POTSHARDS: Secure Long-Term Storage Without Encryption

Mark W. Storer Kevin M. Greenan
*University of California, Santa Cruz*
Ethan L. Miller
Kaladhar Voruganti
*Network Appliance†*

*Presenters: Ilana Levitan, Koby Brandas*
Secure Long-Term Storage

Secure - Secrecy, Availability, Integrity

• Outside Adversaries.
• Inside Adversaries.
Secure Long-Term Storage

Secure - Secrecy, Availability, Integrity

• Outside Adversaries.
• Inside Adversaries.

Long-Term - Potentially indefinite data lifetimes.
Goal of Secure Long-Term Storage

Providing security for relatively static data with an indefinite lifetime.
Properties of Secure Long-Term Storage

• Data must be accessible to authorized only.
Properties of Secure Long-Term Storage

• Data must be accessible to authorized only.

• Data must be accessible to authorized within reasonable amount of time.
Properties of Secure Long-Term Storage

• Data must be accessible to authorized only.

• Data must be accessible to authorized within reasonable amount of time.

• Data must be accessible to authorized, even to those who might lack a specific key.
Properties of Secure Long-Term Storage

• Data must be accessible to authorized only.

• Data must be accessible to authorized within reasonable amount of time.

• Data must be accessible to authorized, even to those who might lack a specific key.

• Confirming the integrity of the data.
Why can’t we use encryption?

* File secrecy which rely on the explicit use of keyed encryption is OK ONLY for short term secrecy needs.

* Encryption is only computationally secure. Namely: $C = E_k(M)$.
  [Computers can try to calculate this].

* There is a struggle between cryptography and cryptanalysis. For example, DES cipher considered secure in 1976, but in 1999 DES message could be cracked in under a day!!!

* Adversary who have an encrypted archive need only wait for cryptanalysis techniques to catch up to the encryption used at that time.
Problems in Current Models:

* **Venti, Elephant**:  
  - name blocks based on a secure hash of their data  
  - easy way to verify the content of a block against its name  
  - There is an encryption, using standard encryption algorithm -> Focus on the near term time scale.

* **Glacier**: Has no security & can’t save long – term archival storage

* **LOCKSS**:  
  - provide long term storage for public information  
  - No file secrecy  
  - Ensure the correctness of the data in the archives.
Secrecy via Splitting

secret sharing refers to any method for distributing a secret among a group of participants, each of which allocates a share of the secret. The secret can only be reconstructed when the shares are combined together; individual shares are of no use on their own.
Secrecy via Splitting

Let $k$ and $n$ be positive integers, $k \leq n$. A - $(k,n)$ threshold scheme is a method of sharing a secret $S$ among a set of $n$ participants in such a way that any $k$ participants can compute the value of the secret, but no group of $k - 1$ or fewer can do so.
Simple Example

An \((n, n)\) splitting scheme for a block B:
Simple Example

An \((n,n)\) splitting scheme for a block \(B\):

1. Randomly generate \(X_1, X_2, \ldots, X_{n-1}\)
Simple Example

An \((n, n)\) splitting scheme for a block \(B\):

1. Randomly generate \(X_1, X_2, ..., X_{n-1}\)
2. Choose \(X_n\) such that \(X_1 \oplus X_2 \oplus ... \oplus X_{n-1} \oplus X_n = B\)
Shamir's Secret Sharing - motivation
Shamir's Secret Sharing - motivation

How many **straight lines** go through this point?
Shamir's Secret Sharing - motivation
Shamir's Secret Sharing - motivation

How many straight lines go through this point?

For these lines, what can be the value of \( f(0) \)?
Shamir's Secret Sharing - motivation

How many **straight lines** go through these two points?
Shamir's Secret Sharing - motivation

How many \textit{straight lines} go through these two point?

Only one: f(x)=9-x.
Shamir's Secret Sharing - motivation

How many **straight lines** go through these two points?

For these lines, what can be the value of $f(0)$?
Shamir's Secret Sharing - motivation

How many **straight lines** go through these two points?

For these lines, what can be the value of $f(0)$?
Shamir's Secret Sharing - motivation

• Two points determine one straight line.
Shamir's Secret Sharing - motivation

- **Two** points determine one *straight line*.

- Given **one** point of a line $f$, $f(0)$ can be anything.

  Given **two** point of a line $f$, $f(0)$ can be only one value.
**Shamir's Secret Sharing - motivation**

Alice wants to split a secret between Bob and Carl.

The secret is 6.

Alice chooses a secret line $f$ such that $f(0)$ is 6 and gives Bob and Carl each one other point of the line.
Shamir's Secret Sharing - motivation

The secret is 6.

The secret line is $f(x) = 6 - 0.5x$. 
Shamir's Secret Sharing - motivation

Send two points

One for Bob, and one for Carl.
Shamir's Secret Sharing - motivation

Send two points

One for Bob, and one for Carl.

These points are the

shares of the secret.
Shamir's Secret Sharing - motivation

What does Bob know about the secret on his own?
Shamir's Secret Sharing - motivation

What does **Carl** know about the secret on his own?

Carl knows $f(8) = 2$
Shamir's Secret Sharing - motivation

What do Bob and Carl know about the secret together?
Shamir's Secret Sharing - motivation

Only **one straight line** through two points.
Shamir's Secret Sharing - motivation

Alice gives another point to Dave.

Who can retrieve the secret?
Shamir's Secret Sharing - motivation

Alice gives another point to Dave.

Who can retrieve the secret?

Any two points are sufficient!
Shamir's Secret Sharing - motivation

• **Two** points determine one **straight line**.

• Given **one** point of a line \( f \), \( f(0) \) can be anything.

  Given **two** points of a line \( f \), \( f(0) \) can be only one value.

• Use straight lines to **share** a secret over \( n \) participants:
  require more than **1** share to retrieve it.
Shamir's Secret Sharing

Split a secret $S$ into $n$ shares, such that

- any combination of $\leq L$ shares cannot learn the secret.
- any combination of $> L$ shares learns the secret.
Shamir's Secret Sharing

Split a secret $S$ into $n$ shares, such that

- any combination of $\leq L$ shares cannot learn the secret.
- any combination of $> L$ shares learns the secret.

How?
Shamir's Secret Sharing

Split a secret $S$ into $n$ shares, such that

- any combination of $\leq L$ shares cannot learn the secret.
- any combination of $> L$ shares learns the secret.

How?

Construct a degree-$L$ polynomial $f$ such that $f(0)=S$ and computes: $\text{share}_1=f(1)$, $\text{share}_2=f(2)$, $\ldots$, $\text{share}(n)=f(n)$. 
Shamir's Secret Sharing

Every \( L+1 \) shares can build the polynomial by computing Lagrange basis polynomials.

Therefore, we can retrieve the secret by \( L+1 \) shares.
Example - Shamir's Secret Sharing

Suppose that our secret is 1234.
Example - Shamir's Secret Sharing

Suppose that our secret is 1234.

We wish to divide the secret into 6 parts ( \( n = 6 \) ), where any subset of 3 parts ( \( k = 3 \) ), is sufficient to reconstruct the secret.
Example - Shamir's Secret Sharing

Suppose that our secret is 1234.
We wish to divide the secret into 6 parts ( \( n = 6 \) ), where any subset of 3 parts ( \( k = 3 \) ), is sufficient to reconstruct the secret.
At random we obtain two numbers: 166 and 94.
So, \( a_0 = 1234, \quad a_1 = 166, \quad a_2 = 94 \)
Example - Shamir's Secret Sharing

Suppose that our secret is 1234.

We wish to divide the secret into 6 parts (\( n = 6 \)), where any subset of 3 parts (\( k = 3 \)), is sufficient to reconstruct the secret.

At random we obtain two numbers: 166 and 94.

So, \( a_0 = 1234, \ a_1 = 166, \ a_2 = 94 \)

Our polynomial to produce secret shares (points) is therefore:

\[
f(x) = 1234 + 166 \cdot x + 94 \cdot x^2
\]
Example - Shamir's Secret Sharing

We construct 6 points $D_{x-1} = (x, f(x))$ from the polynomial:

$D_0 = (1, 1494)$

$D_1 = (2, 1942)$

$D_2 = (3, 2578)$

$D_3 = (4, 3402)$

$D_4 = (5, 4414)$

$D_5 = (6, 5614)$
Example - Shamir's Secret Sharing

We construct 6 points \( D_{x-1} = (x, f(x)) \) from the polynomial:

\[
D_0 = (1, 1494) \\
D_1 = (2, 1942) \\
D_2 = (3, 2578) \\
D_3 = (4, 3402) \\
D_4 = (5, 4414) \\
D_5 = (6, 5614)
\]

Why didn’t we use \((0, f(0))\)?
Example - Shamir's Secret Sharing

We give each participant a different single point.

In order to reconstruct the secret any 3 points will be enough.

Let us consider:

\[(x_0, y_0) = (2, 1942)\]
\[(x_1, y_1) = (4, 3402)\]
\[(x_2, y_2) = (5, 4414)\]
Example - Shamir's Secret Sharing

We will compute Lagrange basis polynomials:

\[
\begin{align*}
  l_0 &= \frac{x-x_1}{x_0-x_1} \cdot \frac{x-x_2}{x_0-x_2} = \frac{x-4}{2-4} \cdot \frac{x-5}{2-5} = \frac{1}{6} \cdot x^2 - \frac{3}{2} \cdot x + \frac{10}{3} \\
  l_1 &= \frac{x-x_0}{x_1-x_0} \cdot \frac{x-x_2}{x_1-x_2} = \frac{x-4}{4-2} \cdot \frac{x-5}{4-5} = -\frac{1}{2} \cdot x^2 + \frac{7}{2} \cdot x - 5 \\
  l_2 &= \frac{x-x_0}{x_2-x_0} \cdot \frac{x-x_1}{x_2-x_1} = \frac{x-2}{5-2} \cdot \frac{x-4}{5-4} = \frac{1}{3} \cdot x^2 - 2 \cdot x + \frac{8}{3}
\end{align*}
\]
Example - Shamir's Secret Sharing

Therefore,

\[ f(x) = \sum_{j=0}^{2} y_j \cdot l_j(x) = 1234 + 166 \cdot x + 94 \cdot x^2 \]
Example - Shamir's Secret Sharing

Therefore,

\[ f(x) = \sum_{j=0}^{2} y_j \cdot l_j(x) = 1234 + 166 \cdot x + 94 \cdot x^2 \]

and

\[ S = f(0) \]
Archives

• Has number of independent archives.
• Archives assist each other through distributed RAID techniques to protect the system from archive loss.
• Each archive exists within its own security domain. [Each of them can have the ability to protect it’s own data].
• Independent archives helps to exploit the benefits of geographic diversity in physical archive locations. [The story of Twin towers]
Data transformation:

- Transforms *objects* into a set of secure shards which are distributed to the archives.
- Can fulfill request from either a single client / many clients.
- Runs on a system separate from the archives on which the shard reside
- The data reach the archive in a secure form.
- The process can be done in parallel.
Data Entities

**Objects:**
- contain the data that users submit to the system at the beginning.

```
<table>
<thead>
<tr>
<th>hash</th>
<th>objectID</th>
<th>data</th>
</tr>
</thead>
</table>
```

**Shards:**
- The pieces actually stored in the achieve. [split 2 is Shamir Split]

```
<table>
<thead>
<tr>
<th>shardID</th>
<th>apx. ptr.</th>
<th>split2 (fragment)</th>
</tr>
</thead>
</table>
```

**Fragment:**
- used within the data transformation component during the production of shards. [split 1 is XOR]

```
<table>
<thead>
<tr>
<th>hash</th>
<th>objectID</th>
<th>frag. ID</th>
<th>S</th>
<th>shard list</th>
<th>split1(object)</th>
</tr>
</thead>
</table>
```

Stores procedure:

1. POTSHARD splits user data into secure shards.
2. Shard ID returned to user
3. Shards are distributed to a number of achieves. [There is redundancy]
Stores procedure:

1. POTSHARD splits user data into secure shards.
2. Shard ID returned to user
3. Shards are distributed to a number of achieves. [There is redundancy]
Stores procedure:

1. POTSHARD splits user data into secure shards.
2. Shard ID returned to user
3. Shards are distributed to a number of achieves. [There is redundancy]
Stores procedure:

1. POTSHARD splits user data into secure shards.
2. Shard ID returned to user.
3. Shards are distributed to a number of achieves. [There is redundancy]
Read procedure:

1. User sends shard IDs and authentication info to POTSHARDS client
2. Client requests shards from achieves.
3. Client reconstitute the data from m shards.
4. Verify data using hash
5. Client send data to user
Read procedure:

1. User sends shard IDs and authentication info to POTSHARDS client
2. Client requests shards from achieves.
3. Client reconstitute the data from m shards.
4. Verify data using hash
5. Client send data to user
Read procedure:

1. User sends shard IDs and authentication info to POTSHARDS client
2. Client requests shards from achieves.
3. Client reconstitute the data from m shards.
4. Verify data using hash
5. Client send data to user
Read procedure:

1. User sends shard IDs and authentication info to POTSHARDS client
2. Client requests shards from achieves.
3. Client reconstitute the data from m shards.
4. Verify data using hash
5. Client send data to user
Read procedure:

1. User sends shard IDs and authentication info to POTSHARDS client
2. Client requests shards from achieves.
3. Client reconstitute the data from m shards.
4. Verify data using hash
5. Client send data to user
Shard’s naming:

- Shard ID has 128 unique 128 bit identifier: 40 first bit identify the client, 88 next bit Identify the data entity.

- population of region: The numbers of valid shards

- \[ \text{density} := \frac{\text{(popularity)}}{\text{total number of shards}} \]

- difficult to detect intruders [will be explained later]

- portion of the namespace is being wasted
User Index

- When shards are created, the *exact* names of the shards are returned to the user along with their archive location.

- **Exact pointers are not stored in the shards themselves. [They are not available to someone who tries to attack the archive].**

- relationship between shards, fragments, objects, and files is saved in the user Index. [In order to enable fast retrieval]

- User index can be maintained by the client application or by physical token.
User Index Structure

- The index for each user can be stored in POTSHARDS as a linked list of index pages

- Each user maintains his private index. Advantages:
  1. The compromise of a user index does not affect the security of other users’ data.
  2. The system does not know about the relationship between user’s shards.
The Transformation Component
The Transformation Component: Pre-Processing

Divides files into fixed-sized objects.
The Transformation Component: Secrecy Split
Takes an object and produces a set of fragments.
The Transformation Component: Secrecy Split

• Fragments are generated at the first level of secret splitting, which is tuned for secrecy. Using an XOR-based algorithm that produces fragments from an object.
The Transformation Component: Secrecy Split

- Fragments are generated at the first level of secret splitting, which is tuned for secrecy. Using an XOR-based algorithm that produces fragments from an object.

- To ensure security, the random data required for XOR splitting can be obtained through a physical process such as radio-active decay or thermal noise.
The Transformation Component: Availability Split

Takes a fragment and produces a tuple of shards
The Transformation Component:
Availability Split

• A tuple of shards is produced from a fragment.
The Transformation Component: Availability Split

- A tuple of shards is produced from a fragment.

- This split is tuned for availability which allows reconstitution in the event that an archive is down or unavailable when a request is made.
Why 2 Splittings?

• The two-levels of splitting can be viewed as a tree with an increased fan out compared to one level of splitting.
Why 2 Splittings?

- The two-levels of splitting can be viewed as a tree with an increased fan out compared to one level of splitting.
- It allows useful metadata to be stored with the fragments as this data will be kept secret by the second level of splitting.
The Transformation Component: Placement

Determines how to distribute the shards to the archives
Approximated pointers:
How to retrieve data?

After the transformation process completes, the user gets an index mapping the original data to its shards.

But, *What if the index is lost?*
Naive Solution:
Solution:

• Each shard has an approximate pointer to the next shard in the fragment, with the last shard pointing back to the first and completing cycle.

• We take the low order R bits, $R = 2^r$ of the next shard id, and hide the true value.

• “Following” the pointer involves retrieving many shards in order to confirm the data with hash (some of which might not exist).

Legitimate user:
Can retrieve his easily his data
How to detect an intruder?

• Intruder doesn’t know exactly which shards from the archive he need to steal.

• Intruder need to steal all the shards that the approximate pointer refer to + the shards they refer to and so on.... [some shards even does not exist]

• Each user has to bypass the authentication mechanisms of each archive

• Archive can identify the access pattern of a thief [obtain shards that does not exist]
Regenerate fragments

2 approaches:

1. Try every possible chain of length $m$ -> build fragment -> verify with hash

2. Try chains of length $n$ which represent a cycle -> build fragment -> verify with hash [also called: ring heuristic]
ring heuristic advantages

• Shamir secret splitting algorithm is computationally expensive.
• Ring heuristic reduce the number of failed reconstitution attempts

The number of cycles of length $n$ is lower than the number of paths of length $m$ [many paths of length $n$ do not make cycles].
User Index & Encryption Key

Similar:
- They both contain the information needed to rebuild the user’s data

Different:
- If the index is lost or damaged, it can be recovered from the data
- If a user can prove a legal right to data, he also can recover the data.
Archive Design

• POTSHARDS is handled by a set of independent archives that store shards
• Use RAID techniques in order to deal with data loss.
• Each block in the archive holds redundancy / shards.

- 3 Steps when shards arrive:
  1. A random block is chosen as the storage location of the shard.
  2. the shard is placed in the last available slot in the block.
  3. corresponding parity updates are sent to the proper archives.

The problem: there is a risk of information leakage during archive recovery!
Secure Archive Reconstruction

If an archive is lost, recovery proceeds as follows:

1. Choose an archive which has space and is not a member of the redundancy group being recovered.
2. Elect one archive to manage the rebuilding through a sequence of chained requests which proceed according to the diagram below.
Each archive can independently reconstruct a block of failed data without revealing any information about its data. 😊

Proof:

\[ \gamma = a' + b' + c' + d' \]
\[ \eta = (a + a') + (b + b') + (c + c') + (d + d') \]
\[ \gamma + \eta = a + b + c + d \]
Experiment 1:

(a) 2 of 3 split.

(b) 3 of 5 split.
**Experiment 2:**

<table>
<thead>
<tr>
<th>Name Space</th>
<th>Shards</th>
<th>False Rings</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bits</td>
<td>4190</td>
<td>24451</td>
<td>6715 sec</td>
</tr>
<tr>
<td>32 bits</td>
<td>4190</td>
<td>0</td>
<td>225 sec</td>
</tr>
<tr>
<td>Bad</td>
<td>good</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| • Takes a lot of space because of Shamir Splitting  
• High time cost (But not so critical: because write once, read maybe principle) - encryption faster | • Fully security  
• Without use of encryption  
• Doesn’t need to maintain keys. |
Conclusions

• POTSHARDS is designed for long-term security

• Secrecy without relying on computation based mechanisms

• The combination of secret splitting and approximate pointers forces an attacker to steal an exponential number of shares in order to reconstitute a single fragment of user data
Questions?