Proofs of Ownership in Remote Storage Systems

Shai Halevi (IBM)
Danny Harnik (IBM)
Benny Pinkas (Bar Ilan)
Alexandra Shulman-Peleg (IBM)

Cloud backup services

- Online file backup and synchronization is huge
- Mozy
  - Over one million customers and 50,000 business customers. Over 75 PetaByte stored.
- Dropbox
  - Over three million customers.
- And many more… many services geared towards enterprises

Screen shot of a backup process

- You can examine your backup history

MozyHome History

Start Time	Type	Duration	Result	F...	Size	Files Enco...	Size Enco...	Files Transfer...	Size Transfer...
14/05/2010 00:18	Manual ...	00:03:33	Success	2...	30.0 GB	1	7.9 MB	0	0 bytes
13/05/2010 23:56	Automa...	00:04:01	Success	2...	30.0 GB	5	990.9 KB	5	990.9 KB
13/05/2010 21:49	Automa...	00:03:02	Success	2...	30.0 GB	4	110.9 KB	4	110.9 KB
13/05/2010 19:30	Automa...	00:03:09	Success	2...	30.0 GB	6	848.4 KB	6	848.4 KB
13/05/2010 17:30	Automa...	00:02:06	Success	2...	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 15:30	Automa...	00:02:05	Success	2...	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 13:30	Automa...	00:02:12	Success	2...	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 11:29	Automa...	00:03:12	Success	2...	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 09:29	Automa...	00:02:10	Success	2...	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 07:29	Automa...	00:07:39	Success	2...	30.0 GB	29	26.7 MB	22	14.0 MB
12/05/2010 20:15	Automa...	00:06:36	Success	2...	30.0 GB	4	3.1 MB	4	3.1 MB
12/05/2010 18:15	Automa...	00:07:46	Success	2...	30.0 GB	5	4.5 MB	5	4.5 MB
12/05/2010 16:08	Automa...	00:04:08	Success	2...	30.0 GB	3	135.6 KB	3	135.6 KB
12/05/2010 14:08	Automa...	00:04:10	Success	2...	30.0 GB	2	23.6 KB	2	23.6 KB
12/05/2010 11:54	Automa...	00:09:32	Success	2...	30.0 GB	16	266.7 KB	16	266.7 KB
12/05/2010 09:37	Automa...	00:02:28	Success	2...	30.0 GB	0	0 bytes	0	0 bytes
12/05/2010 07:37	Automa...	00:13:41	Success	2...	30.0 GB	27	43.3 MB	26	19.1 MB
10/05/2010 13:07	Automa...	00:04:00	Success	2...	30.0 GB	18	3.1 MB	15	2.6 MB
10/05/2010 08:07	Automa...	00:02:50	Success	2...	30.1 GB	0	0 bytes	0	0 bytes
10/05/2010 05:58	Automa...	00:02:46	Success	2...	30.8 GB	0	0 bytes	0	0 bytes
10/05/2010 02:11	Automa...	03:45:24	Success	2...	30.8 GB	3	701.4 MB	3	701.4 MB
09/05/2010 23:03	Automa...	03:07:34	Success	2...	30.6 GB	6	453.7 MB	6	453.7 MB
09/05/2010 21:36	Automa...	00:01:50	CancelError0	0	0 bytes	0	0 bytes	0	0 bytes
09/05/2010 18:54	Automa...	00:03:14	Success	2...	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 16:54	Automa...	00:03:33	Success	2...	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 14:54	Automa...	00:07:06	Success	2...	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 12:54	Automa...	00:05:14	Success	2...	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 10:54	Automa...	00:03:43	Success	2...	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 08:54	Automa...	00:03:42	Success	2...	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 06:20	Automa...	00:04:18	Success	2...	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 03:27	Automa...	00:02:24	Success	2...	30.1 GB	3	4.1 KB	3	4.1 KB
08/05/2010 23:16	Automa...	00:04:03	Success	2...	30.1 GB	14	10.2 MB	12	555.7 KB
08/05/2010 20:05	Automa...	00:02:31	Success	2...	30.1 GB	0	0 bytes	0	0 bytes
08/05/2010 19:53	Manual ...	00:05:32	Success	2...	30.1 GB	1	3.1 MB	1	3.1 MB

File	Path	Patch Size	Encoding T...	Transfer Ti...	Transfer R...	Other Details
Copy of my presentation.p...	C:\Documents and Setting...	7.9 MB	00:00:00			File already on MozyHome servers

Copy of my presentation

File already on MozyHome servers

- But sometimes strange things happen…

MozyHome History

Start Time	Type	Duration	Result	F…	Size	Files Enco…	Size Enco…	Files Transfer…	Size Transfer…
14/05/2010 01:16	Manual …	00:02:20	Success	2…	30.4 GB	1	175.0 MB	0	0 bytes
14/05/2010 01:13	Manual …	00:02:23	Success	2…	30.2 GB	1	233.7 MB	0	0 bytes
14/05/2010 00:54	Manual …	00:02:51	Success	2…	30.0 GB	1	2.4 MB	1	2.4 MB
14/05/2010 00:47	Manual …	00:05:07	Success	2…	30.0 GB	1	106.6 KB	1	106.6 KB
14/05/2010 00:44	Manual …	00:02:13	Success	2…	30.0 GB	2	46.9 KB	2	46.9 KB
14/05/2010 00:40	Manual …	00:02:09	Success	2…	30.0 GB	2	164.0 KB	2	164.0 KB
14/05/2010 00:36	Manual …	00:02:37	Success	2…	30.0 GB	3	239.1 KB	3	239.1 KB
14/05/2010 00:18	Manual …	00:03:33	Success	2…	30.0 GB	1	7.9 MB	0	0 bytes
13/05/2010 23:56	Automa…	00:04:01	Success	2…	30.0 GB	5	990.9 KB	5	990.9 KB
13/05/2010 21:49	Automa…	00:03:02	Success	2…	30.0 GB	4	110.9 KB	4	110.9 KB
13/05/2010 19:30	Automa…	00:03:09	Success	2…	30.0 GB	6	848.4 KB	6	848.4 KB
13/05/2010 17:30	Automa…	00:02:06	Success	2…	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 15:30	Automa…	00:02:05	Success	2…	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 13:30	Automa…	00:02:12	Success	2…	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 11:29	Automa…	00:03:12	Success	2…	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 09:29	Automa…	00:02:10	Success	2…	30.0 GB	0	0 bytes	0	0 bytes
13/05/2010 07:29	Automa…	00:07:39	Success	2…	30.0 GB	29	26.7 MB	22	14.0 MB
12/05/2010 20:15	Automa…	00:06:36	Success	2…	30.0 GB	4	3.1 MB	4	3.1 MB
12/05/2010 18:15	Automa…	00:07:46	Success	2…	30.0 GB	5	4.5 MB	5	4.5 MB
12/05/2010 16:08	Automa…	00:04:08	Success	2…	30.0 GB	3	135.6 KB	3	135.6 KB
12/05/2010 14:08	Automa…	00:04:10	Success	2…	30.0 GB	2	23.6 KB	2	23.6 KB
12/05/2010 11:54	Automa…	00:09:32	Success	2…	30.0 GB	16	266.7 KB	16	266.7 KB
12/05/2010 09:37	Automa…	00:02:28	Success	2…	30.0 GB	0	0 bytes	0	0 bytes
12/05/2010 07:37	Automa…	00:13:41	Success	2…	30.0 GB	27	43.3 MB	26	19.1 MB
10/05/2010 13:07	Automa…	00:04:00	Success	2…	30.0 GB	18	3.1 MB	15	2.6 MB
10/05/2010 08:07	Automa…	00:02:50	Success	2…	30.1 GB	0	0 bytes	0	0 bytes
10/05/2010 05:58	Automa…	00:02:46	Success	2…	30.8 GB	0	0 bytes	0	0 bytes
10/05/2010 02:11	Automa…	03:45:24	Success	2…	30.8 GB	3	701.4 MB	3	701.4 MB
09/05/2010 23:03	Automa…	03:07:34	Success	2…	30.6 GB	6	453.7 MB	6	453.7 MB
09/05/2010 21:36	Automa…	00:01:50	CancelError0	0	0 bytes	0	0 bytes	0	0 bytes
09/05/2010 18:54	Automa…	00:03:14	Success	2…	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 16:54	Automa…	00:03:33	Success	2…	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 14:54	Automa…	00:07:06	Success	2…	30.1 GB	0	0 bytes	0	0 bytes
09/05/2010 12:54	Automa…	00:05:14	Success	2…	30.1 GB	0	0 bytes	0	0 bytes

File	Path	Patch Size	Encoding T…	Transfer Ti…	Transfer R…	Other Details
30.Rock.S03E20.HDTV.XviD-LOL.avi	C:\Documents and Setting…	175.0 MB	00:00:22			File already on MozyHome servers

30.Rock.S03E20.HDTV.Xvid-LOL.avi 175MB

Deduplication

- Deduplication = storing and uploading only a single copy of redundant data
  - Applied at the file or block level
- Major savings in backup environments (>90% savings in common scenarios)
  - "most impactful storage technology"
    - July 2009: EMC acquires DataDomain for $2.1B
    - April 2008: IBM acquires Dilligent for $200M
    - July 2010: DELL acquires Ocarina for ???

Deduplication

- Cross-user deduplication
  - If two or more users store the same file, only a single copy is stored
- Source-based deduplication
  - Deduplication is performed at the client side
    - If the server has the file, no need to upload
  - Saves bandwidth as well as storage
  - Known also as “Client-side deduplication” or “WAN deduplication”

Deduplication and security

- Server state is a “joint resource” across different users
- Answer to “does-file-exist-on-server” leaks one bit of information about other users
  - [Harnik/Pinkas/Shulman-Peleg 2010] use this channel to leak “interesting” information
- Opens the door to stealing files
  - This work

Talk Outline

- A file-stealing attack
  - Attack description, some details
  - Discussion of real-life significance
- Our solution: proofs of ownership
  - Definition(s)
  - Relation to similar notions (PORs/PDPs)
  - Constructions

A File-Stealing Attack

Use of Hash Values

- Hash of file serves as identifier for content
- During upload
  - Client computes and sends hash of file
  - If hash value found (dedup), skip upload
  - Else (hash not fount) ask to upload the data
  - Either way, remember that client “owns” file
    - Client is then allowed to download the file back, e.g. when performing a Restore

The Attack

- Attacker obtains hash for victim's file
  - More on how to do it later
- Connects to server, tries to upload the file
  - Server asks for hash, attacker complies
  - Server skips upload, remembers that attacker owns the file
- Attacker asks to restore the file, downloads it from the server

The Attack

- Attacker obtains hash … victim's file
  - Mo… how …
- Conn… the file
  - Se… mplies
  - S… s that attacker owns th…
- Attacker asks … resto… the file, downloads it from the server

If you can get the hash of the file, you can get the file

Getting the hash value

- Hash is not meant to be secret
  - The dedup procedure may use a common hash function (e.g., SHA1, MD5)
- May be used for other purposes:
  - “Shouldn’t not reveal anything about the file”
  - Fingerprint software/media, timestamp contributions, …
    - E.g., I publish a fingerprint of my software, one user backs it up, now everyone can get it from server

Getting the hash value (2)

- Malicious software
  - A malicious software on Bob's machine wants to stealthily leak all his files to Alice
  - Instead of sending huge files, can send the short hash values of the files
    - Much harder to detect and prevent
- Also true for server break-in
  - Dump all hashes in memory and run…
  - Even if detected, only remedy is to turn off dedup for affected files (essentially forever)

Getting the hash value (3)

- Content distribution network (CDN)
  - Alice wants to share a huge file with her friends
  - Uploads file to server, sends hash to friends
  - Friends use backup service to download file
- Server used as a CDN, unknowingly
  - Might break its cost structure
    - If it planned on serving only a few restore ops
  - Might break the law
    - If huge file was copyrighted

Is This a Real Problem?

- How hard it is to implement the attack?
  - Leo Dorrendorf & Benny Pinkas Implemented the attacks against two major storage servers
  - Not quite straightforward, not very hard either
  - In some cases the standard client software keeps a control-file with all hash values
    - Makes the attack a lot easier

Is This a Real Problem? (2)

- Emerging open protocols for cloud storage
  - E.g. CDMI from SNIA (storage standards body)
- Support for client-side dedup is coming
- Standardization makes the CDN attack trivial, simplifies also other attacks
- Practical solutions to these attacks are needed as an enabler for this technology

Is This a Real Problem? (3)

"Overall, I liked the paper but felt that it is a solution searching for a problem"

Anonymous reviewer, USENIX Security 2011

Is This a Real Problem? (4)

- Dropship, a new open-source project by Wladimir van der Laan (April 2011)
  - "written in Python. Allow you to download to your Dropbox any file, which description we got in JSON format (similar as description propagated in .torrent files)."
  - "Have you ever dreamt about the ability to download new movies in a super fast, safe way from distributed network? Are you interested … in downloading with maximum bandwidth wherever you are, 24/7, with super safe connection and being extremely anonymous"
- Implemented the CDN attack over Dropbox

Is This a Real Problem? (4)

- Dropbox's CTO contacted the creator of Dropship, requested "in a really civil way" that he takes the project off of github
  - Project reveals Dropbox'es protocol
  - Can support piracy
- van der Laan complied
- Follow-up discussion on slashdot (mostly about "censorship")

Is This a Real Problem? (5)

- Concurrent work:
  “Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space”
  - Mulazzani, Schrittwieser, Leithner, Huber, and Weippl (*SBA Research*)
- Implemented the same attacks against Dropbox

To appear in USENIX Security 2011

Our Solution:
Proofs-of-Ownership

A Naïve Solution

- Use application-specific hash, salt
  - e.g. SHA("service name" | salt | file)
  - Other applications won't use the same hash
  - Solves fingerprinting/timestamping scenarios
- But hash is still not secret
  - All clients must know hash function
- Does not address root cause of problem
  - Large file is still represented by a short string, if you can get the short string then you get the file
- Many attack scenarios remain (CDN, break-in, etc.)

A Better Naïve Solution

- Use a challenge-response mechanism
- E.g., for every upload server picks a random nonce, asks client to compute SHA(nonce | file)
  - This "proves" that client knows the file ☺
  - But server must retrieve the whole file from secondary storage to check the answer ☹
- We want a better proof mechanism

Proofs of Ownership (PoWs)

- Protocol for client (prover) and server (verifier)
  - Client has the file
  - Server stores only short verification information
    - Verification information computed from the file
  - The proof itself is bandwidth-efficient
    - Much shorter than sending the whole file
- Adversary may have partial info about the file
  - E.g., its hash value, maybe more
- Want proof to succeed only if client has the whole file

Security Definition

- In the spirit of the bounded-retrieval model
  - Also reminiscent of [GJM'02]
- Roughly follows the CDN attack scenario
- The requirement (informally):

As long as the file has sufficient entropy left (from the adversary's perspective), the proof will fail whp

Security Definition

1. File chosen from adversarial distribution
2. Verifier computes verification information
3. Adversary has accomplices that get file, interact with verifier, leak to adversary
   - But leakage is limited
4. Then adversary interacts with verifier
   - No communication with accomplices now (we do not protect against man-in-middle)

Security Definition

- Strict definition:
  As long as leakage is less bits than
  initial-min-entrpoy – security-parameter
  adversary only has negligible probability of convincing the verifier
- Later we relax this requirement

Practical Considerations

- Low bandwidth
- Very short verification information
  - Only a few bytes per file
- Efficient processing by client, server
  - File itself may be very large, perhaps does not even fit in main memory
  - Would be nice to have a steaming solution, (e.g., similar to just computing SHA(file))

Relation to Proofs of Retrievability

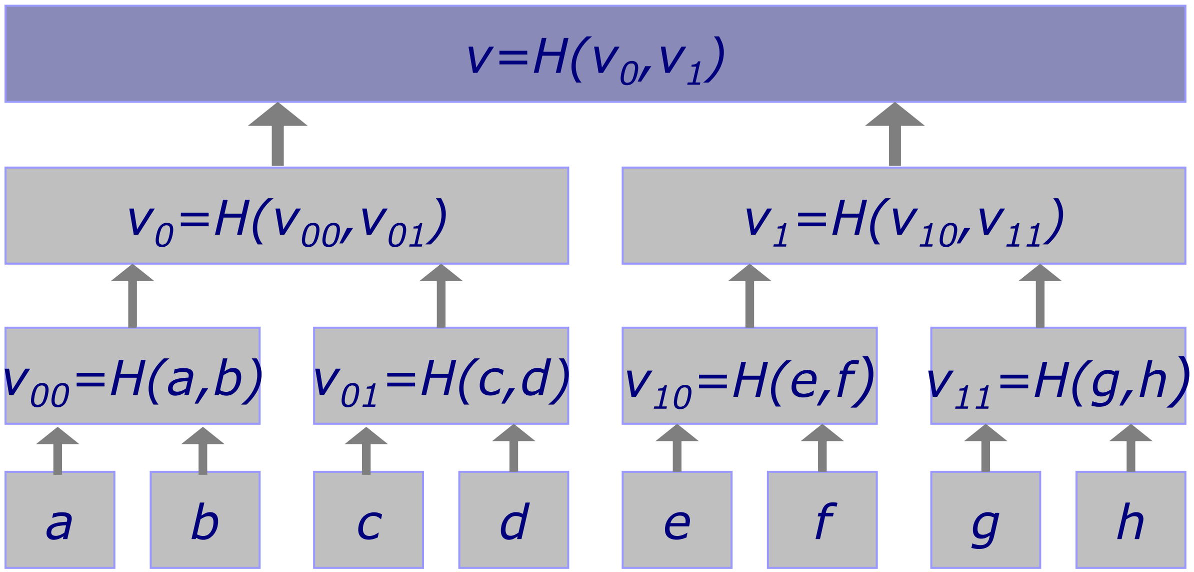
- In PORs [JK07,ABC$^+$07], server proves to client that it actually stores its file
  - Role-reversal from PoW's
- In most PORs, client (verifier) has secret state, file is pre-processed using this state before uploading to server
  - E.g., client embeds many authentication tags, then verifies a random subset of them
    - [NR05],[JK07],[SW08],[DVW09]
  - Cannot be done in our setting

Relation to Proofs of Retrievability

- Security definition of POR is strictly stronger than our definition of PoW
  - Requires an extractor a-la-POK
- Any POR without preprocessing is a PoW
  - But not every PoW is a POR
  - One of our constructions is a POR, others are not

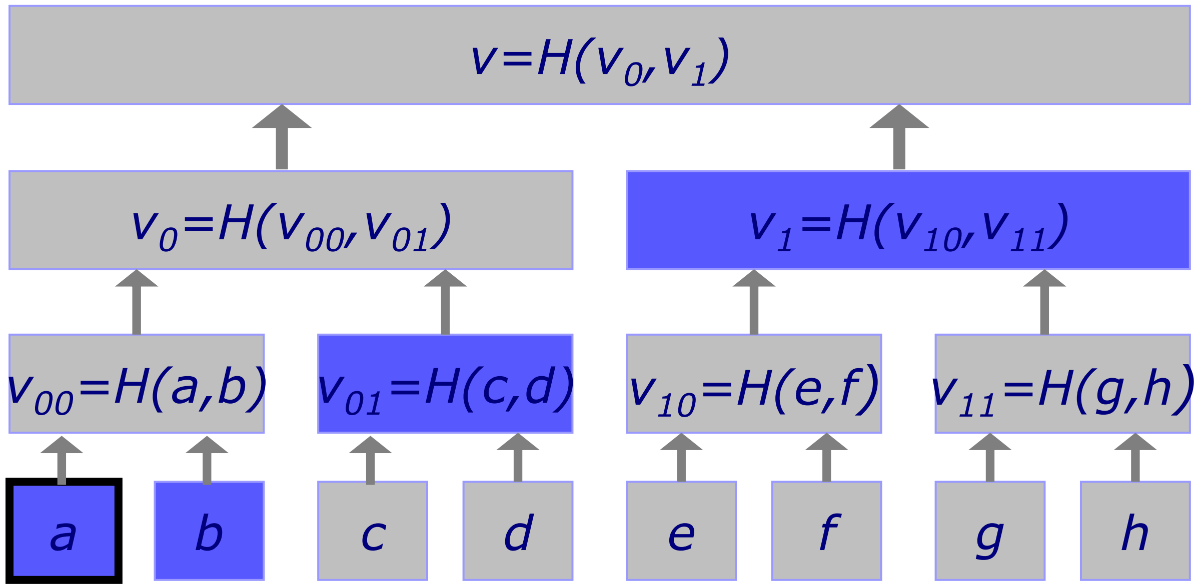
Background: Merkle Hash Trees

- Committing to n values, $x_1,\ldots,x_n$, such that
  - The commitment is short (a single hash value)
  - Can "open any x_i" with a de-commitment message of length O(log n)

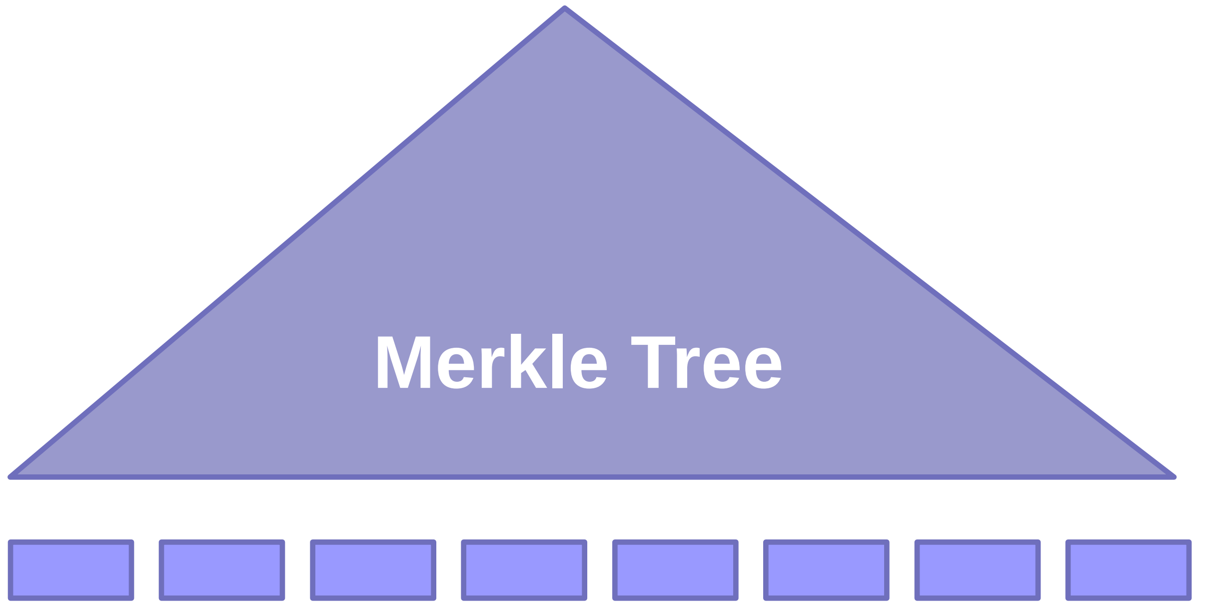


Background: Merkle Hash Trees

- The commitment is the root value v
- To open a leaf, send the sibling path from that leaf to the root
  - E.g., opening leaf *a* by providing *b*, v_{01}, and v_1



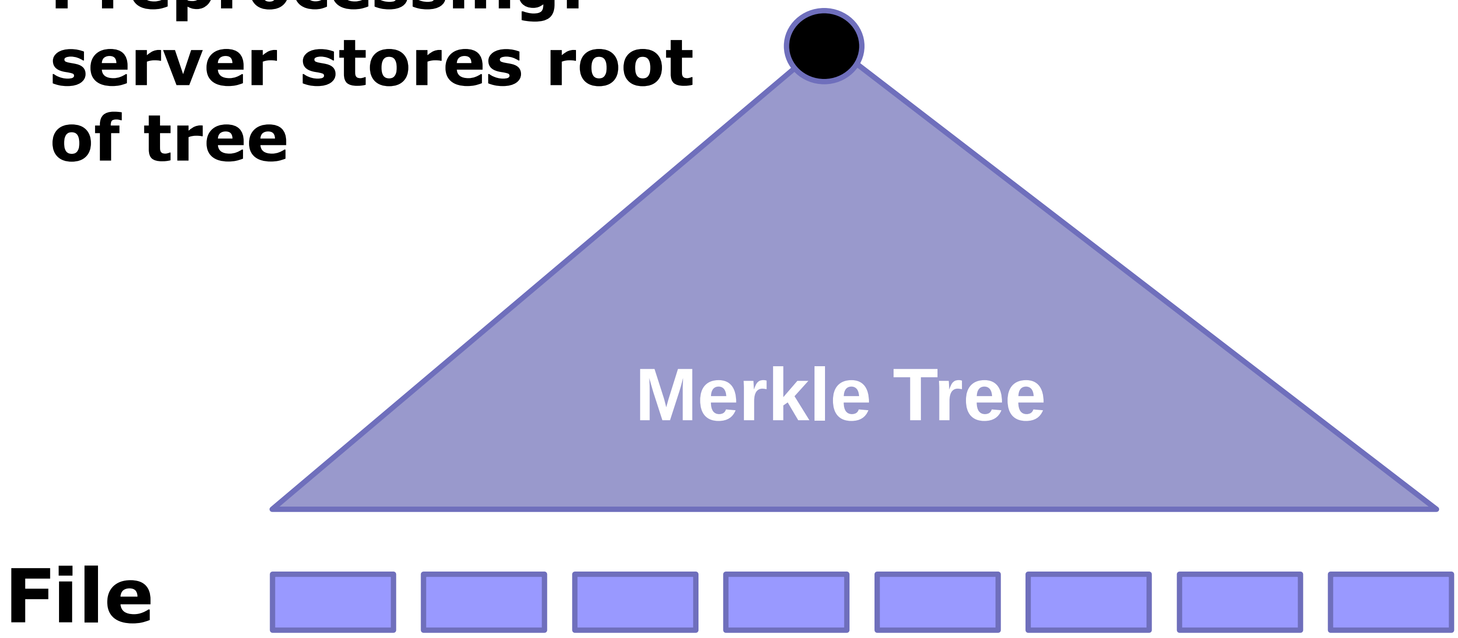
Solution – first attempt



File

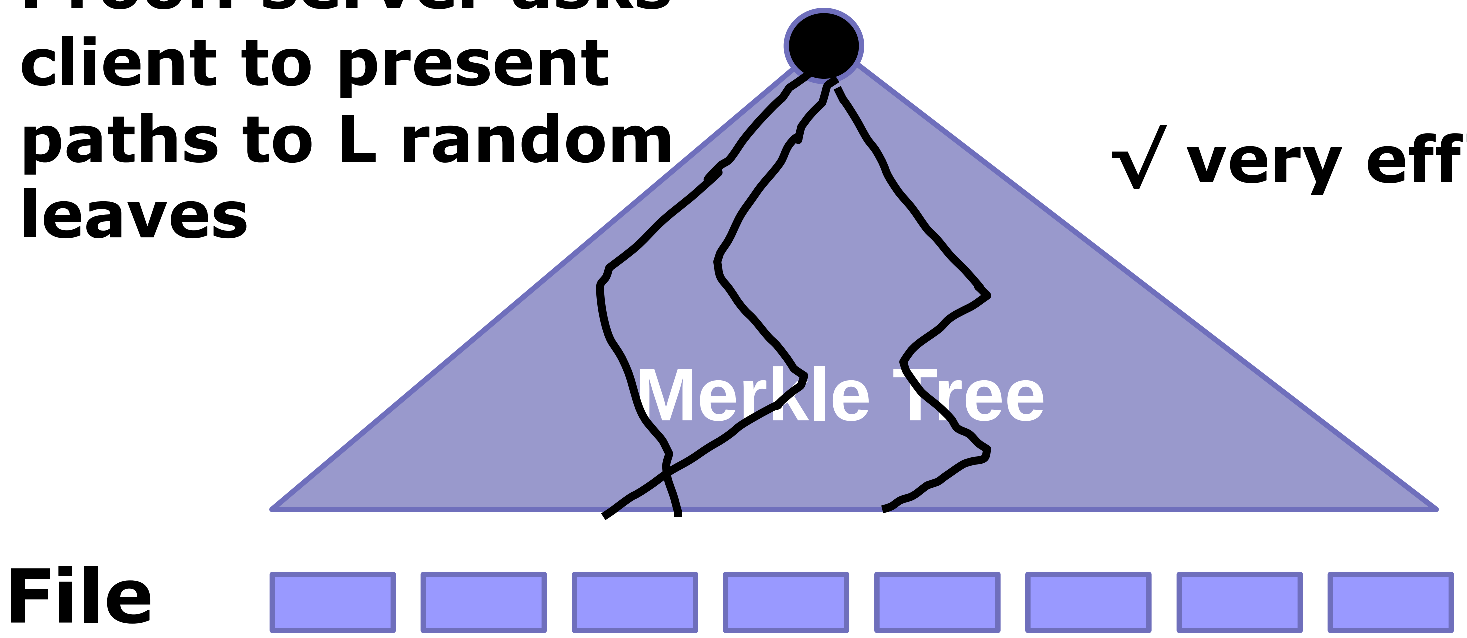
Solution – first attempt

Preprocessing: server stores root of tree



Solution – first attempt

Proof: server asks client to present paths to L random leaves

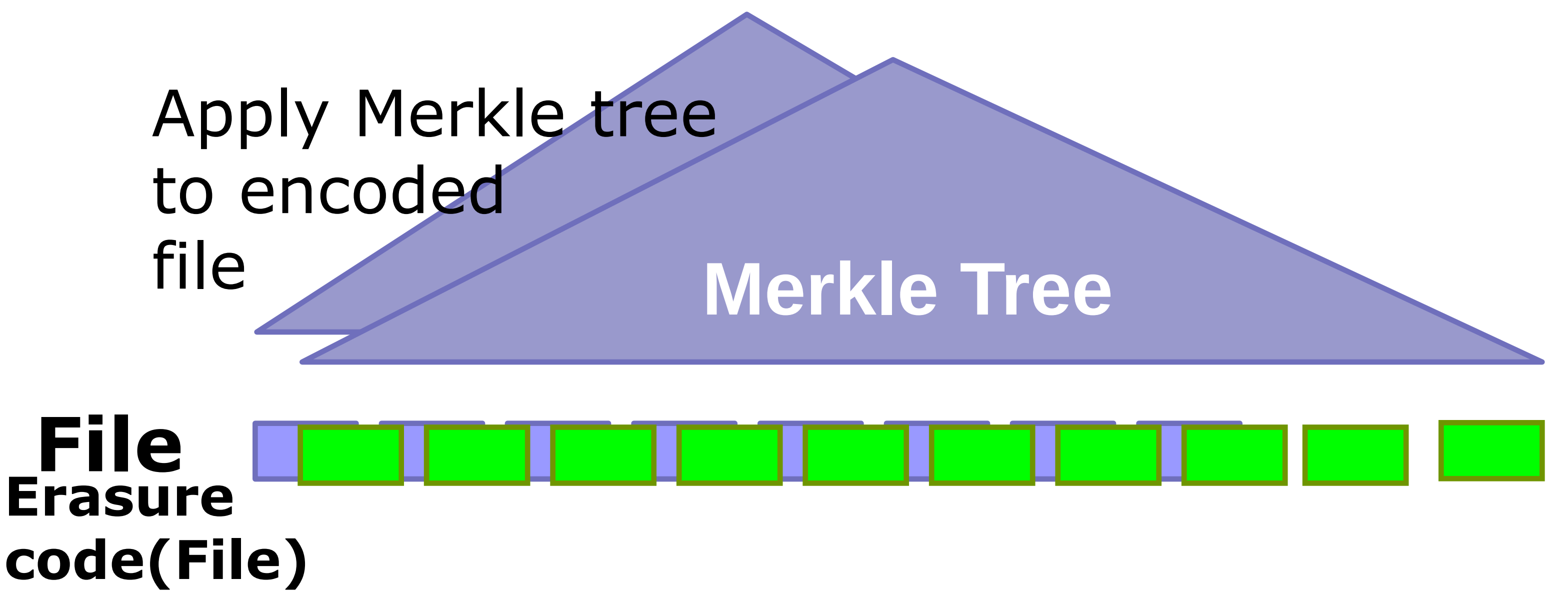


√ very efficient

A client which knows only a *p* fraction of the file, succeeds with prob < p^L.

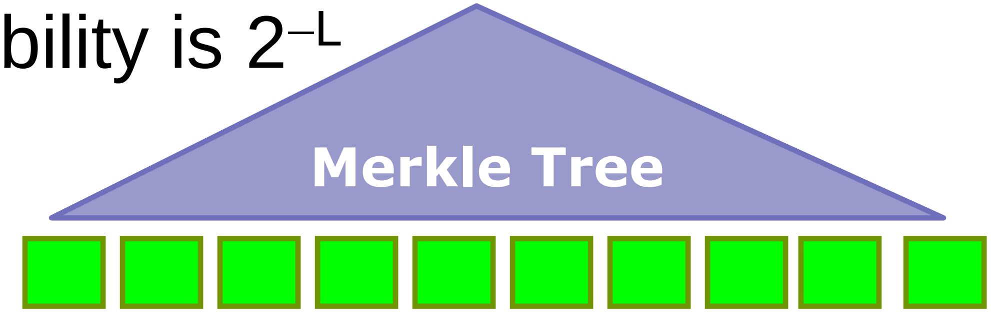
Problem and solution

- Adversary that knows a large fraction of the blocks (say, 95%), can pass the test with reasonable probability (0.95^{10}=0.6).
- Solution:



Construction 1: Erasure code & Merkle tree

- Erasure code property: knowledge of, say, 50% of the encoding suffices to recover original file
  - attacker who misses even a single block of the file, does not know > 50% of the encoding
  - Fails in each Merkle tree query w.p. 50%.
  - Cheating probability is 2^{-L}



Proof of Security

- Merkle-tree lemma: Given a prover that succeeds with probability ε^L, can extracts ~ ε-fraction of the base of the tree whp
  - A simple “hardness amplification” result
- Proof uses extractor to extract the file from the adversary (whp)
  - Must be “the right file”, or else a hash collision
  - Contradicts the fact that file has high min-entropy from adversary’s perspective
- This is actually a POR, not just a PoW

Efficiency?

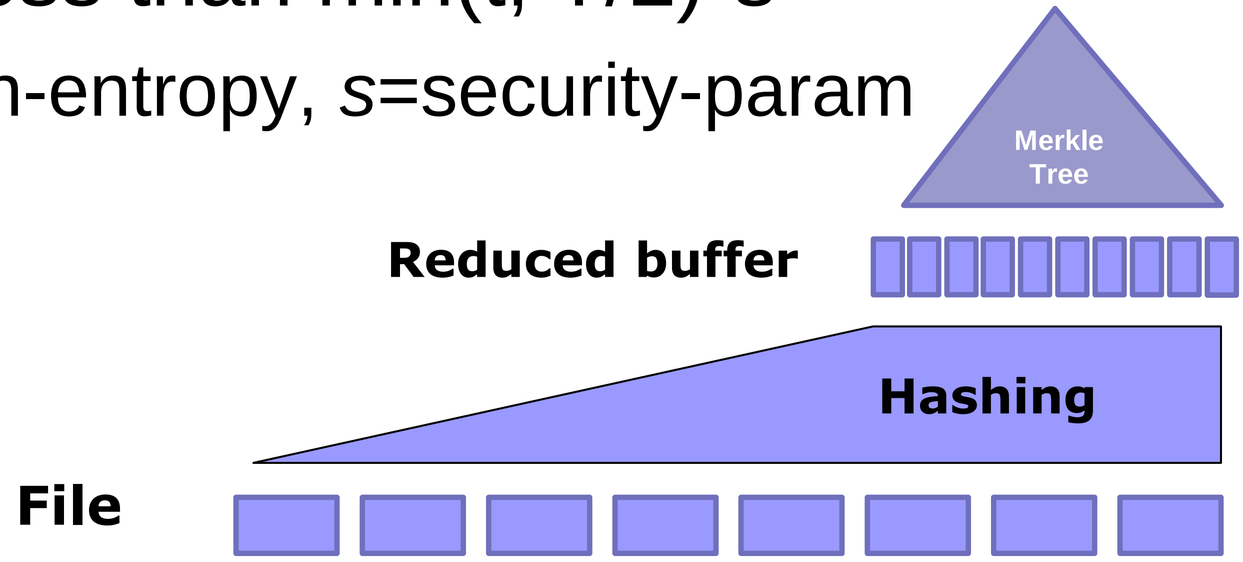
- Computing an erasure code for a large file
  - No streaming solution (that we know of)
  - Need random-access to either input or output of the encoding procedure
- Very expensive if file doesn’t fit in memory
  - Many many disk-seeks

Small Space Protocols

- Seems hard for the strict security definition
  - Small space at client is “small representation” of file, leaking it lets one complete the proof
  - (Of course, this is not an impossibility proof)
- Relax the requirement
  - Introduce a threshold τ, adversary may succeed if it gets τ leakage bits of the file
    - Set τ large, not huge (e.g., τ = 1GB)
  - Protocol works in space $O(\tau)$

Construction 2: Hash & Merkle Tree

- Universal hashing to reduce file to an T-byte buffer, Merkle-tree over the buffer
- Security: Adversary fails whp if leakage amount is less than min(t, T/2)-s
  - *t*=initial-min-entropy, *s*=security-param



Proof of Security

- Similar to before
- Main lemma: no leakage function ***leak(F)*** lets the adversary learn a large fraction of the hashed buffer ***h(F)***
  - Assuming that ***leak*** has short output
  - Even if ***leak*** can depend on ***h***
  - With high probability over the choice of ***h***
- Use pairwise-independence + union-bound

Proof of Security, Main Lemma

- $\mathcal{D}$ is distribution on $\{0,1\}^M$, min-entropy ≥k
- h is pairwise independent $h : \{0,1\}^M \rightarrow \{0,1\}^{bT}$
  - b is size of Merkle-tree leaves (b≥2 bits)
  - We assume that k < T/3
- Then whp over the choice of h, for *every* large subset of blocks S $\subseteq$ {1,2,…,T}, |S|>$\frac{2T}{3}$ the projection h($\mathcal{D}$)$_S$ has min-entropy ≥ k-1

Proof: roughly, no collisions so min-entropy is not reduced (and then union bound)

Efficient Enough?

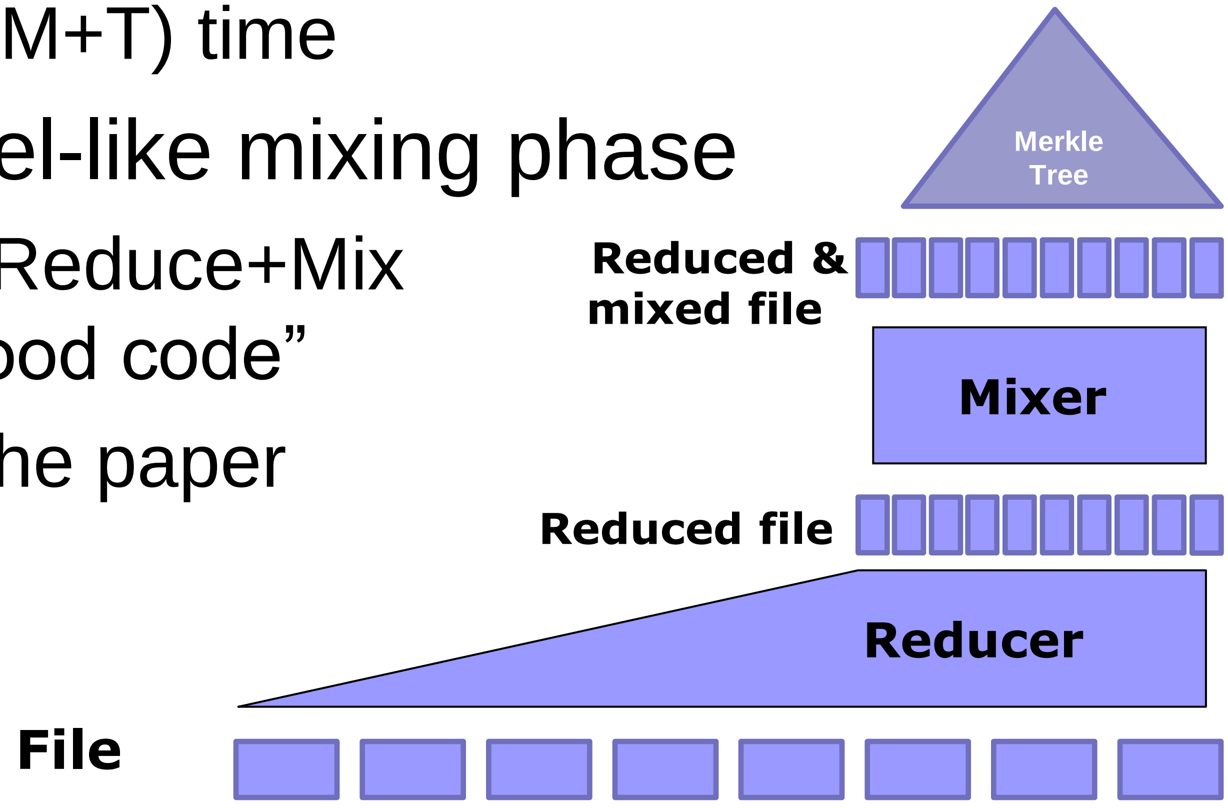
- Hashing output fits in memory, can compute it in “streaming fashion” ☺
- Still not as efficient as we would want
  - File size M, buffer size T, hashing takes $\Omega(M \cdot T)$ time ☹
- Can we do better?

Construction 3: Reduce, Mix & Merkle

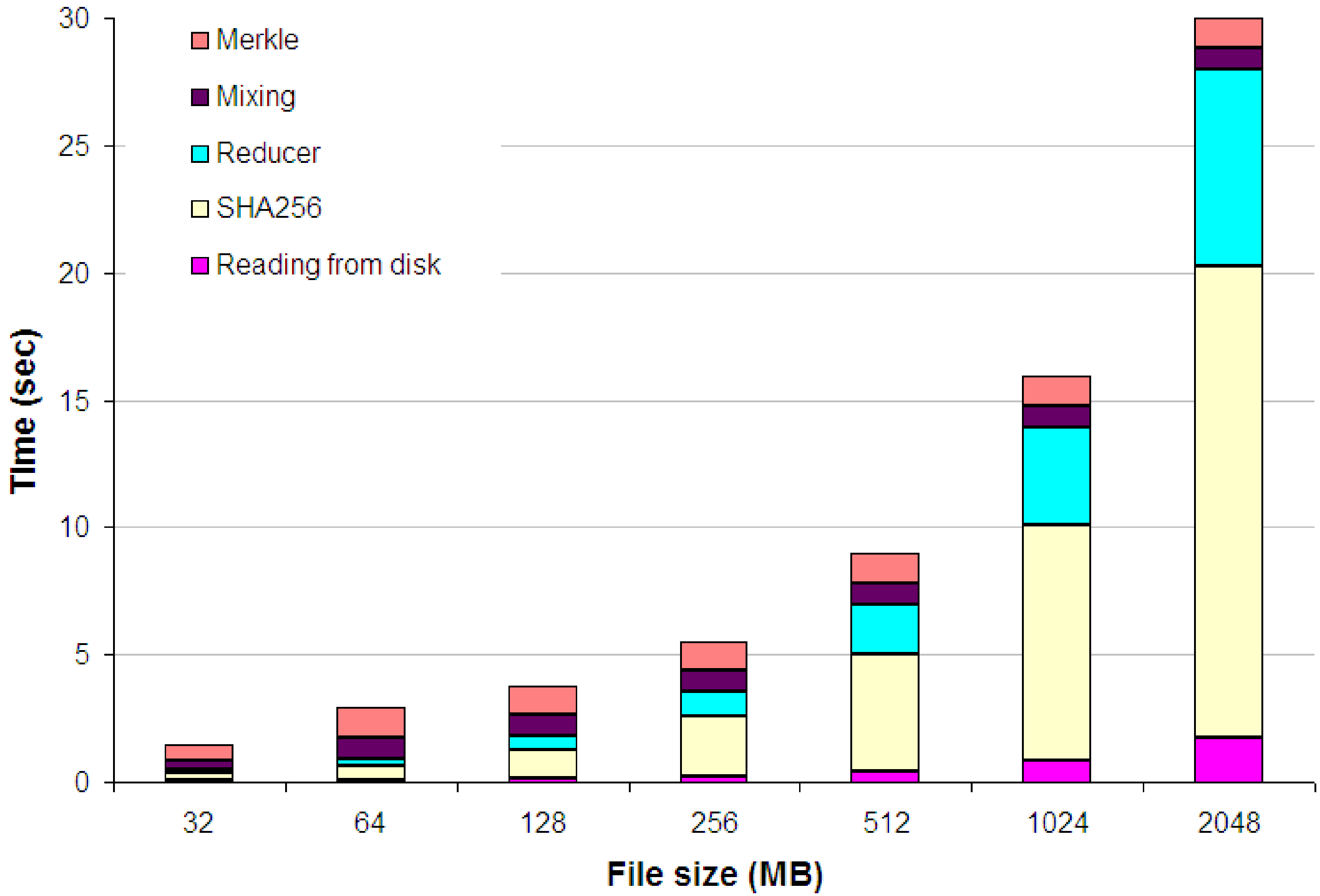
- Want to use a simpler length-reduction than universal hashing
  - Goal: If adversary is missing even a small part of the file (after leakage), it will miss a large fraction of the reduced-length buffer
- We design an efficient ad-hoc procedure, “hope that it works”
  - We prove security against a certain class of input distributions, under a coding assumption

Construction 3: Reduce, Mix & Merkle

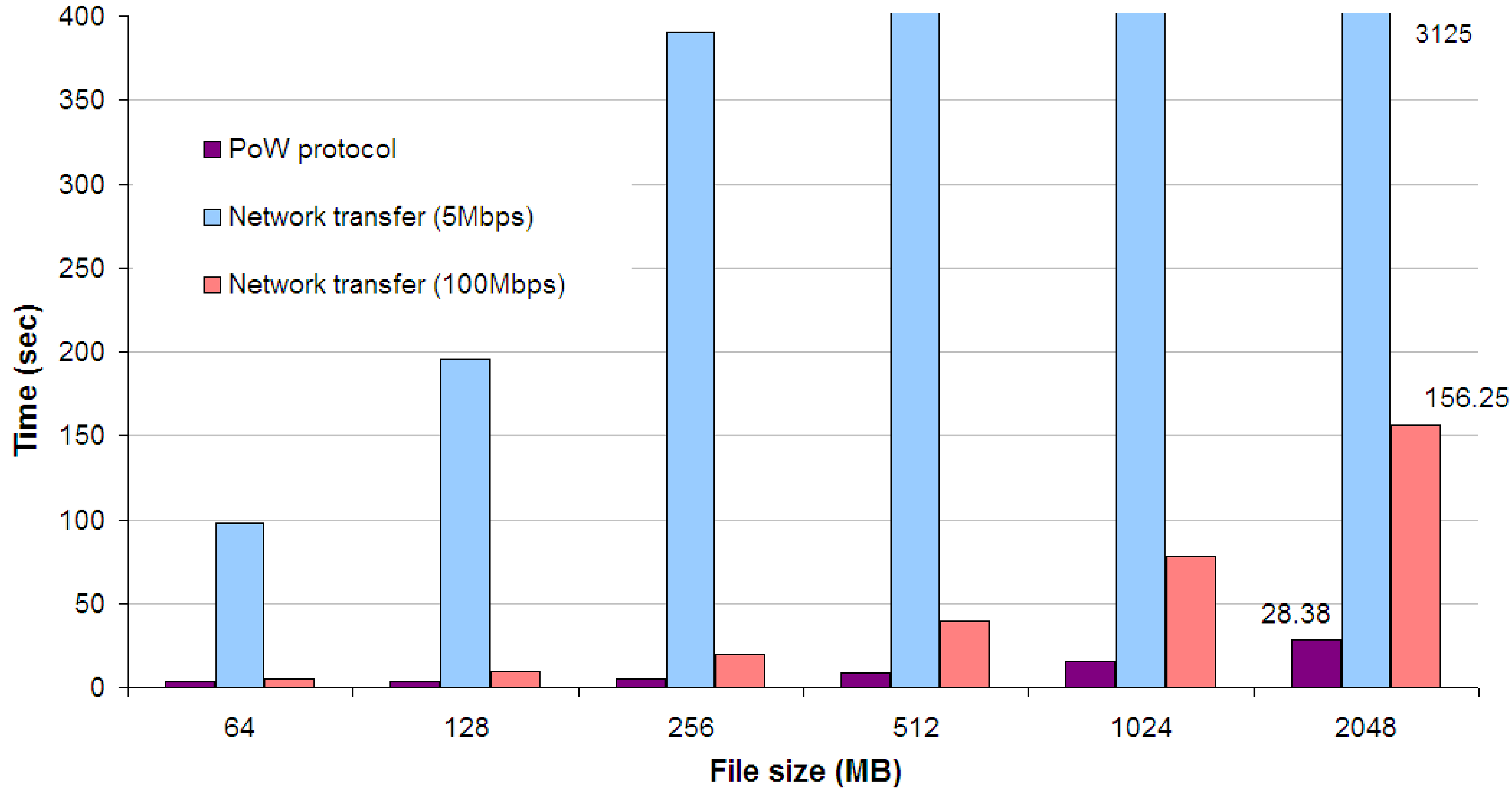
- Reducer: XOR each block to a constant number of random locations
  - Runs in O(M+T) time
- Add a Feistel-like mixing phase
  - Hope that Reduce+Mix make a “good code”
  - Details in the paper



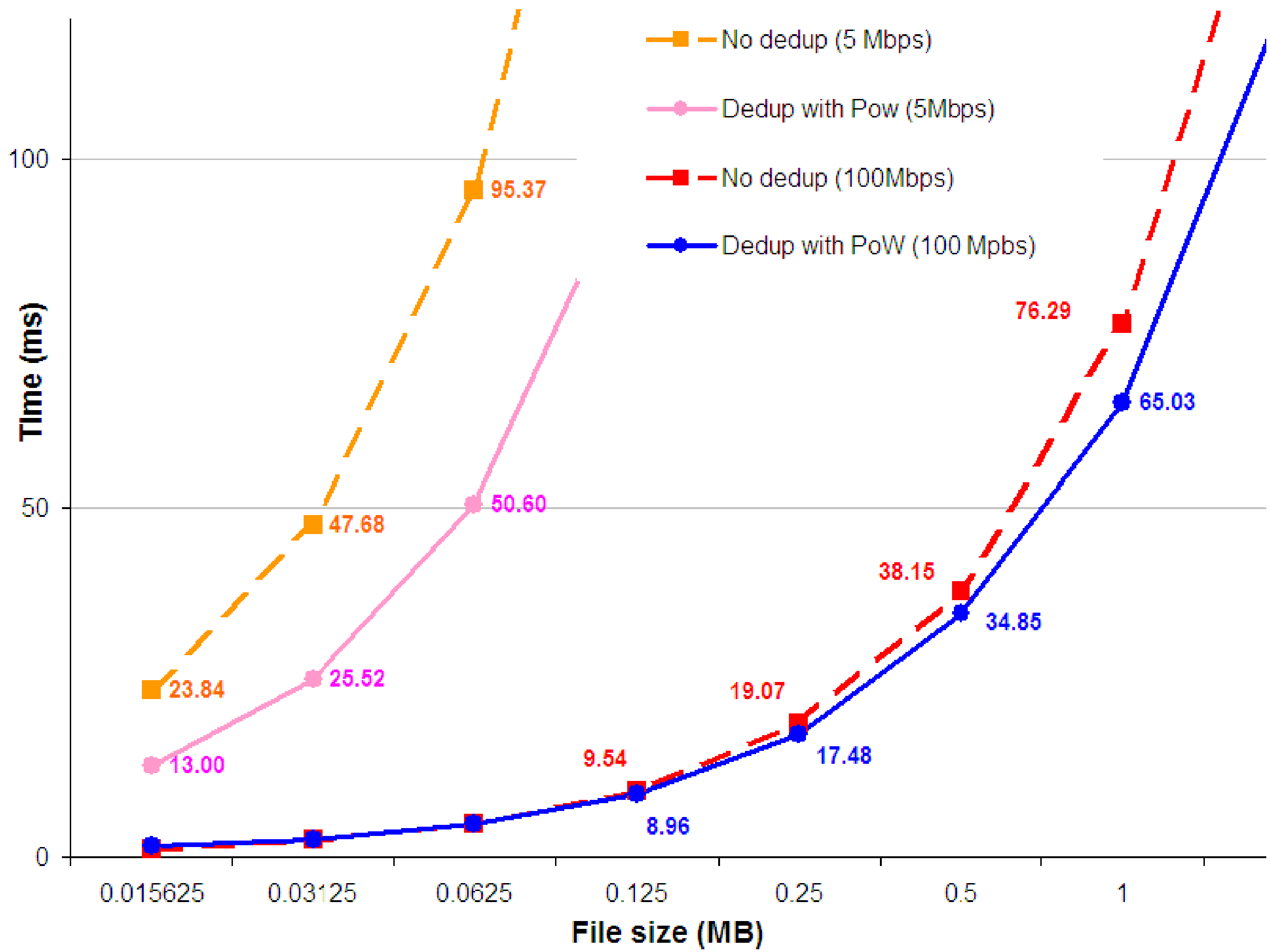
Performance of streaming PoW



Running PoW vs. Sending the File



When is it Worth the Effort?



Conclusions

- Deduplication offers huge savings and yet might leak information about other users
- Most vendors just now becoming aware of this
- The challenge: offer meaningful privacy guarantees with a limited effect on cost
- Major challenge in making it practical….