PORs: Proofs of Retrievability for Large Files

Ari Juels  Burt Kaliski
RSA Laboratories  EMC Innovation Network

ACM CCS Conference, 2007

presented by

Serkan Uzunbaz

Source: www.rsa.com/rsalabs/node.asp?id=3357
Outline

• Introduction and Motivation
• Some example approaches
• Overview of PORs
• The Sentinel POR-SYS overview
• The Sentinel POR-SYS details
• Security Analysis
• Drawbacks
• Conclusion and Discussion
The Subprime Mortgage Debacle

- Banks issued dubious mortgages to customers
- Banks packed mortgages as securities and sold them
- They “outsourced” risk
- But risk came home to roost...
  - E.g. Bank of America’s Q3 ’07 earnings dropped 32%
- The moral: Risk is passed around in complex, difficult-to-trace ways...
Introduction and Motivation

- Increasing popularity in outsourcing
  - Delegation of computing services to outside entities
- Software as a Service (Saas) is now a pillar in the internet-technology
- Storage is no exception to the outsourcing
  - Online data-backup services
  - Amazon Simple Storage Service (S3)
    - allows programmers to access data through web-service calls
  - Peer-to-peer data archiving [Cooper et al., 2002]
Introduction and Motivation

• Users and enterprises rely on diverse sets of data repositories;

  – they will require new forms of assurance of
    • integrity
    • accessibility of their data
One day, Alice’s machine crashes, so she contacts MegArchive…
Other financially motivated service degradation

• MegArchive moves Alice’s file to a slow disk array
  – File is there, but retrieval is unacceptably slow
• MegArchive uses cheap storage that degrades over time
• MegArchive throws away a portion of Alice’s file to save space
  – Most users don’t retrieve their backup files anyway
Some trivial approaches

• Simple replication
  – Pros: higher-assurance
  – Cons: unnecessarily and unsustainably high expense

• Data-dispersion scheme [Rabin, 89]
  – Share data across multiple repositories with minimum redundancy
  – Data is available given the integrity of k-out-of-n repositories
  – More efficient but still redundant
PORs: Proofs of Retrievability

- Alice determines that an archive possesses a file or data object $F$
  - kind of Proof of Knowledge (POK) but for a large file $F$
    - unlike a POK; neither the prover nor the verifier need actually knowledge of $F$
  - Archive: Prover
    - Alice: Verifier
  - **Goal**: To verify that archive do not delete or modify files prior to retrieval
    - without downloading the files
Some simple POR approaches

• The simple approach: Alice periodically downloads $F$
  – This is resource-intensive!

• A keyed hash function $h_k(F)$
  – Alice stores $r = h_k(F)$ with secret key $k$
  – Release $k$ and request $r$ from the Prover
  – High resource costs:
    • One hash for each check
    • Prover processes entire file each time

• What about spot-checking instead?
Spot checking: Preparation

Archive

Alice

\( f_2 \quad f_1 \quad f_3 \)

\( F \)
Spot checking: Verification

Archive

\[ \tilde{f}_2 \tilde{f}_1 \tilde{f}_3 \]

\[ \tilde{F} \]

Alice

\[ f_1 \neq f_2 \neq f_3 \]
Spot checking: Verification

Archive

\[ \tilde{f}_2 \quad \text{\(\times\)} \quad \tilde{f}_3 \]

\[ F \]

Alice

\[ f_1 \quad f_2 \quad f_3 \]
Spot checking

Pros:
• Alice needn’t download all of $F$
• Can detect large erasure / corruption

Cons:
• Alice must store chunks of $F$
• Can’t detect small erasure / corruption
MAC refinement: Preparation

Archive

\[ f_2 \quad f_1 \quad f_3 \]

\[ F \quad MAC_k[f_1] \]

\[ c_2 \quad c_1 \quad c_3 \]

Alice

\[ k \]
MAC refinement: Preparation

Archive

\[ f_2 \] \[ f_1 \] \[ f_3 \]

\[ F \] \[ \text{MAC}_k[f_1] \]

\[ c_2 \] \[ c_1 \] \[ c_3 \]

Alice

\( k \)
MAC refinement: Preparation

Archive

$F$

Alice

$k$
MAC refinement: Verification

Pros:
- Alice needn’t store any of $F$
- Can detect large erasure / corruption

Cons:
- Can’t detect small erasure / corruption
Error correcting code

• With ECC to encode $F \rightarrow F^*$, 
  big error in $F^*$ induces small error in $F$

• In effect, we can amplify errors in stored file
ECC + MAC
[loosely like LIBBI ’03, NR ’06]

Pros:
• Alice needn’t store any of $F$—only key $k$
• Alice can detect even small corruption in $F$ (= large corruption in $F^*$)
ECC + MAC: Verification

**Example:**
- ECC corrects up to 10% corruption
- Alice checks 30 positions
- Each check detects corruption with probability $\geq .9$
- Alice detects insurmountable corruption with prob. $\geq 1 - .9^{30} > 95%$
A challenge

- Applying such an ECC to all of $F$ is impractical
- Instead, we can *stripe* the ECC

- We’ll need many stripes for efficiency
- But then a clever adversarial archive can corrupt *selectively*...
Selective corruption

- Adversary corrupts just one stripe $S$
- File is now irretrievable!
- Unless Alice checks $S$, she does not detect corruption
- If $S$ is tiny fraction of file, adversary escapes detection with high probability
Solution: Hide ECC stripes

- Do secret, randomized partitioning of $F$ into stripes
  - E.g. use secret key to generate pseudorandom permutation and then choose stripes sequentially
- Encrypt and permute parity bits
  - In paper, they permute and encrypt $F$ too, but not strictly necessary
- $F$ itself is untouched, i.e., intact within $F^*$!
- But adversary does not know where stripes are, so
- Adversary cannot feasibly target a stripe!
Another challenge:
How do we formally define a POR?

• Adversary may respond correctly to challenges, but deny service on retrieval!

• No global solution: Pattern / number of requests betrays nature of request

• A practical approach to definition:
  1. Assume memoryless adversary, e.g., software that can be rewound/reset
  2. Ensure that challenge interface is identical to retrieval interface, i.e., formal extraction model
     – Limit distinction to pattern / number of requests
The Sentinel POR-SYS
The Sentinel POR-SYS overview

• Setup Phase
  – Alice (Verifier V) encrypts the file F
  
  – She embeds sentinels in random positions in encrypted F; sentinels being randomly constructed check values
    • indistinguishable from other file blocks (encryption)
  
  – Let \( \tilde{F} \) denote the file F with its embedded sentinels
The Sentinel POR-SYS overview

• Verification Phase
  – To ensure that the archive has retained $F$
  – Alice specifies the positions in $\tilde{F}$ and asks the archive to return the corresponding sentinel values

• Security
  – Archive can not distinguish sentinels
  – If it deletes or modifies a substantial portion, it will change the sentinels with a high probability
  – To detect corruption in small portion of the file, the user applies error-correcting code
  – Alice also permutes the file to ensure that sentinels are randomly dispersed
The Sentinel POR-SYS details

• Basit unit of storage: $\ell$– bit blocks
  – cipher operates on $\ell$– bit blocks, sentinels are $\ell$– bit

• 1. Error Correction
  – Treat the file as k-block chunks
  – To each chunk, apply an (n, k, d) error correcting code
    • each chunk is expanded to n blocks
    • we have the ability to correct up to $d/2$ errors
The Sentinel POR-SYS details

• 2. Encryption
  – Symmetric key encryption
  – Blocks should be decrypted in isolation
    • Therefore we require the encryption operate independently on plaintext blocks
      – $\ell$– bit block cipher
      – strem cipher
The Sentinel POR-SYS details

• 3. Sentinel Creation
  – Compute a set of \( s \) sentinels \( \{a_1, a_2, a_3, \ldots a_s\} \)
    – \( a_w = f(k, w) \)
  – Use function \( f(k, w) \) to create \( \ell \)– bit sentinels where \( k \) is used as a seed

• 4. Permutation
  – Apply the permutation function \( g \) to the blocks

The output file is \( \tilde{F} \)
Challange-Response and Security

• Alice then challenges the archive with some random positions of the sentinels

• The archive responds with the corresponding blocks

• The security of the system depends on the number of sentinels per challenge
  – not on the total number of available sentinels
Drawbacks and Related research

• Implementation
  – $F$ may be too large to fit in main memory
  – Improvement: QoS guarantees

• File updates: PORs break!
  – Efficient fix?

• Extended models
  – E.g., publicly verifiable

• Algebraic approaches
  – Burns et al., CCS ’07 and other previous work [E.g., FB ’06]
Conclusion and Discussion

• In Sentinel POR-SYS
  – Small communication cost
  – Very few number of memory access for prover
  – Small amount of storage requirements for verifier

• Any questions?
  – Thanks for listening!