

# Proving Confidentiality in a File System Using DiskSec (Poster #2)

OSDI '18

Atalay Mert İleri, Tej Chajed,  
Adam Chlipala, Frans Kaashoek, Nickolai Zeldovich  
MIT CSAIL

# Storage Systems Contain Confidential Data

Users rely on the storage system to maintain their confidentiality.

- A file system will be used as a case study in this talk.

# Confidentiality in a File System

- Alice and Bob share a file-system on the same machine
- Bob tries to learn the content of Alice's files

**Threat model:** Bob can call the file-system interface and cannot bypass it.

- can't steal the disk
- can't read or write directly to the disk etc.

# Bugs May Leak Confidential Data

File-systems are also subject to confidentiality bugs.

## Examples

- Crash can expose deleted data (ext4 - 2017)
- Anyone can change POSIX ACLs (NFS - 2016)
- Truncated data can be accessed (btrfs - 2015)
- Crash can expose data (ext4 - 2014)
- Anyone can change POSIX ACLs (btrfs, gfs2 - 2010)
- ...

# Approach: Formal Verification

- Write a specification that captures the desired behavior of the system.
- Prove that implementation satisfies the specification.
- As long as specification accurately captures the desired behavior, implementation details are irrelevant.
- We have verified file systems with correctness specifications (e.g. DFSCQ [SOSP'17]).

# Functional Specifications Do Not Ensure Confidentiality

Functional specifications ensure many security properties.  
(e.g. no memory corruption, no disk corruption etc.)

## **Example: Specification for readdir**

`readdir` can return entries in any order.

<code>/dir_a</code>	<code>readdir(...) ⇒ dir_a, dir_b</code>	✓
<code>/dir_b</code>	<code>dir_b, dir_a</code>	✓

# Functional Specifications Do Not Ensure Confidentiality

```
def readdir(...)
  dirs = get_dirlist(...)
  if (alice.txt file contains 'a')
    return sort(dirs)
  else
    return reverse_sort(dirs)
```

- Meets specification
- Leaks confidential data

Nondeterministic functional specifications allow breach of confidentiality.

Confidentiality requires better specifications.

# State of the Art in Verifying Confidentiality

## Existing Systems

- seL4 [SSP'13]
- Ironclad [OSDI'14]
- CertiKOS [PLDI'16]
- Komodo [SOSP'17]
- Nickel [OSDI'18]

Above systems use non-interference for their confidentiality specifications.

Non-interference does not allow **any** data exposure from Alice to Bob.



# Non-interference is Not Suitable for File System Confidentiality.

- File systems have discretionary access control
- File systems intentionally expose metadata.

# Contributions

## **DiskSec**

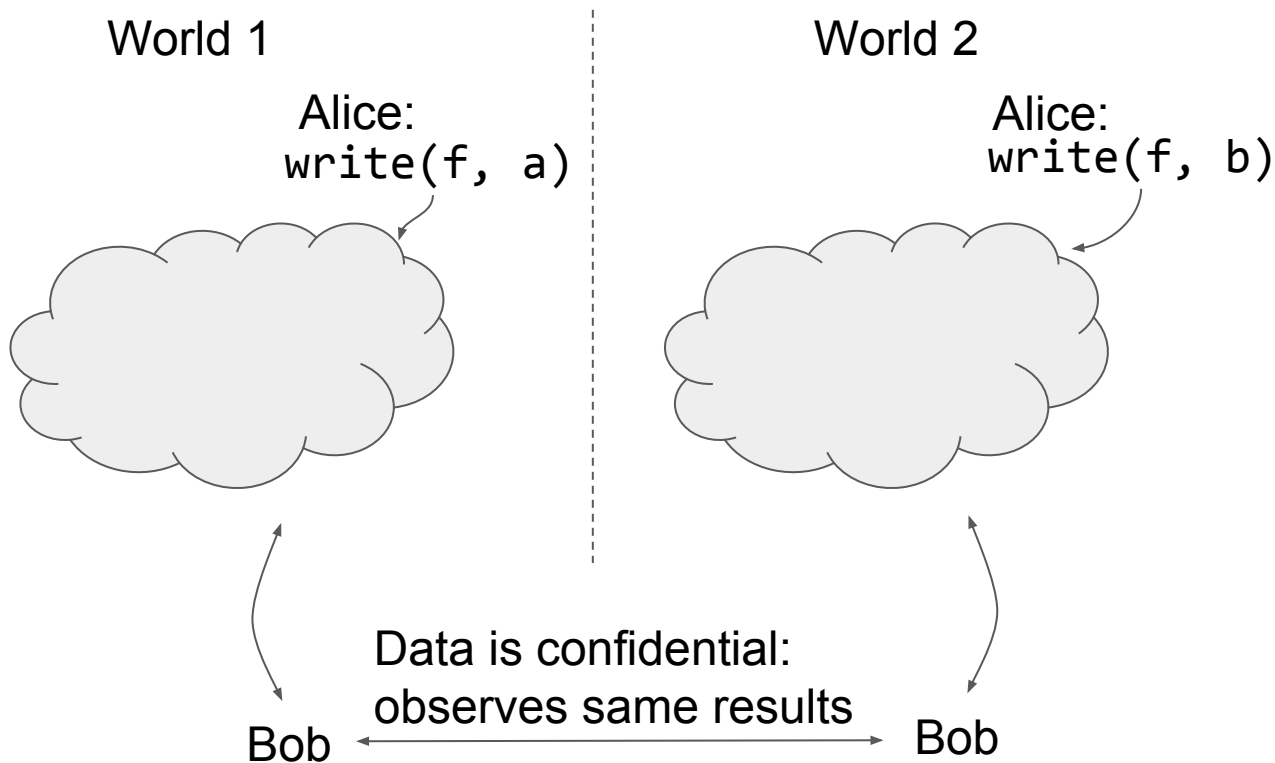
Framework for proving confidentiality of storage systems.

- File-system confidentiality specification.
- Proof technique to track ownership of the data.
- DiskSec implemented and proven in Coq Proof Assistant.

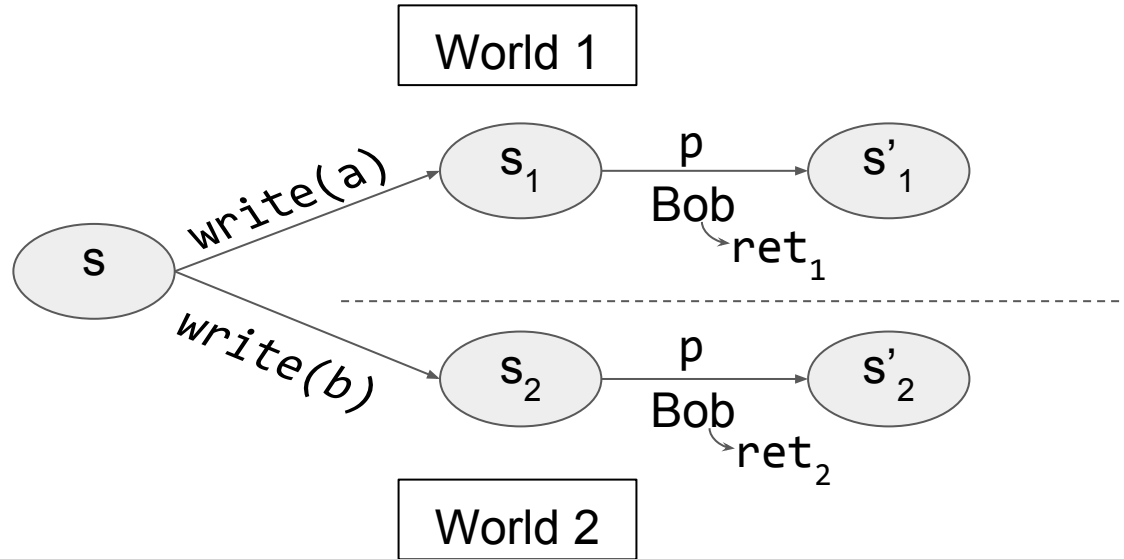
## **Evaluation**

- SFSCQ file system: extension of DFSCQ with confidentiality theorem
- Confidentiality for simple app on top of SFSCQ

# Bob Cannot Infer Alice's Confidential Data



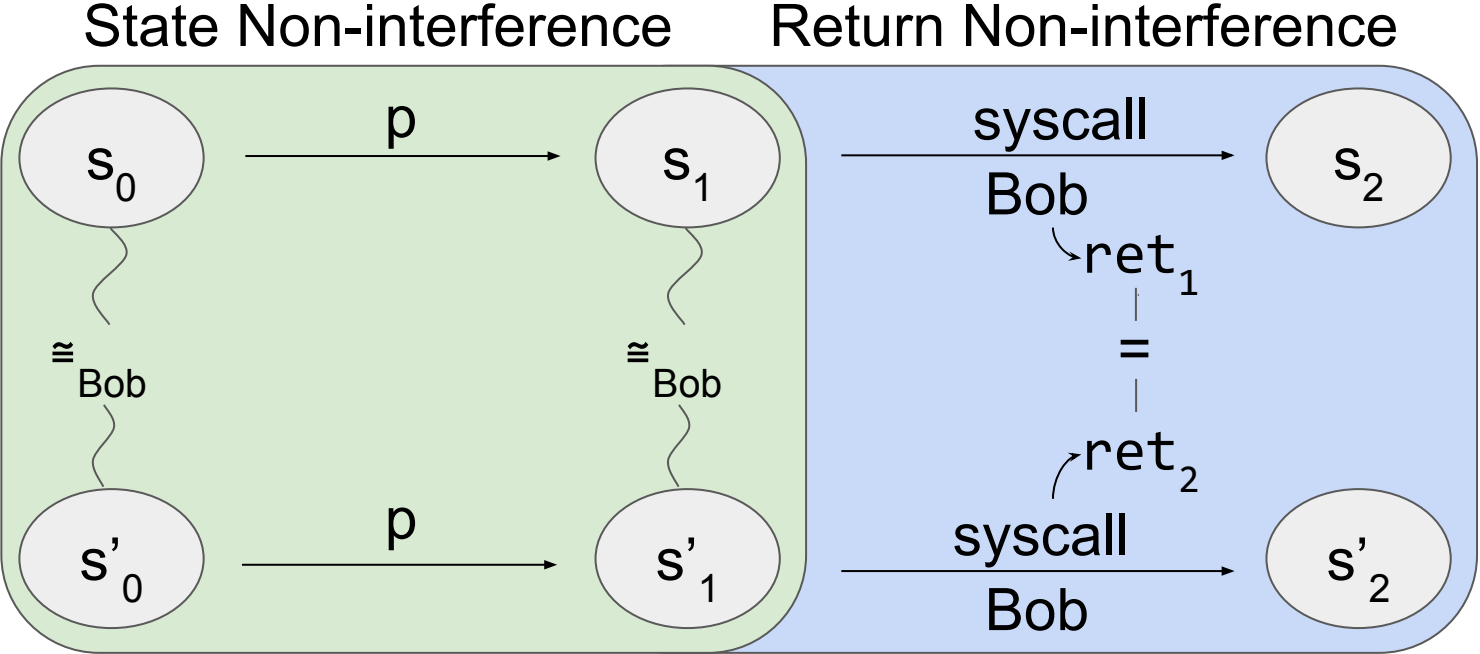
# Confidentiality Means Other Users See Same Thing Regardless of Your Data



Confidentiality  
requires that  
 $ret_1 = ret_2$

Two states are equivalent with respect to a user ( $\cong_{user}$ ),  
if all the data **visible to that user** is the same in both states.

# Our Confidentiality Specification: Data Non-interference



# Data Non-interference is a Good Confidentiality Specification for File Systems

## Data non-interference

- allows discretionary access control,
- allows exposing of metadata,
- forbids exposing of user data
  - even indirectly (e.g. readdir)

# How can We Prove Data Non-interference?

Data non-interference require more complicated proofs than functional correctness.

- Require reasoning about behavior of two executions.

**Insight:** File systems mostly does not inspect user data.

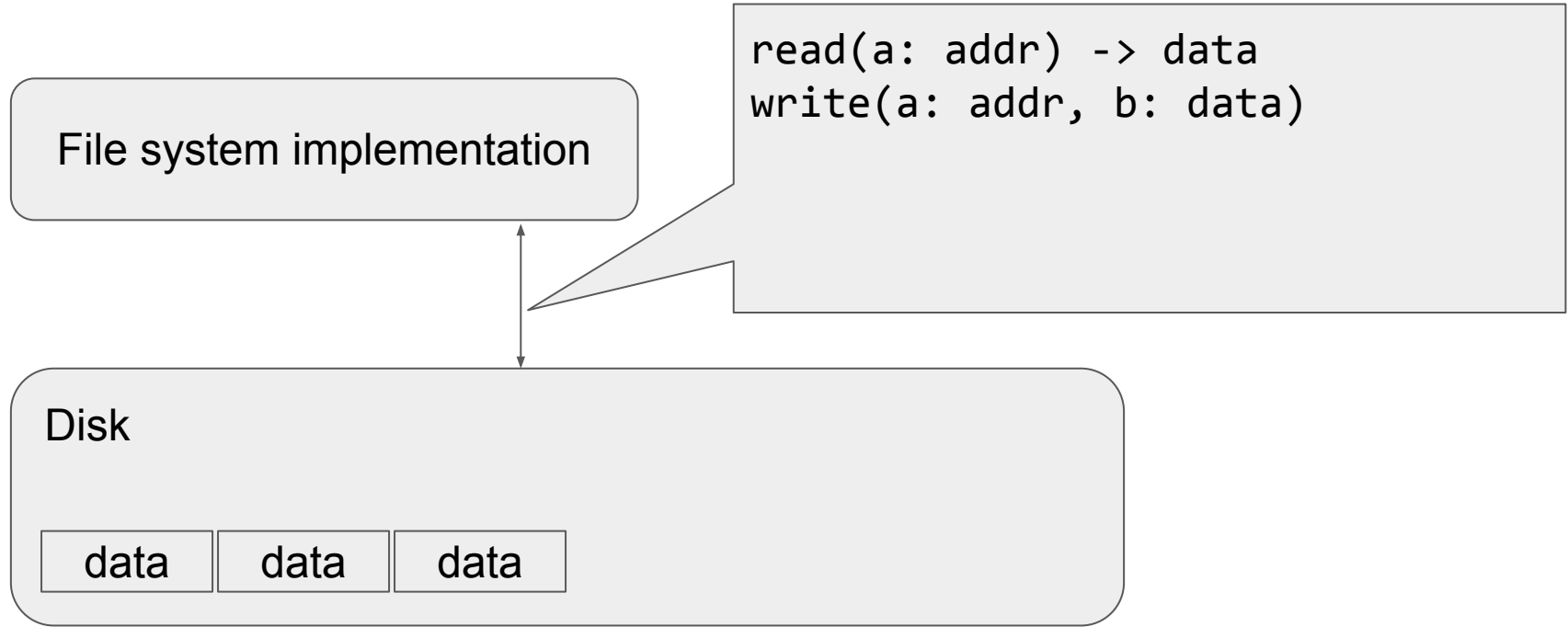
- Suffices to reason about where user data is accessed in one execution.

# Our Approach: Sealed Blocks

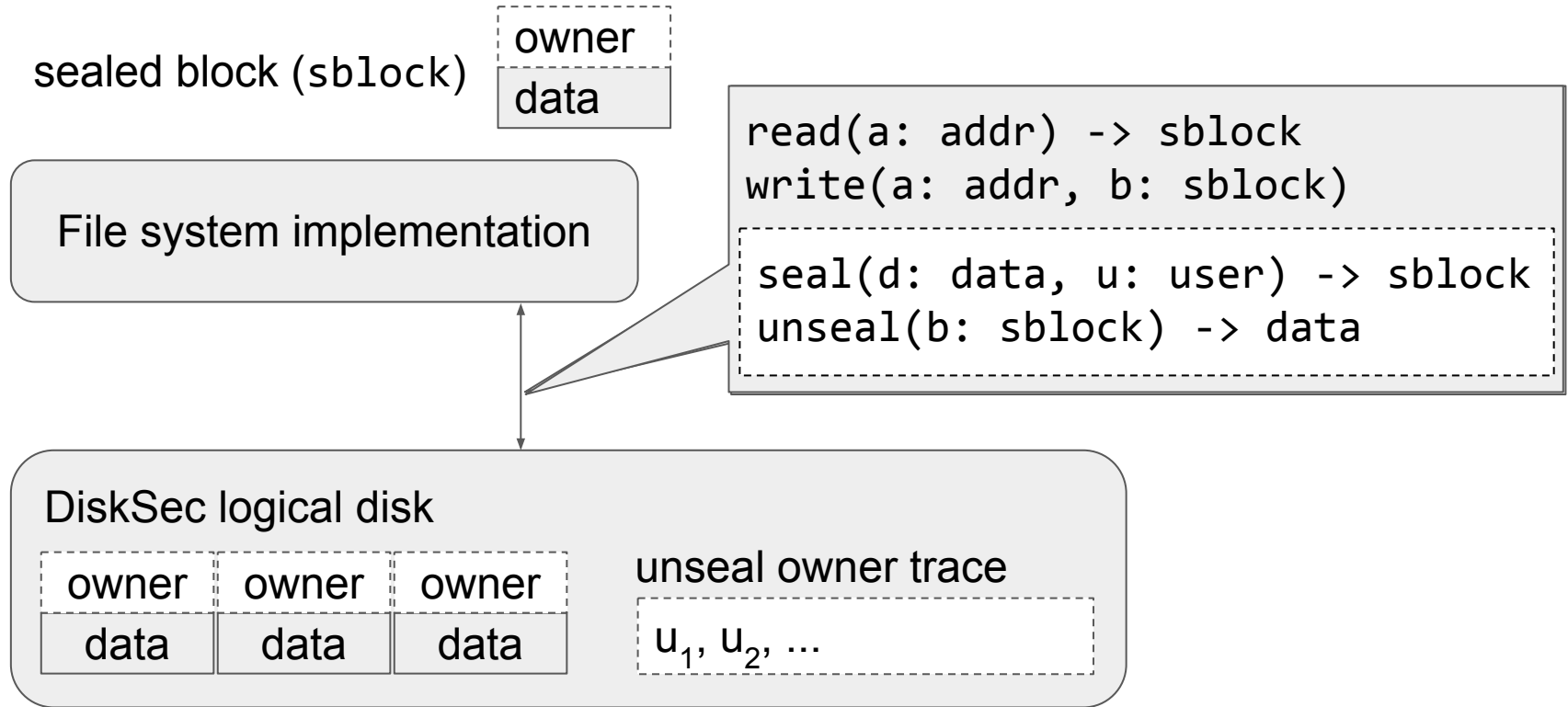
- **Pretend** that all disk blocks are **logically** sealed.
- Function needs to request an **unseal** to access the data content.
- Functions can be analyzed to prove that they do not unseal user data.



# Standard Disk Infrastructure



# DiskSec Infrastructure



# How to Use DiskSec?

1. Developer instruments his code with seals, unseals and access control checks.

## Standard Implementation

```
def read(f, ...)
  data = read_disk(f, ...)
  return data
```

## DiskSec Implementation

```
def read (f, ...)
  if (can_access(f))
    sealed_data = read_disk(f, ...)
    data = unseal(sealed_data)
    return data
  else
    return error
```

2. Developer proves that a certain property holds for the unseal trace of the implementation.

# Sealed Blocks Simplify Confidentiality Proofs

## Unseal Public

Function only unseals data accessible to **every user**.

Unseal Public → Data Non-interference

## Unseal Secure

Function only unseals data accessible to the **current user**

Unseal Secure → Return Non-interference

In this case, state non-interference needs to be proven separately.

# DiskSec Summary

- Provides infrastructure for access control in storage systems.
- Formalizes data non-interference as a confidentiality specification.
- Simplifies proof effort by reducing data non-interference proofs to unseal trace proofs.

# Applying DiskSec: SFSCQ Overview

- Based on DFSCQ [SOSP'17]
- Supports multiple users
- Simplified permission model
  - All metadata, including file names, are public.
  - File contents may be public or private.
  - File owner is set upon creation.
- Fully implemented and verified in Coq Proof Assistant

# Evaluation

- Did we prove DFSCQ satisfies data non-interference?
  - Not completely.
  - Needed to remove an advanced feature.
- Is performance the same as DFSCQ?
  - SFSCQ code = DFSCQ code + access control checks
- How much effort did it require?
  - Took one author ~3 months

# Conclusions

- Correctness specifications are not enough for confidentiality.
- Data non-interference is a suitable confidentiality specification for file systems.
- We designed and implemented DiskSec, a framework for confidentiality proofs for storage systems.
- We implemented SFSCQ, the first file system with machine-checkable confidentiality proofs, using DiskSec.

<https://github.com/mit-pdos/fscq/tree/security>