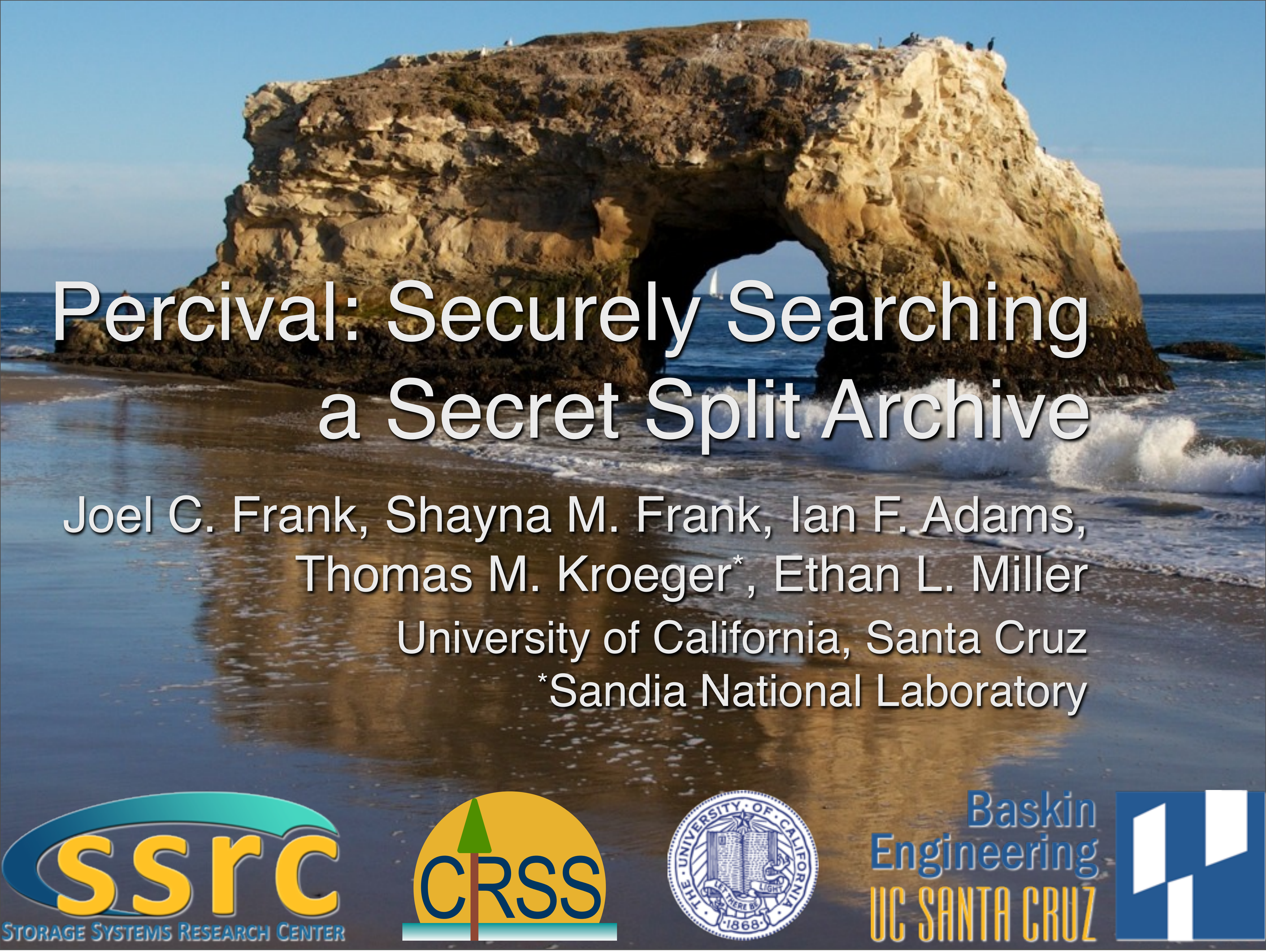


Percival: Securely Searching a Secret Split Archive

Joel C. Frank, Shayna M. Frank, Ian F. Adams,
Thomas M. Kroeger[*], Ethan L. Miller
University of California, Santa Cruz
[*]Sandia National Laboratory

Where does all that HPTS data go?

- We're generating over an exabyte of data per day!
  - Medica
  - Sensor
  - Person
  - "Small"

- Much of g time
  - "Games
  - Medica
  - Sensor g., climate & struct
  - Personal data → never want to delete *anything*
- A lot of data wants to live for a very long time...

The challenge

- We're storing a lot of data for a very long time
- This data can be very large: terabytes to petabytes per person / sensor network!

- Some (much?) of this data is very sensitive
  - Medical records
  - Corporate and government data
  - Sensor data: structural monitors, geo sensors (oil), etc.
- Attacks on this data can occur over a long period of time
  - Difficult to trust any one organization/site with it
- But we need to use this data, too!
  - Read old information
  - Search through stored data for useful information

The challenge, in brief

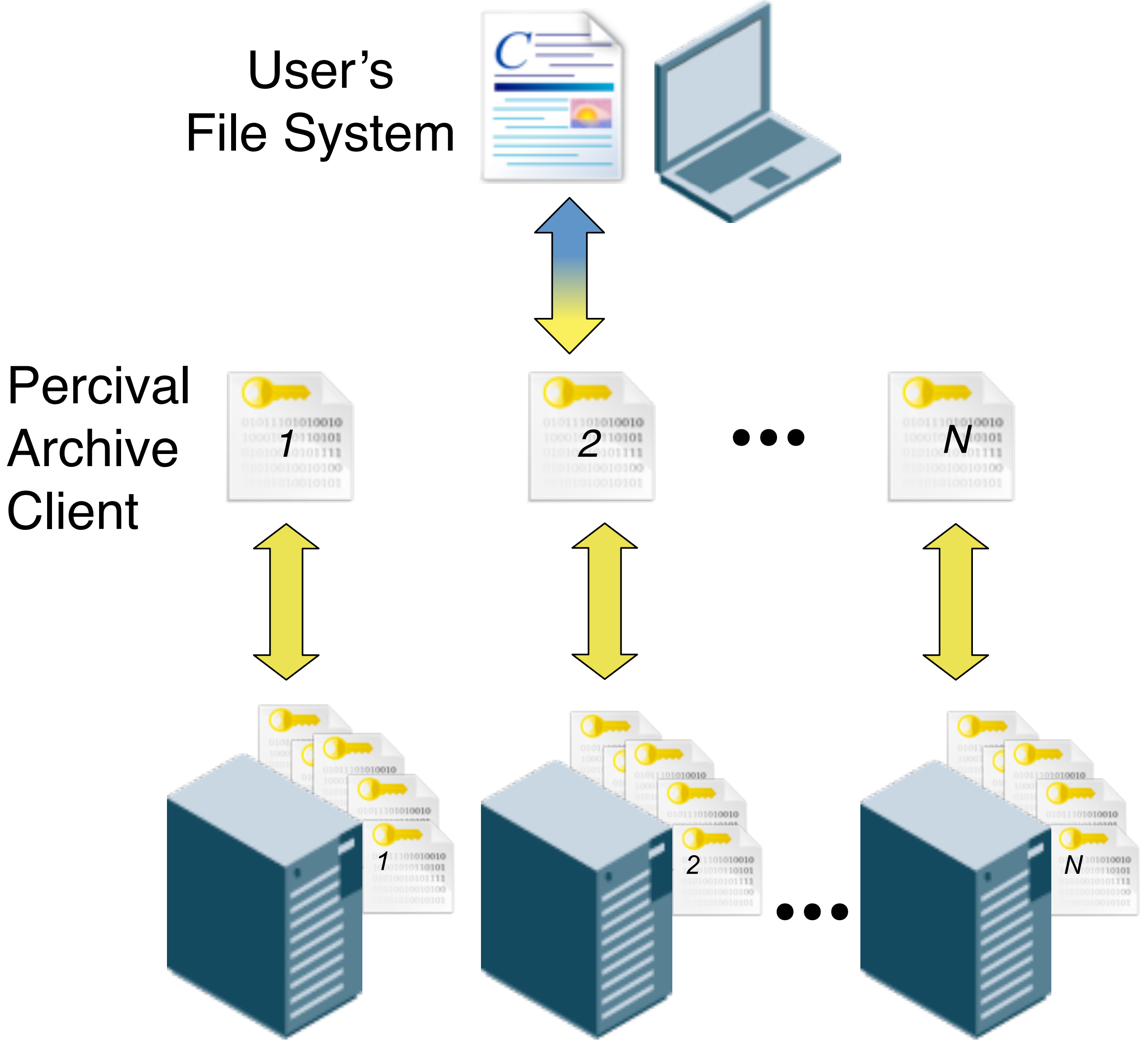
We need to reconcile our needs for privacy and utility for long-term data storage!

Threat model

- Attacker has
  - Unlimited computing power / storage
  - Unlimited time
  - Full access to any compromised repository
  - Ability to save past queries to compromised repositories
- Assume *M*-1 repositories have been compromised
- Compromise of authentication mechanism is outside of scope
  - But it’s straightforward to change authentication mechanism without touching all of the data!

Challenge 1: store the data

- Use secret sharing to generate shares
- Distribute shares to each of *N* archives
  - Need at least *M* shares to rebuild
  - *N* and *M* are configurable
- Require authorization to return data to requester
- POTSHARDS and other systems do this
  - Still need work to reduce overhead of splitting



Data Custodians
Distributed across multiple sites.

How does this help?

- No "information" at any one site
  - Must compromise *M* sites to gain any useful information
  - Difficult to do this undetectably
- Immune to key loss
  - Archives can pool their shares to allow rebuilding of data
- Immune to key / encryption algorithm compromise
  - Many forms of secret splitting are information-theoretically secure
  - No amount of NSA tomfoolery can weaken this...
- Difficult to identify "related" shares on different archives
  - Several approaches to make this possible

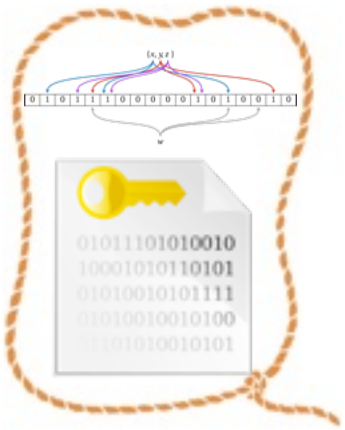
Challenge 2: search the data

- This level of security is great, but...
- How can we *find* anything in this system?
  - Want to prevent archive maintainers from figuring out what we're looking for
  - Want to prevent archive maintainers from identifying relationships between shares
- Client needs to tag shares on each archive
  - Tags need to be "nonsense" to archive
  - Tags need to be different across archives
  - Need to prevent (or at least reduce) possibility of correlating documents by monitoring search requests
  - But, tags need to be readily searchable (of course)

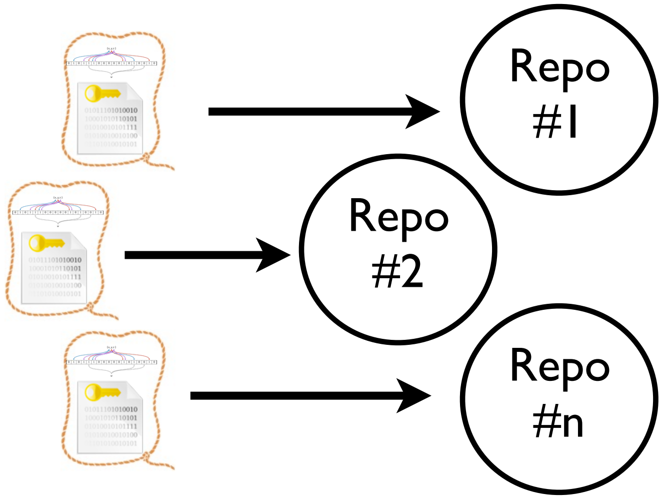
Percival overview

File Ingestion

For each file

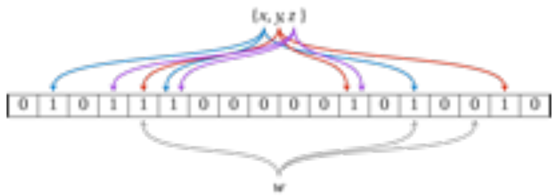


Generate a Bloom filter
for each share



Distribute these
bundles, one per
repository

Searching

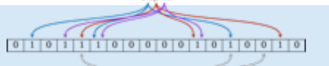
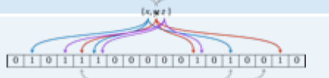
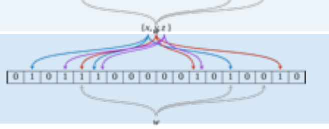


Create a Bloom filter
from the search
terms

Client Side | Server Side

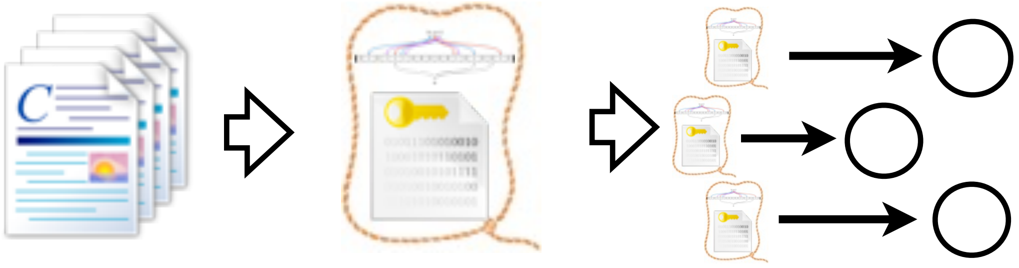
Compare it to each
share's filter, and
generate results map

Server Side | Client Side

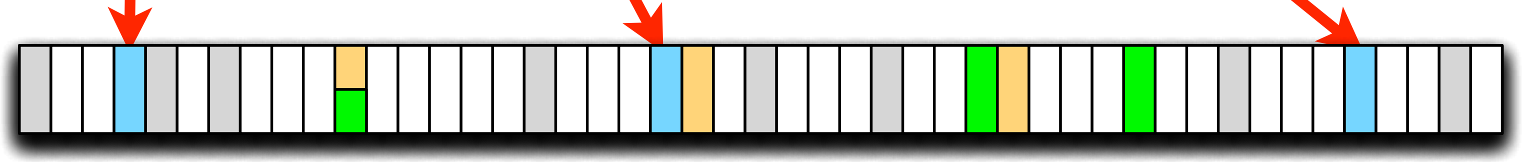
filename	results filter
file1	
file2	
file3	

Process the results

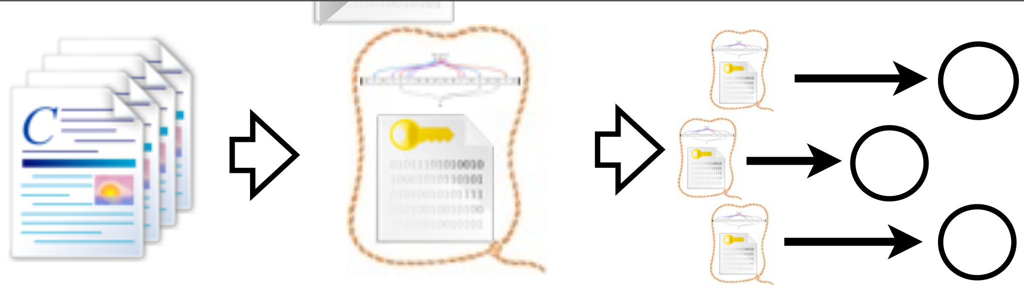
Design: ingestion



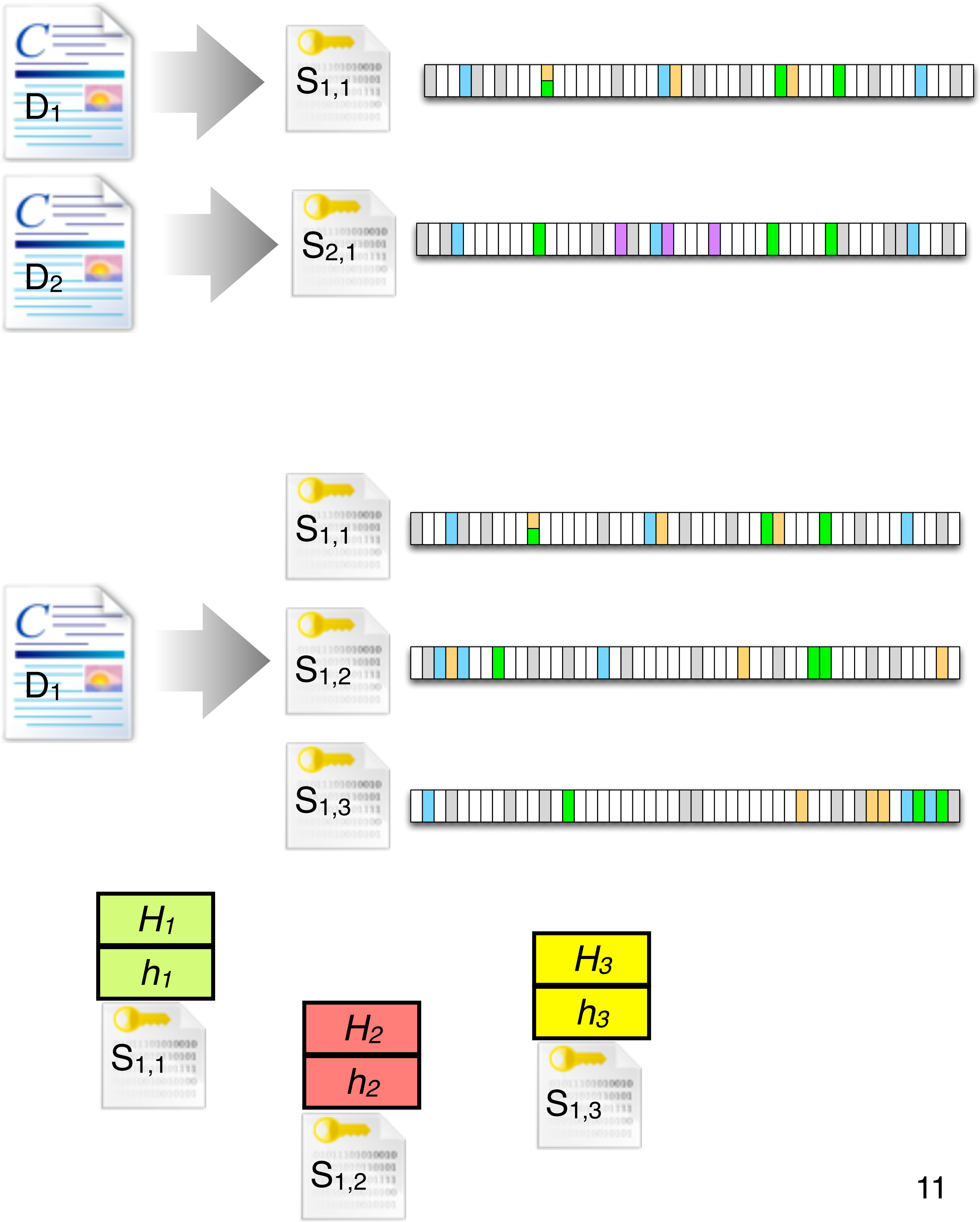
- Pre-index each share with a Bloom filter
  - Generate list of terms *W*
  - Combine each term, w_i, with the repository key, key_r
    $v_i = KeyedHash(w_i, key_r)$
  - Generate *k* locations using *k* hash functions of v_i and set the corresponding bits in the Bloom filter for *r*
- Problem: it may be possible to associate shares on *r* with the same bits set in the Bloom filter
- Solution: set randomly-selected bits in the Bloom filter for each share on each repository (chaff)
  - Obscures the relationship between set bits and terms
  - Increases the number of false positives



Design: ingestion



- Shares with similar terms still differ in Bloom filters
  - Amount of chaff is tunable—currently investigating tradeoffs
- Different Bloom filter for each repository
  - Difficult to correlate shares across repositories
- Add H_i, h_i to each share
  - H = hash($data$)
  - H_i = hash (H, key_r)
  - Share of H: h_i = split (H, i)



Design: search

filename
results filter
file1
file2
file3

Client

- Generate a search Bloom filter for each repository
- Send each Bloom filter and hit threshold to each repository

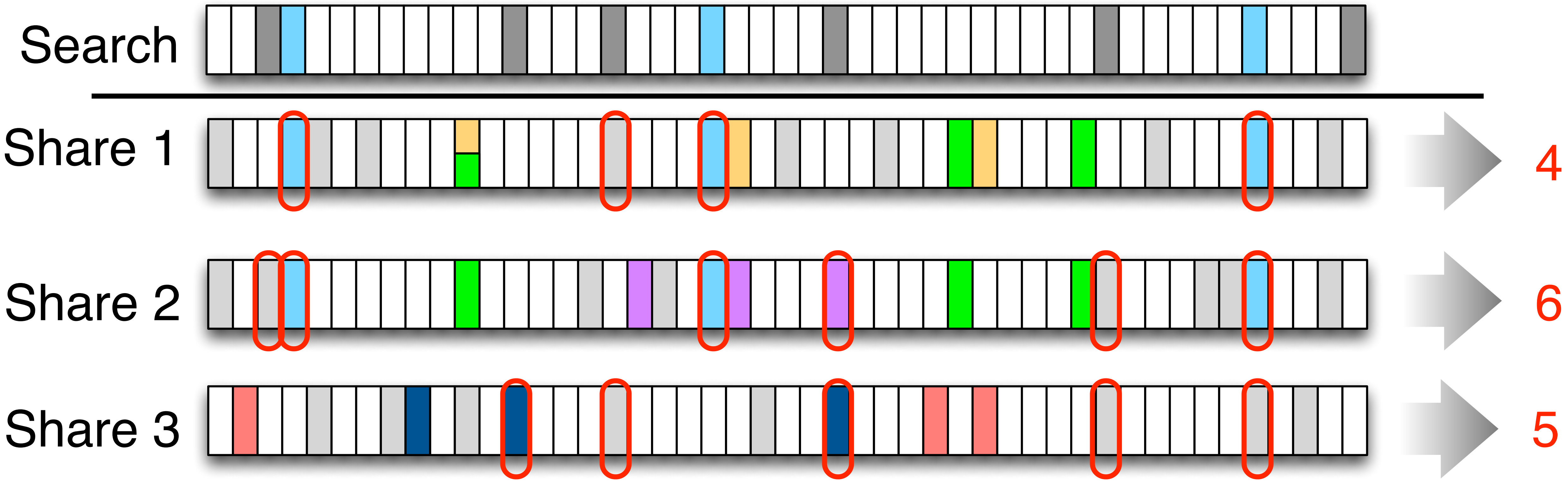
Server

- Calculate intersection for each share's Bloom filter
- Hit threshold met?
- Return list of shares that meet the threshold

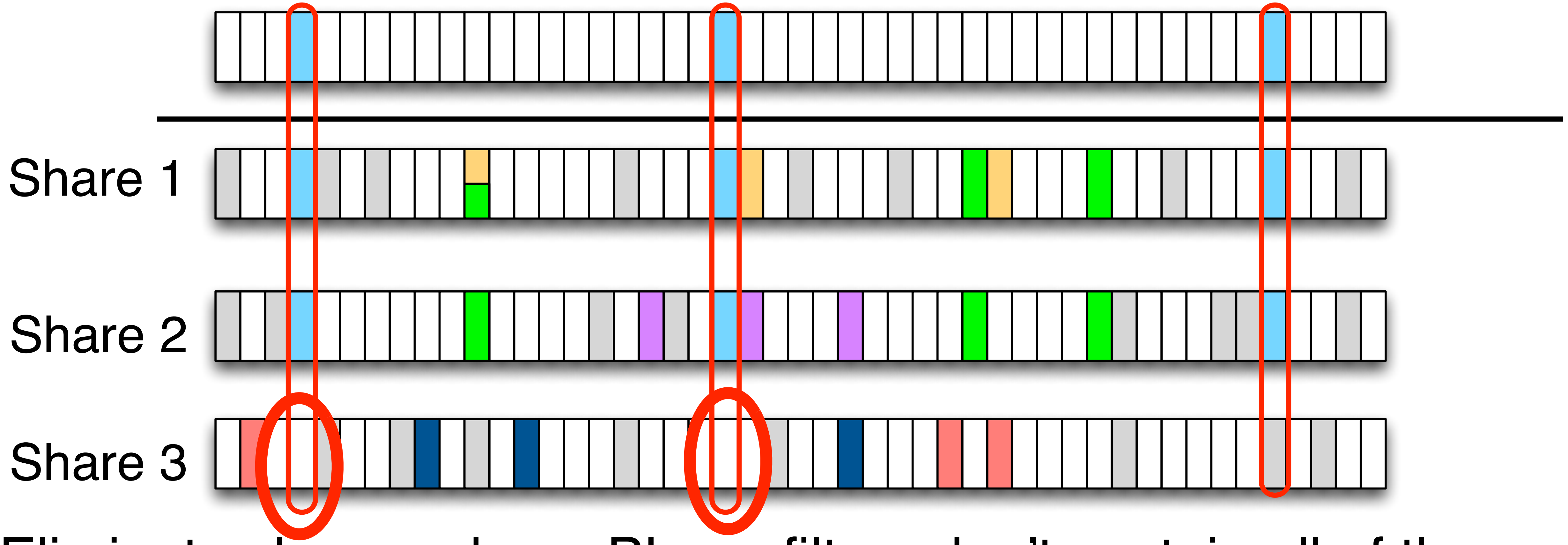
- Get results from each server
- Identify documents with shares in each result list
- Request shares from each repository

Search: using the Bloom filters

- Set *b* bits in search Bloom filter using same hash functions that were used when shares were stored
  - Use key_r to generate different filters for each repository
- Add chaff bits to search Bloom filter
  - Again, goal is to make correlating different searches more difficult
- Require archive to return all results with at least *b* bits that match
  - This contains a superset of desired results



Search: identifying results at the client



- Eliminate shares whose Bloom filters don't contain all of the "real" bits
- Try all combinations of shares, one from each repo
  - Reassemble the hash value from the split hashes
  - Verify reassembled value using key_r against keyed hash stored in one of the shares
- Request full shares to rebuild the desired data

Search: issues

- Is combinatoric reassembly slow?
  - Depends on the number of shares that pass the Bloom filter test
  - Typically not an issue with low false positive rates
  - Can become large for large share "width"

- Is use of Bloom filters slow or inefficient?
  - Can use techniques for faster searches
  - Can compress Bloom filters (especially results)
    - Results need only include bits that match the search

How secure is it?

- Data can’t be rebuilt without sufficient shares
  - Attempts to get large quantities of data from independent archives will raise suspicion
- What about targeted attacks?
  - Difficult to correlate searches across archives to identify related shares
  - Recombination is much harder without eliminating shares that don’t contain all search term bits
- Can attacker learn search terms?
  - Set bits are different for each archive
  - Set bits are obscured in both index and search filters
- Currently investigating *how* well this hides information...

Where are we now?

- Working on a prototype with Sandia National Labs
- Investigating tradeoffs in
  - Obfuscation of bit groups
    - Adjust filter size → loading → false hit rate
  - Methods to mitigate false hit rate
  - Methods to increase computational bounds to determine key_r
- Exploring long-term attacks that attempt to correlate searches, even with chaff on both ingest and search
- Working on better ways to split secrets more efficiently
- Rebuilding shares after an archive failure

Wrapping it up

- Long-term archives will be
  - Very large
  - Under constant threat from attacks
    - Lost encryption keys
    - Compromised keys
    - Outdated encryption
- But we need to support search and access!
- Combine secret split archives with searches using Bloom filters with chaff
  - Hides relationships between shares on a single archive
  - Hides relationships between shares across archives
  - Makes compromise much more difficult
- Still much to be done....

Questions?

Collaborators

Joel C. Frank

Shayna M. Frank

Ian F. Adams

Thomas M. Kroeger

http://www.ssrc.ucsc.edu/proj/archive.html