Athos: Efficient Authentication of Outsourced File Systems

Michael Goodrich - University of California Irvine, USA

Charalampos Papamanthou - Brown University, USA

Roberto Tamassia - Brown University, USA

Nikos Triandopoulos – University of Aarhus, Denmark

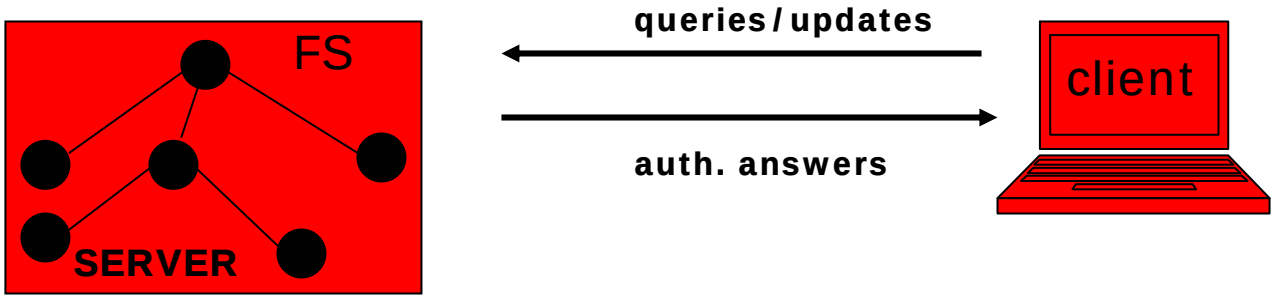
September 16th, 2008, Taipei, Taiwan

Information Security Conference 2008 (ISC 2008)

Research supported in part by the U.S. National Science Foundation under grants IIS–0713403, IIS-0713046, CNS-0312760 and OCI–0724806, the I3P Institute under a U.S.DHS award, the Center for Algorithmic Game Theory at the University of Aarhus under an award from the Carlsberg Foundation, the Center for Geometric Computing and the Kanellakis Fellowship at Brown University, and IAM Technology.

Scope and Problem

- Authenticated File Systems (FS)
- A client C outsources a FS at a server S due to
  - Space limitations/back-up or archiving purposes



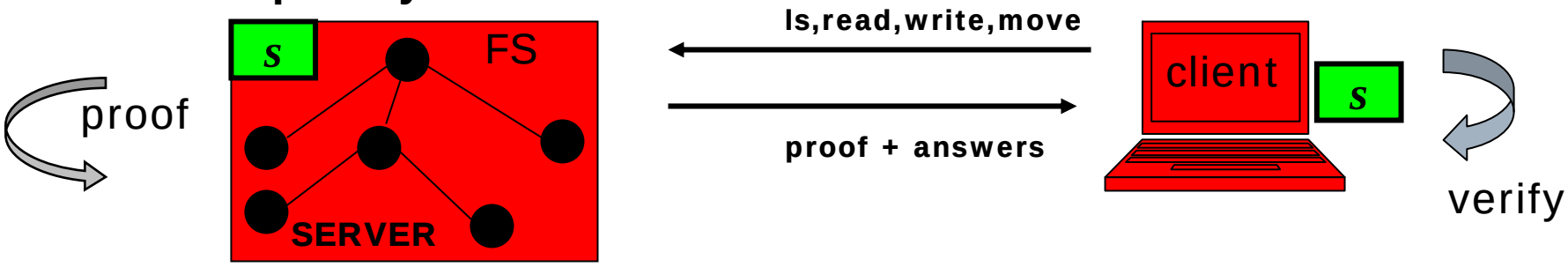
- S is untrusted
- C wants to maintain the integrity of FS, e.g.,
  - When C executes **ls**, nothing is missing
  - Authentication of both **contents** and **hierarchy**
- S should return proofs of correctness for all the above
- This should be done efficiently (e.g., logarithmic cost)

Summary of related work

- Store a hash for each file (e.g., *Miller et al., Cataneo et al.*)
  - Needs $O(n)$ space at the client
  - Authenticates only the content and not the hierarchy
- Build a Merkle Tree on top of FS (e.g., *Yumerefendi and Chase*)
  - Better: Need only $O(1)$ space
  - Needs to rebalance the tree for updates
- Use some trusted hardware (TPM) (e.g., *Oprea and Reiter*)
  - Assumption for some trusted component at the server
- Hashing on the FS tree (e.g., *Goh et al.*)
  - Can authenticate hierarchy
  - Very expensive updates due to unbalanced tree
- SUNDR (*Mazieres et al.*)
  - Solution given is under a different model though
  - Multiple clients can access/update the file system
  - Best you can achieve: Fork consistency
  - S can divide users into different groups that do not see the same state

Model and Contributions

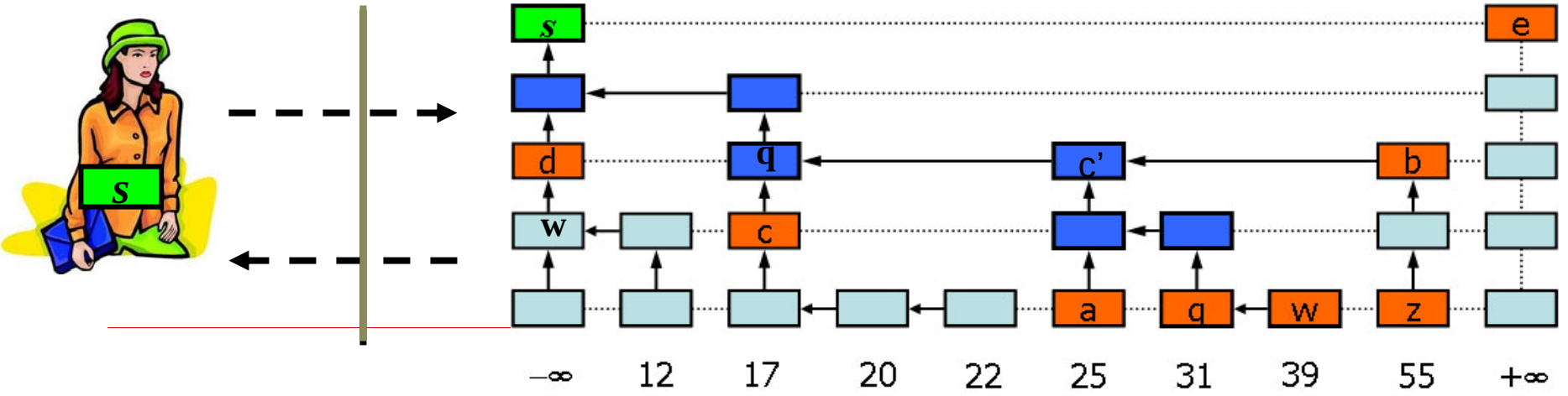
- Two-party model (client - server)
- **Completely** untrusted server



- Client with $O(1)$ trusted storage, ***s*** (state of the file system)
- ***s*** is a function of the FS hierarchy/contents– updated during updates
- Queries, Updates: e.g., **ls**, **read**, **pwd, write**, **move**, **chmod**
- S computes a proof
- C is able to verify by using ***s*** and the proofs returned by S
- Security is based on collision resistant of cryptographic hashing
- Logarithmic costs for verification, auth. updates, communication
- Main building blocks:
  - Two-Party Authenticated Skip List (Papamanthou, Tamassia, ICICS07)
  - Authenticated Dynamic Trees (Goodrich, Tamassia, Triandopoulos, CT-RSA 2003)

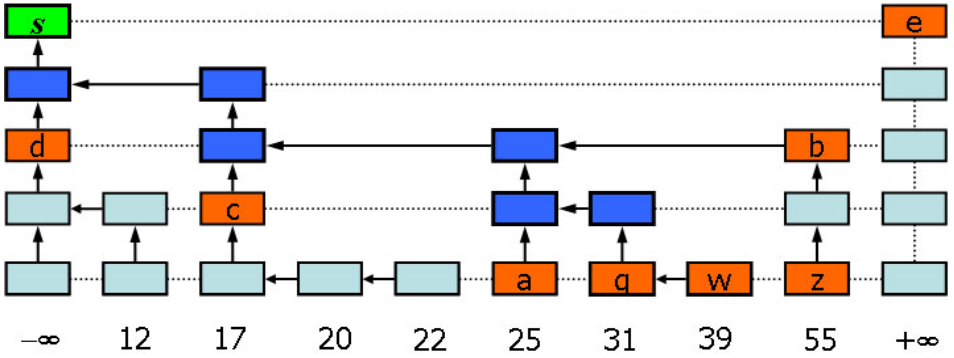
Authenticated Skip List – Flat Directory

- Skip list augmented with cryptographic information
  - Stored by S
  - Used for proofs of correctness for queries/updates
- At node v: $f(v)=h(.,.)$, where h is collision-resistant
  - $f(v)=h([f(right)\ \ell(v)\ \text{key}(v)],[f(down)\ \ell(v)\ \text{key}(v)])$
  - e.g., q = h([c' 2 17],[c 2 17])
- State **s**: label of top left node stored by the client



Query/Verify for file 39

get (39)?



=?

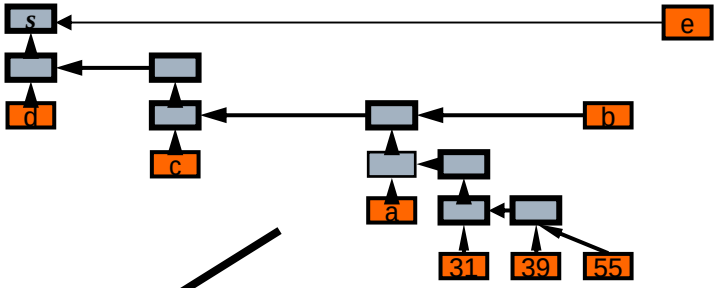
h(,,,)
h(,,)
h(,,)
h(,,)
h(,,)
h(,,)
h(,,)

=

H(.)

f	ℓ	k
55	0	55
39	0	39
31	0	31
a	1	25
b	2	25
c	2	17
d	3	-∞
e	4	-∞

f	ℓ	k
55	0	55
39	0	39
31	0	31
a	1	25
b	2	25
c	2	17
d	3	-∞
e	4	-∞



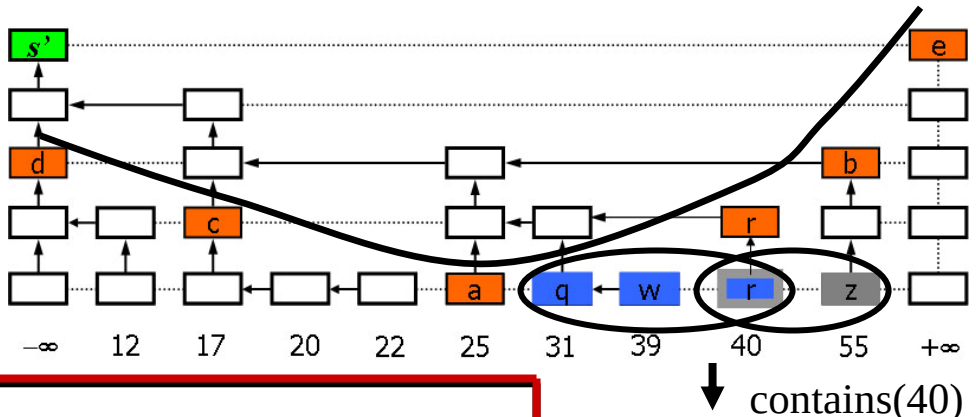
Insert file 40

insert(40)

s' = $H([L\ H(R)\ U])$

s

=?



contains(40)

L= [40 39 31]

R= [55 40]

U= [a b c d e]

f	ℓ	k
55	0	55
40	0	40
39	0	39
31	0	31
a	1	25
b	2	25
c	2	17
d	3	-∞
e	4	-∞

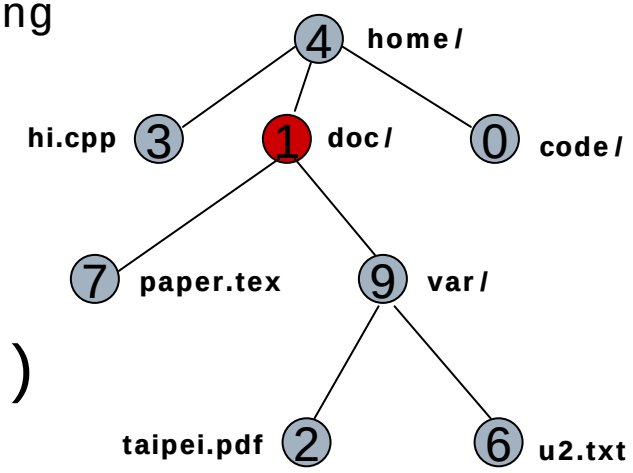
h(,,)
h(,,)
h(,,)
h(,,)
h(,,)
h(,,)
h(,,)

f	ℓ	k
55	0	55
39	0	39
31	0	31
a	1	25
b	2	25
c	2	17
d	3	-∞
e	4	-∞

f	ℓ	k
55	0	55
39	0	39
31	0	31
a	1	25
b	2	25
c	2	17
d	3	-∞
e	4	-∞

From skip lists to file systems (local information)

- For each node v of the file system tree
  - Create one entry in the skip list
  - Key is the i-node of the node (unique)
- With each node *v* store a tuple *l*(*v*) with the following fields
  - **name**, the name of the file (or directory)
  - **hash,** hash of the file contents, if file, else **null**
  - **key(parent)**, i-node of the parent
  - **key(sibling)**, i-node of the sibling
  - **key(backsibling)**, i-node of the backsibling
  - **key(child)**, i-node of the first child



(1,[“doc”,null,4,0,3,7])

(3,[“hi.cpp”,h(hi.cpp),4,1,null,null])

Does path "**home/doc/var/u2.txt**" exist?

s

GET(6)

(6,["u2.txt",h(u2.txt),9,null,2,null]) + **proof**

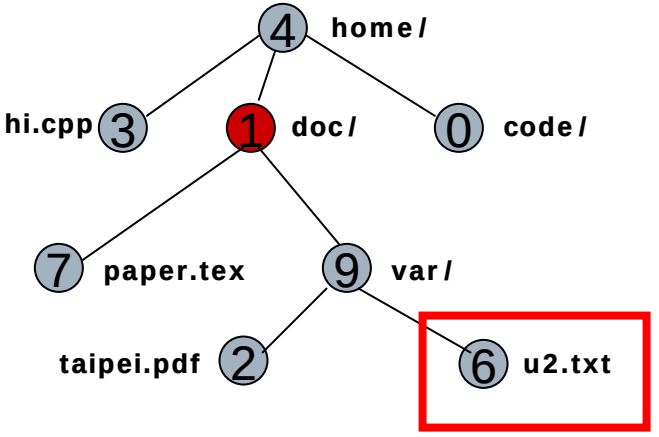
s == hash(**proof**)?

name=="u2.txt"?

Then: parent = 9

(0,["code",null,null,null,1,null])
(1,["doc",null,4,0,3,7])
(2,["taipei.pdf",h(taipei.pdf),9,6,null,null])
(3,["hi.cpp",h(hi.cpp),4,1,null,null])
(4,["home",null,null,null,null,3])
(6,["u2.txt",h(u2.txt),9,null,2,null])
(7,["paper.tex",h(paper.tex),1,9,null,null])
(9,["var",null,1,null,null,2])

s



Does path "**home/doc/var/u2.txt**" exist?

s

GET(9)

(9,["var",null,1,null,null,2]) + **proof**

s == hash(**proof**)?

name=="var"?

Then: parent = 1

(0,["code",null,null,null,1,null])

(1,["doc",null,4,0,3,7])

(2,["taipei.pdf",h(taipei.pdf),9,6,null,null])

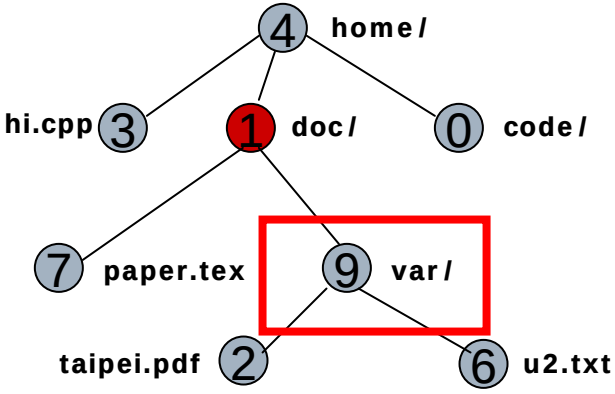
(3,["hi.cpp",h(hi.cpp),4,1,null,null])

(4,["home",null,null,null,null,3])

(6,["u2.txt",h(u2.txt),9,null,2,null])

(7,["paper.tex",h(paper.tex),1,9,null,null])

s (9,["var",null,1,null,null,2])



Does path "**home/doc/var/u2.txt**" exist?

s

GET(1)

(1,["doc",null,4,0,3,7]) + **proof**

(0,["code",null,null,null,1,null])

(1,["doc",null,4,0,3,7])

(2,["taipei.pdf",h(taipei.pdf),9,6,null,null])

(3,["hi.cpp",h(hi.cpp),4,1,null,null])

(4,["home",null,null,null,null,3])

(6,["u2.txt",h(u2.txt),9,null,2,null])

(7,["paper.tex",h(paper.tex),1,9,null,null])

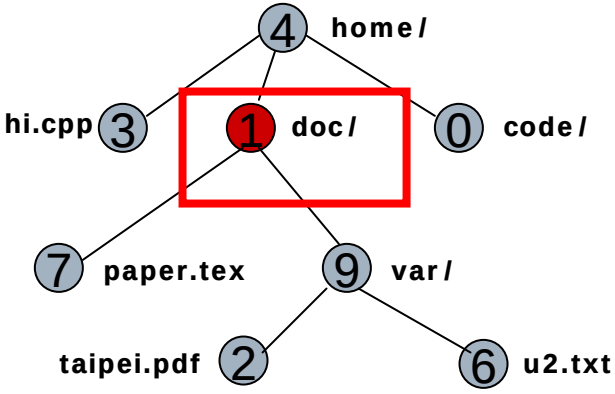
(9,["var",null,1,null,null,2])

s

s == hash(**proof**)?

name=="doc"?

Then: parent = 4



Does path "**home/doc/var/u2.txt**" exist?

s

GET(4)

(4,["home",null,null,null,null,3]) + **proof**

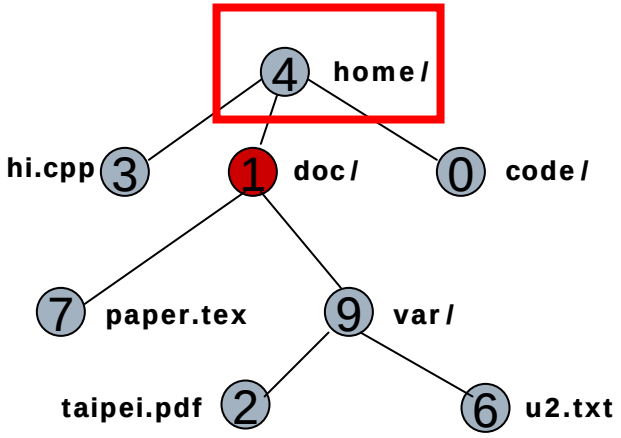
s == hash(**proof**)?

name=="home"?

Then: parent = null
and we are done!

(0,["code",null,null,null,1,null])
(1,["doc",null,4,0,3,7])
(2,["taipei.pdf",h(taipei.pdf),9,6,null,null])
(3,["hi.cpp",h(hi.cpp),4,1,null,null])
(4,["home",null,null,null,null,3])
(6,["u2.txt",h(u2.txt),9,null,2,null])
(7,["paper.tex",h(paper.tex),1,9,null,null])
(9,["var",null,1,null,null,2])

s



Create/read an authenticated FS

- Create: For each node v of the FS tree
  - Compute (key(v), I(v))
  - Use skip list insertion algorithm for (key(v),I(v))
- Let r be the root node of the FS tree
- Call READ(r)
- READ(v):
  - Send query **get**(key(v))
  - Receive proof for binding (key(v), I(v))
  - Authenticate I(v)
  - If **verifies**

    READ(key(child(v)));

    READ(key(sibling(v)));

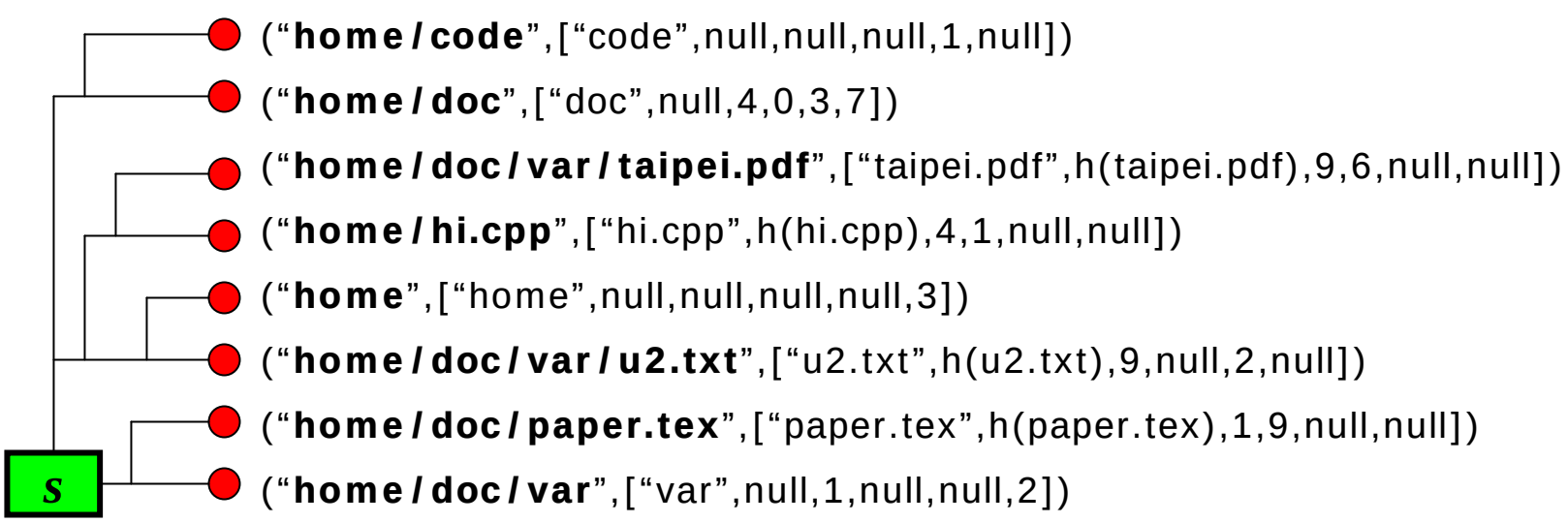
    else

    "INTEGRITY PROBLEM"

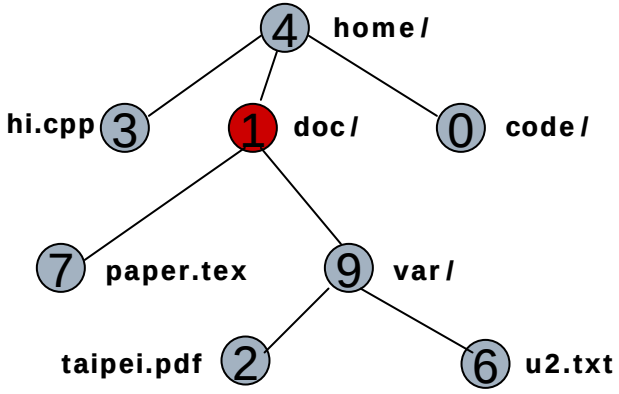
Time complexity - skip list

- P is a path of size k
- n is the size of the file system
- Authentication of P: $O(k \log n)$
- **cd**(P), **read**(P), **write**(P), **rm**(P), **mkdir**(P), **touch**(P): $O(k \log n)$
- **ls**(P): $O((k+l)\log n)$ (l = |returned list|)
- **rmdir**(P): $O((t+k)\log n)$
  - t = |removed directory|
- **move**(P,P'): $O((k+k')\log n)$

From skip lists to file systems (global information)



- path authentication: $O(k+\log n)$
- inefficient moves!



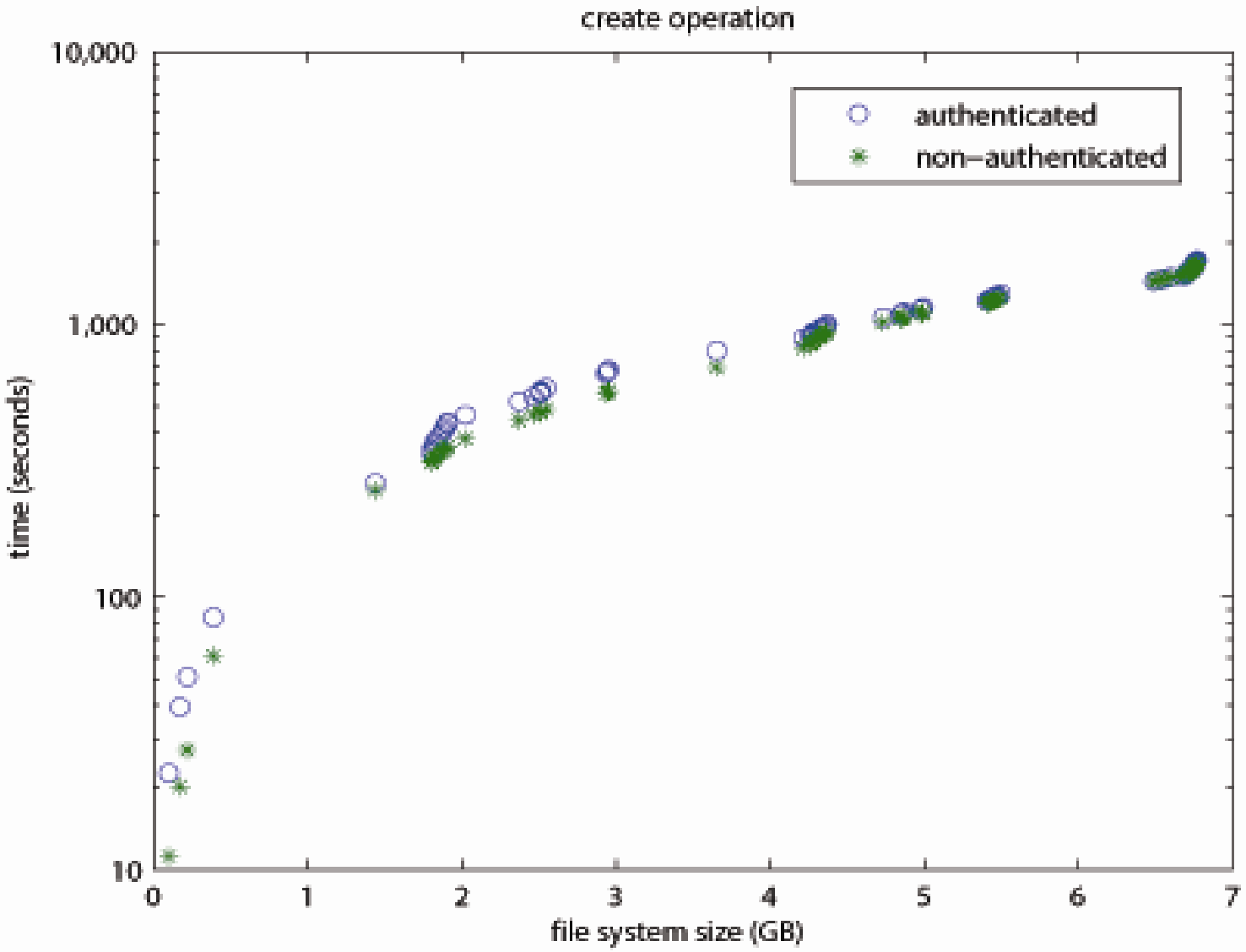
Authenticated dynamic trees

- Direct application of [CT-RSA 2003 Goodrich, Tamassia, Triandopoulos]
- Suppose we have any tree T of n nodes
- All nodes v have some property, say $l(v)$
- The property of a path P is defined as a concatenation of nodes properties (e.g., path name property)
- Let P be a path
- Authenticating path properties $O(k + \log n)$
- Authenticating updates: $O(k + \log n)$
- Better than $O(k \log n)$ and efficient mv/rmdir!
- Practical only for a depth >120 (Werneck and Tarjan, WEA 2007)

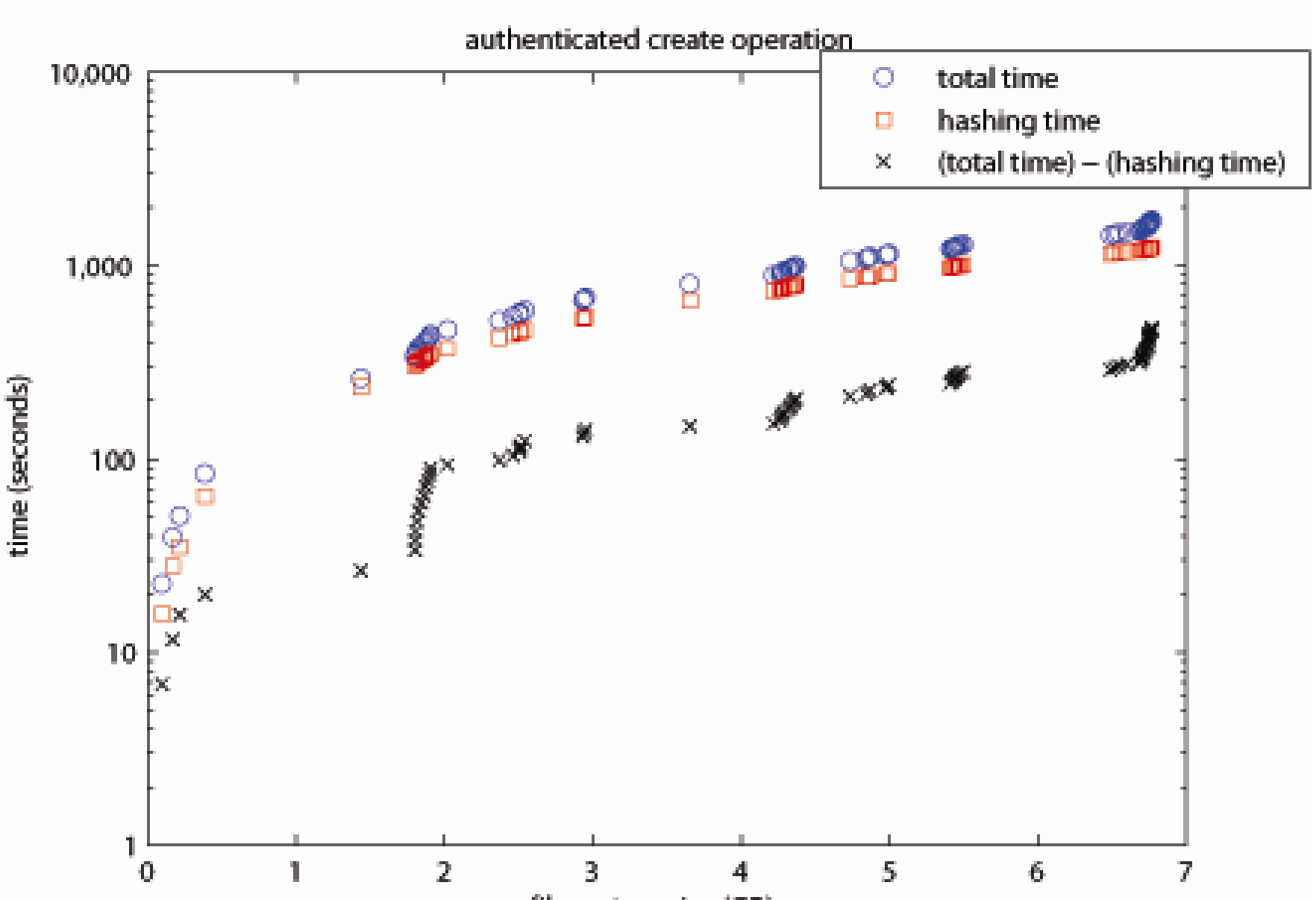
Theoretical Comparison

Operation	Skip lists (local)	Skip lists (global)	Dynamic trees
cd, read, write, rm, mkdir, touch	$k \log n$	$\log n + k$	$\log n + k$
ls	$(k+l) \log n$	$l(\log n + k)$	$k + l + \log n$
rmdir	$(k+t) \log n$	$t \log n + k$	$k + \log n$
mv	$(k+k') \log n$	$t \log n + k + k'$	$k + k' + \log n$

Experiments on a remote 7GB file system



How much does hashing cost?



Conclusions

- A scalable solution for authenticating outsourced file system
- Authentication of hierarchy
- Authentication of major FS operations
- Logarithmic costs
- Three different theoretical approaches
- A working prototype justifying theoretical findings