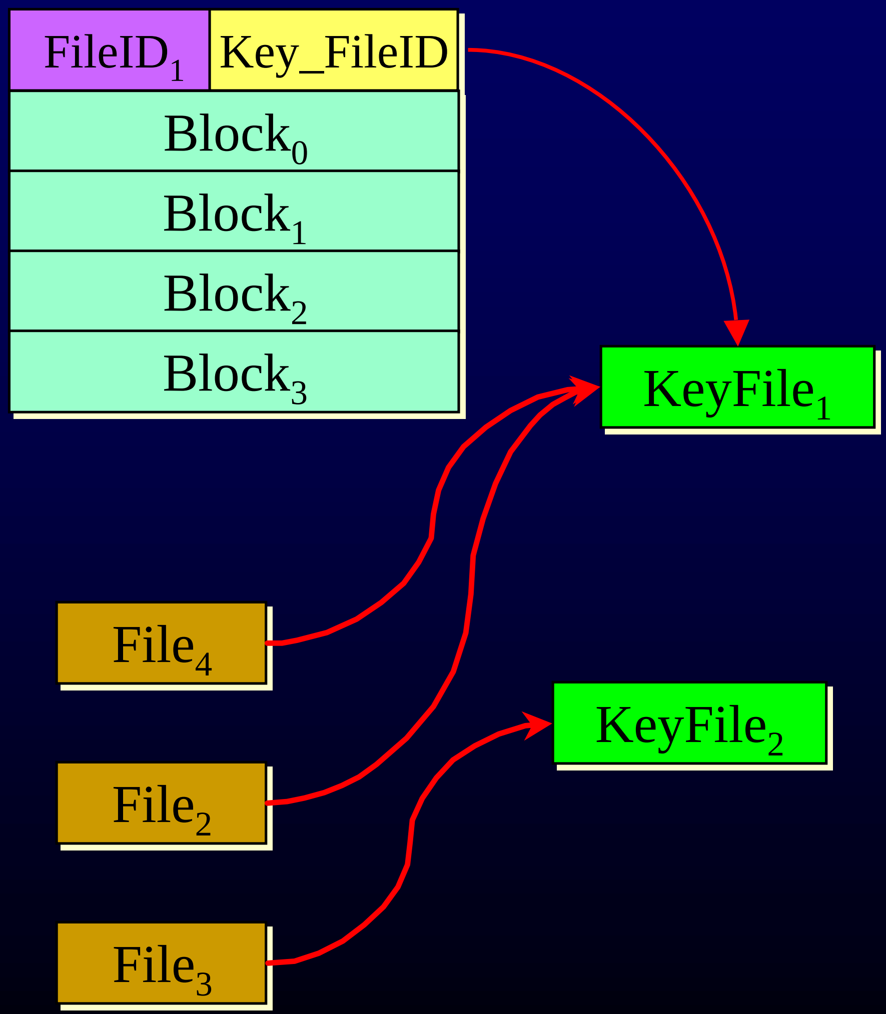
- Provides security mechanisms for an underlying file system
  - SNAD by itself doesn't handle
    - Storage management
    - File indexing structures
  - SNAD relies on the "base" file system for these
- Provides structures for managing security in the file system
  - Provides the necessary objects for securing the underlying file system
  - Where possible, uses existing security techniques

Basic concepts in SNAD

- For each block of data on disk, keep sufficient information to
  - Decode the data (at the client)
  - Validate the sender of the data
  - Ensure data integrity
- Use encryption to keep data secret on disk and in transit
  - Decryption only occurs at the client
  - Sufficient information to decrypt only available at client!
- Allow anyone to read a block!
- Restrict writing to authorized users
- Goal: prevent compromise of data
  - Impossible to protect against denial of service
  - Loss of data may occur => make sure it's noticed!

High-level system design

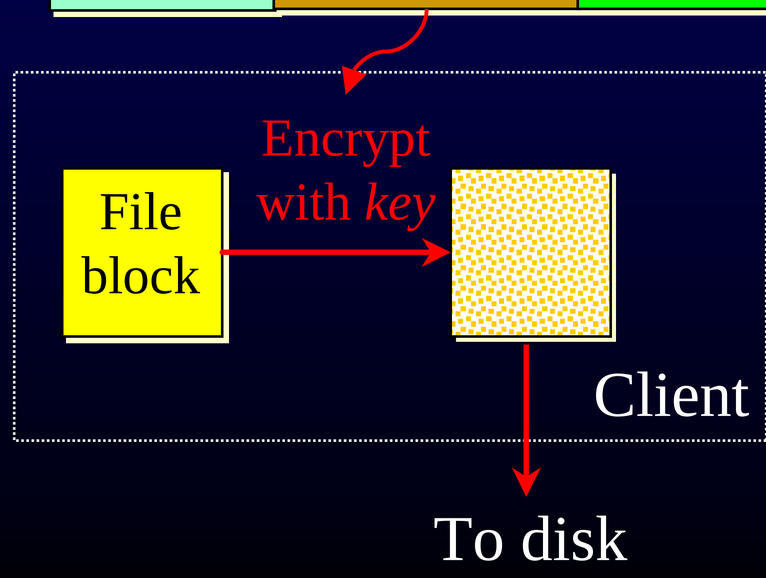
- File objects
  - Decoding information kept as a pointer to a key object
  - All blocks in a file encrypted with same key
  - Metadata kept as in a regular file system
- Certificate object
  - Verifies writers
  - Modification only necessary when new users / groups added
- Key objects
  - Store keys to decrypt files
  - Single key object can be used by multiple files



Key objects

- Key objects hold keys for encrypting & decrypting files
  - One key object per "unique" group
  - Many files can use one key object
- Key objects contain
  - File encryption key
    - Encrypted with each user's / group's public key
    - Disk doesn't have enough information to decrypt key
  - Permission bits for modifying the key object
- On file open, appropriate line of key object sent to user

KeyFileID		Owner
$UserID_0$	$E(Pub_0,Key)$	$Perm_0$
$UserID_1$	$E(Pub_1,Key)$	$Perm_1$
$UserID_2$	$E(Pub_2,Key)$	$Perm_2$
$UserID_3$	$E(Pub_3,Key)$	$Perm_3$



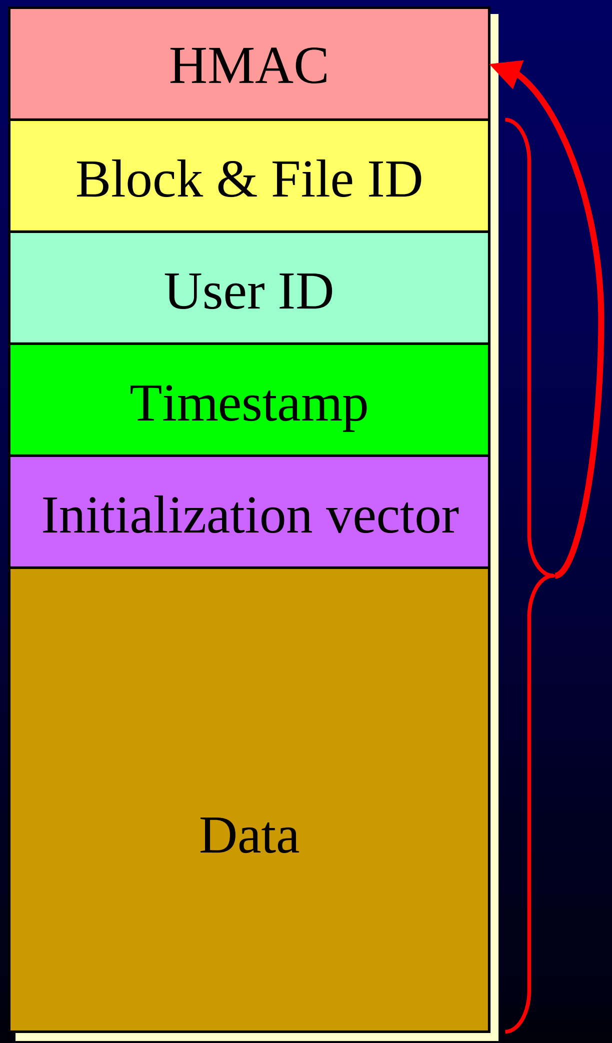
Certificate objects

- One certificate object per disk
- One entry per user / group
  - User ID
  - Public key
    - Convenience for creating key objects
    - Future use for stronger security schemes
  - HMAC (keyed hash) key
  - Timestamp
    - Prevents replay attacks for that user
    - Kept per-user to avoid problems of clock skew
- Certificate Object written securely...

CertificateFileID		Owner	
$UserID_0$	Pub_0	$HMAC\_KEY_0$	Timestamp
$UserID_1$	Pub_1	$HMAC\_KEY_1$	Timestamp
$UserID_2$	Pub_2	$HMAC\_KEY_2$	Timestamp
$UserID_3$	Pub_3	$HMAC\_KEY_3$	Timestamp

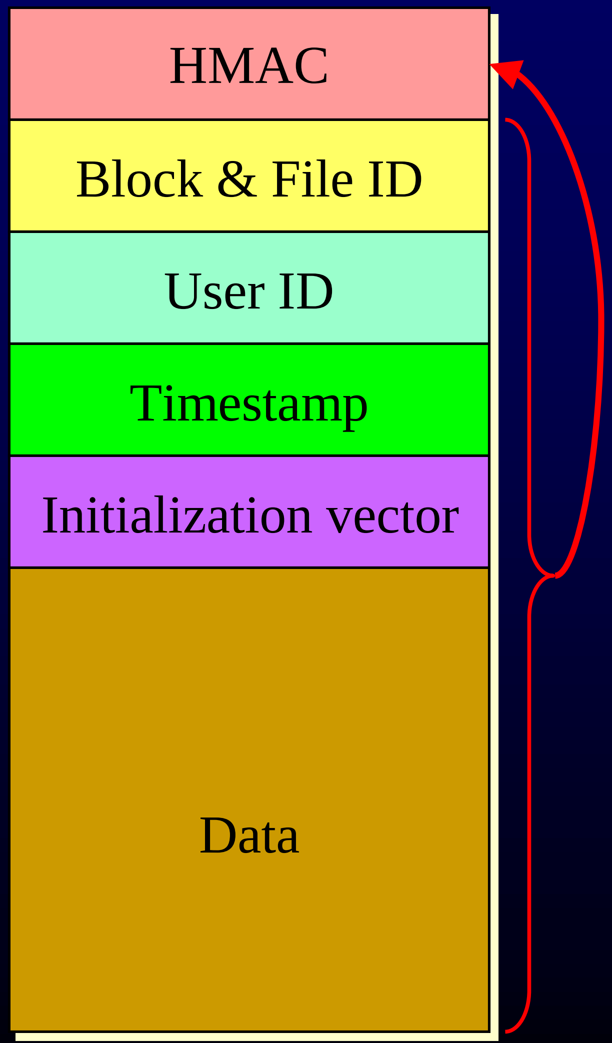
Read operation

- Client opens file
  - Receives entry from key object (may be cached)
  - Obtains file encryption key
- Client requests file block
- Disk reads & sends block
  - Generates HMAC using requesting user's HMAC key
  - Updates user's timestamp
- Client receives block
  - Uses HMAC to verify the integrity of the block
  - Uses key to decrypt the file block using RC5



Write operation

- Client opens file
  - Gets key object entry
  - Obtains file encryption key
- Client encrypts file block
- Client prepends timestamp, user ID, block & file ID
- Client calculates HMAC & sends block to disk
- Disk receives block
  - Verifies HMAC, timestamp, and write permission
  - Updates timestamp
  - Writes block to disk
- Metadata & data need not be stored together on disk...



Creating a file

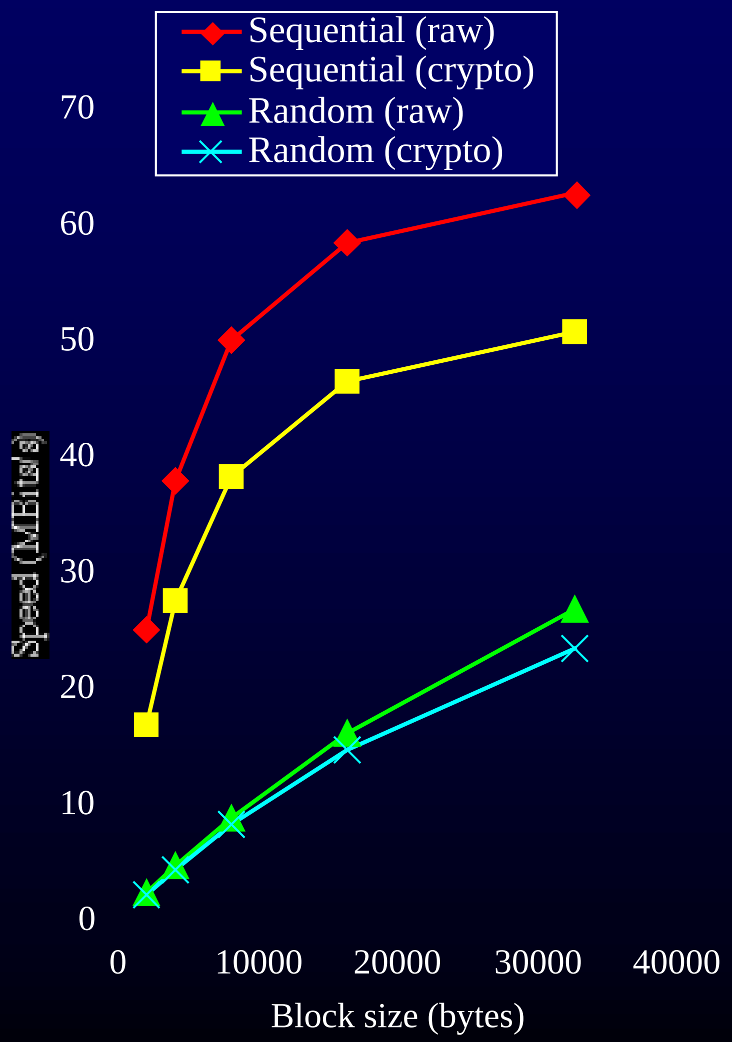
- Creating a file using an existing key object
  - New file simply points to the existing key object
  - All permissions & users listed in key object can use the new file
- Creating a file using a new key object
  - Client sends contents of key object to disk
    - Client needs only public keys of all users to be included in the new key object
    - Client generates a new RC5 key and encrypts it with the public keys of all users in the key object
  - File creation proceeds as above

File system performance

- Clients and servers used nearly identical hardware
  - ~350 MHz PowerPC
    - Faster chips available now
  - Fast Ethernet
  - 10000 RPM high-performance SCSI disks
- Real-time OS on both client & server
- One client, one server in tests
  - Multiple clients & servers being tested currently
    Performance appears to scale with additional clients
- Four different groups of experiments
  - Reads & writes
  - With and without encryption
  - Different block sizes for each set of experiments

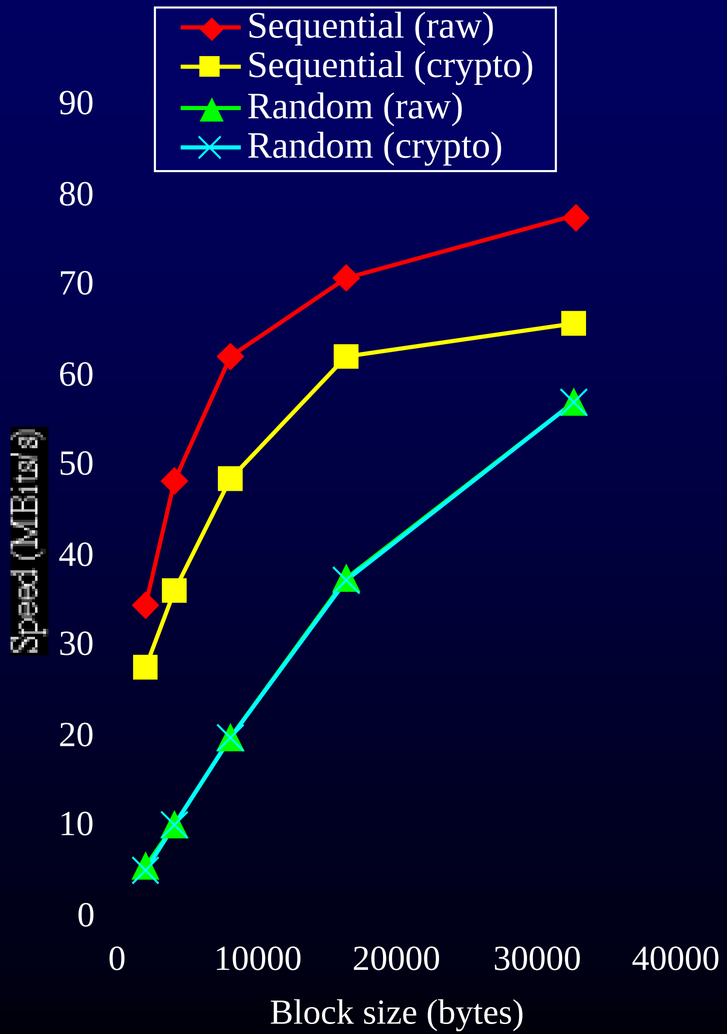
File system performance (reads)

- Reads require
  - On the client
    - 1 decrypt
    - 1 hash
  - On the disk: 1 hash
- Reads limited by client decryption bandwidth
  - Almost no performance difference for random I/O
  - Slight performance drop for sequential I/O
    - Can be solved by using more clients
    - Disk CPU not heavily loaded



File system performance (writes)

- Writes require
  - On the client
    - 1 encrypt
    - 1 hash
  - On the disk: 1 hash
- Writes limited by client encryption bandwidth
  - No performance difference for random I/O
  - Slight performance drop for sequential I/O
    - As with reads, more clients scale up bandwidth
    - Faster clients => less performance gap



Security issues

- Data has end-to-end protection
  - HMAC protects data in transmission
  - Use of HMAC & timestamp prevents spoofing
  - Compromised HMAC key => create undetectably fake data
    - Solve problem using digital signature of HMAC value
    - Requires more CPU time => slower
- Data on disk can’t be read
  - All data stored encrypted
  - Decryption keys not stored in the clear on the disk
- Denial of service still possible
  - Flood disks with read (or write) requests
  - Attack the disks with a sledgehammer...

Future work

- Issues related to key objects
  - Removing a user from a key object
    - Re-encrypt file?
    - Re-encrypt new data only?
  - Key escrow - include an escrowed “user” in every object
- Stronger protection on writes
  - Use HMAC signed by user’s private key
  - Signature too slow - speed improvements?
- Integration into an underlying file system
  - Currently planning to integrate into scalable file system being developed at UMBC
  - Integrate into other file systems?

Summary

- Strong security can be integrated into a scalable file system
  - CPU performance (even for inexpensive CPUs) is sufficient to allow the inclusion of encryption
  - Relatively few new structures are needed for encryption
- Performance with encryption isn't much worse than without

Given the dangers of leaving data in the clear on storage systems, can we afford not to use end-to-end encryption?