Separating key management from file system security

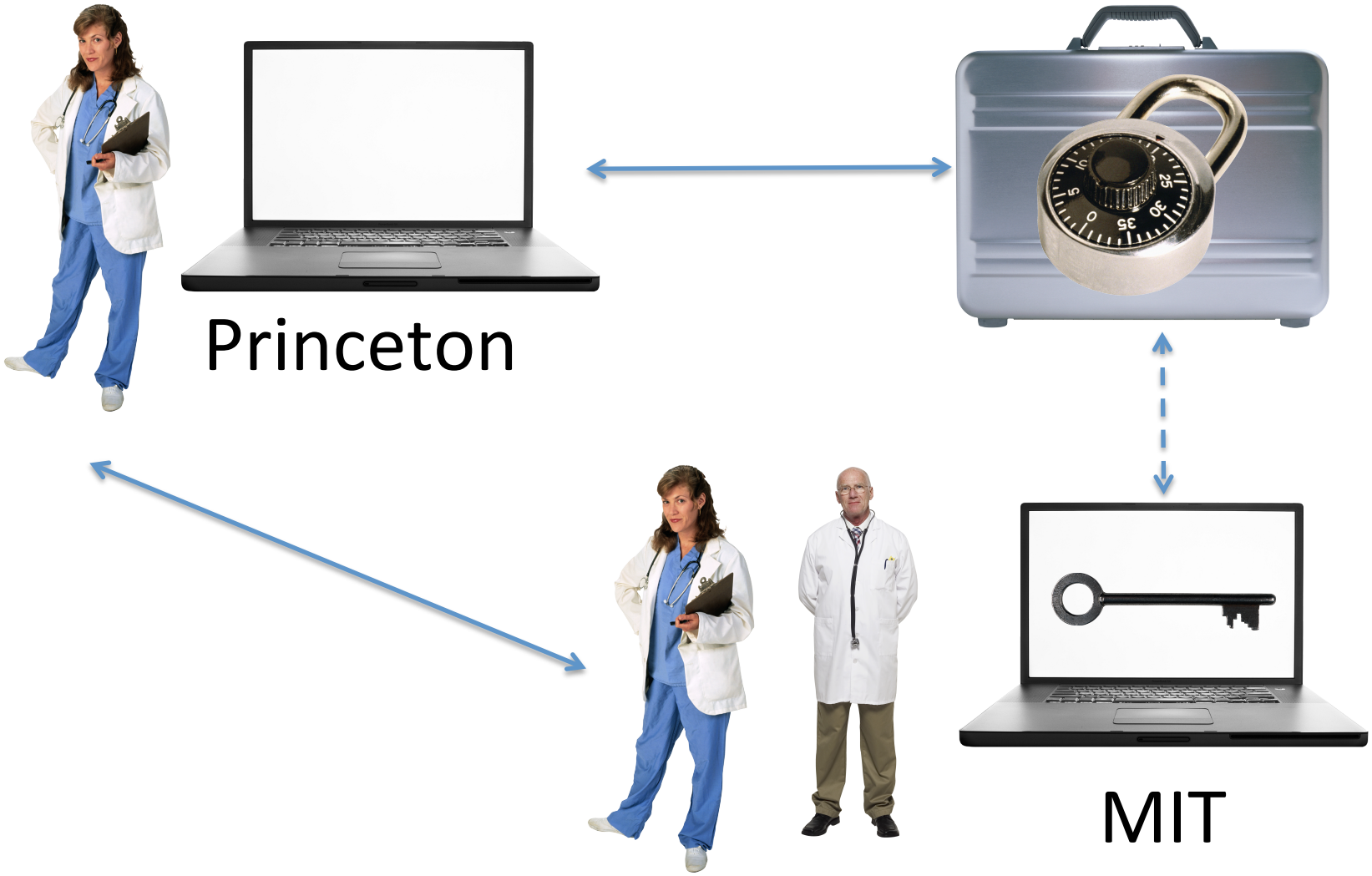
David Mazières, Michael Kaminsky, M. Frans Kaashoek and Emmet Witchel.
MIT LCS

SOSP 1999

The problem.

- Problem:
  - Current file systems lack key management that scales globally (say, for the internet) and is decentralized.
  - Attackers can easily tamper with network traffic, current file systems are insecure.
  - NFS' security is very weak and does not scale well.
  - Most large scale systems need third party (like Verisign) to authenticate.
  - They claim SSL is too complicated for web servers to implement.

Motivational Example



Solution: Self-certifying pathnames!

- Create a self-certifying file system (SFS) that has the following properties:
- Security:
  - Attackers cannot read or modify the file system, but can intercept, modify or inject new packets.
  - Assume attackers are computationally bounded and it is assumed that factoring is hard.
- Versatility:
  - Support a wide variety of authentication methods.
- Global file system:
  - It doesn't matter which client a person uses. What must matter is the user.

Design

- The key idea: self-certifying pathnames!
- SFS servers are accessible under a pathname of form sfs/Location:HostID.
- HostID is crypto hash of Server's Public Key and Location.
- Secure collision-resistent crypto hash: SHA-1.
- Path on remote server appended to the end.

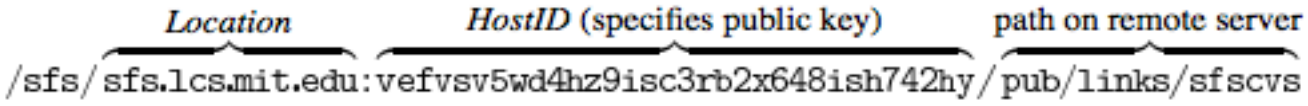


Figure 1: A self-certifying pathname

Design: Important high level ideas.

- Many ways to turn self-certifying pathnames into nice human readable paths or include other key management methods:
  - Using symbolic links.
  - Creating secure bookmarks.
  - Certification authoritites.
  - Password authentication.
- If a server's private key is disclosed, then an attacker can pass malitious files to server. Two mechanisms:
  - Revoke key, which is done by the owner of the file system.
  - Block HostID.

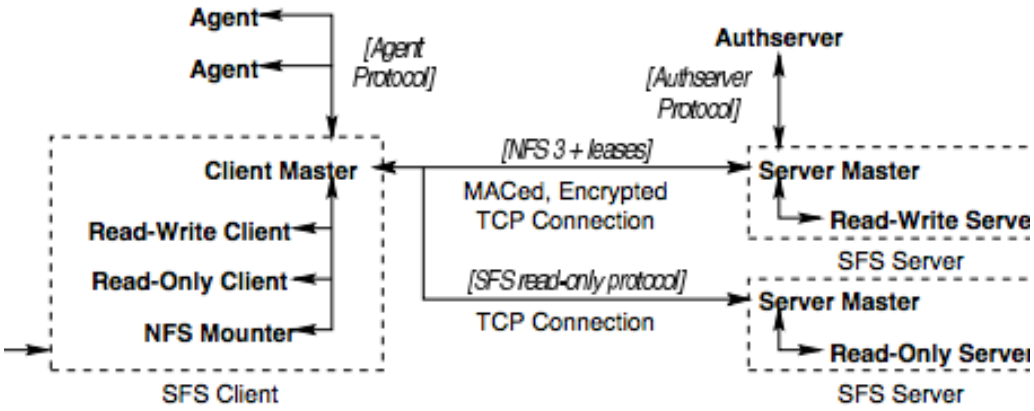


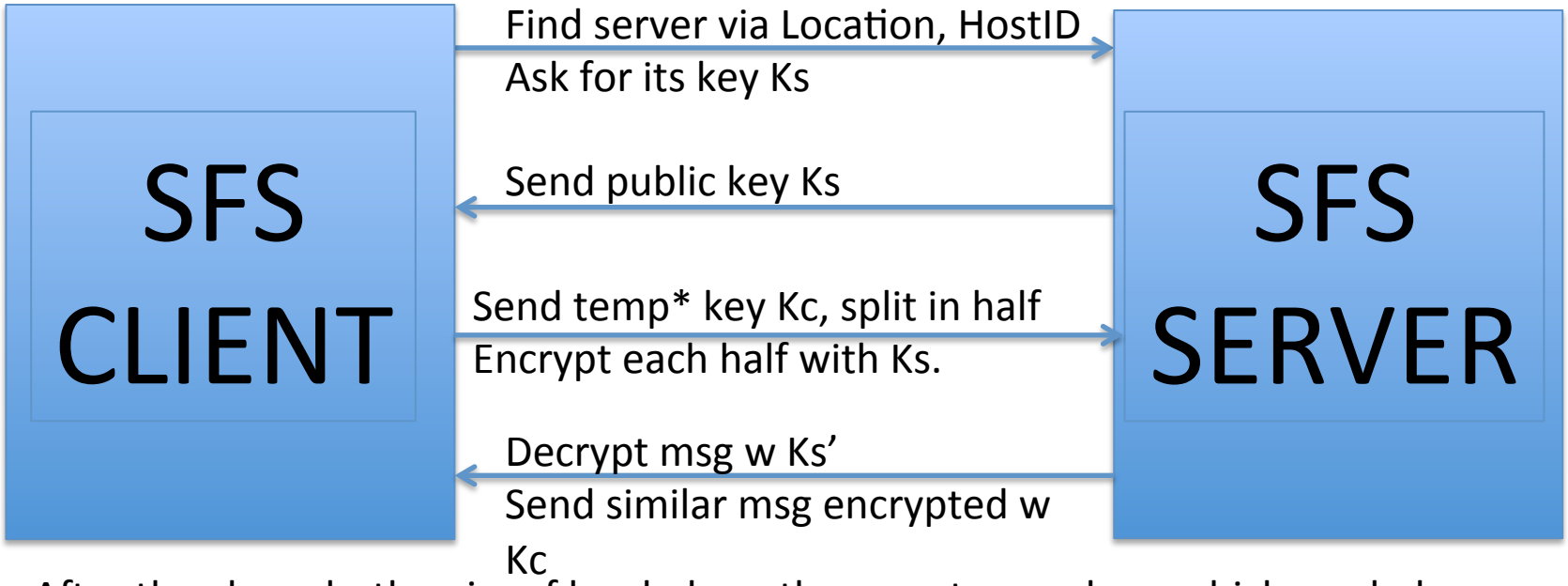
Figure 2: The SFS system components

Implementation: client and server authentication.

Need to establish secure channel on first contact.

Animation based from figure in [1].

*= key is changed periodically to prevent attacks by guessing halves.



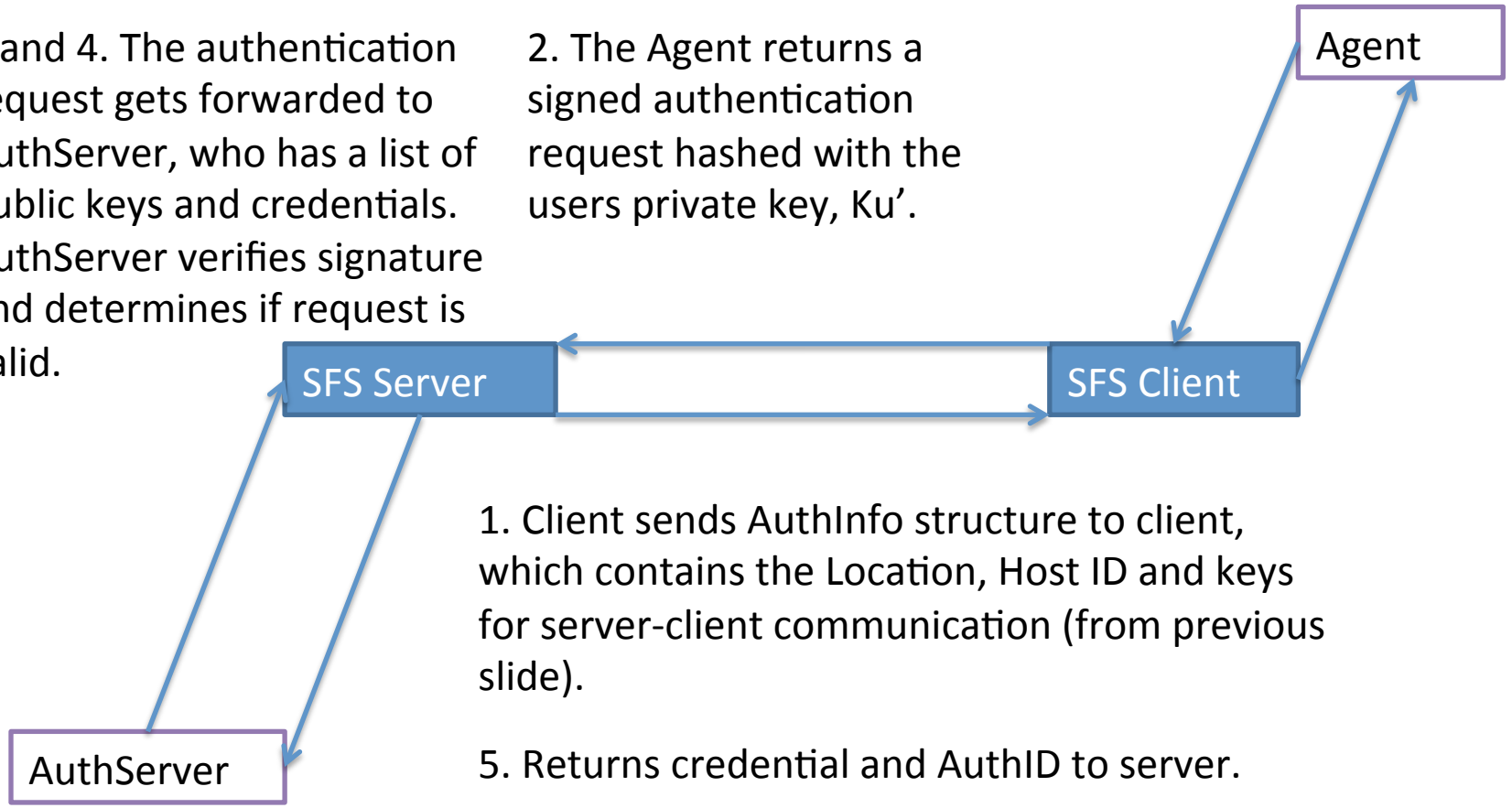
After they have both pairs of key halves, they create new keys which use halves from both keys. They will use these keys to communicate. Moreover, the client knows that the server is who he claims he is.

Implementation: User and server authentication.

Animation based off from figure in [1].

3 and 4. The authentication request gets forwarded to AuthServer, who has a list of public keys and credentials. AuthServer verifies signature and determines if request is valid.

2. The Agent returns a signed authentication request hashed with the users private key, Ku'.



1. Client sends AuthInfo structure to client, which contains the Location, Host ID and keys for server-client communication (from previous slide).

5. Returns credential and AuthID to server.

6. Server assigns authentication number to credentials, returns number to client.

Experimental results

- Set up:
  - Compared SFS with and without cryptographic overhead versus NFS 3 over UDP and NFS 3 over TCP.
- Experiment 1: Microbenchmarks
  - SFS has worse throughput and latency in both cases.
  - Authors attribute this to large user-implementation cost, not cryptographic overhead.

File System	Latency (μsec)	Throughput (Mbyte/sec)
NFS 3 (UDP)	200	9.3
NFS 3 (TCP)	220	7.6
SFS	790	4.1
SFS w/o encryption	770	7.1

Figure 5: Micro-benchmarks for basic operations.

Experimental results

- Experiment 2: E2E performance on MAB and a larger application (compiling a kernel).
  - Results show that user-level implementation of SFS is bottleneck.

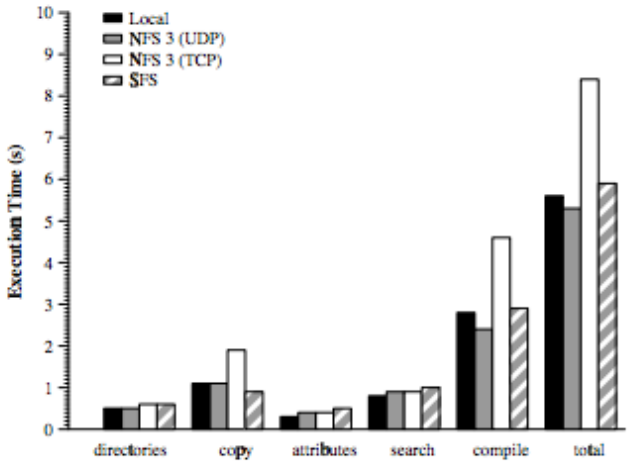


Figure 6: Wall clock execution time (in seconds) for the different phases of the modified Andrew benchmark, run on different file systems. Local is FreeBSD's local FFS file system on the server.

System	Time (seconds)
Local	140
NFS 3 (UDP)	178
NFS 3 (TCP)	207
SFS	197

Figure 7: Compiling the GENERIC FreeBSD 3.3 kernel.

Subsequent work

- This was David Mazières' PhD Thesis.
- David Euresti made a Windows implementation in 2002.
- Even though SFS didn't get a lot of attention, the idea of separating key management from file systems is still relevant today (e.g. UIA paper).
- There's been a lot of work in designing secure file systems (Ori, Shark), specially for distributed systems.
- Authors claim they use SFS numerous times in the paper… Not so used in practice.

Final thoughts.

- Advantages:
  - Decentralized, global, reasonably secure file system.
  - Separates key management from user certification.
  - Modular and portable design.
  - Their model gives attackers a lot of power.
- Disadvantages:
  - Only implemented for UNIX systems.
  - Base performance is worse than current systems.
  - SHA-1 is no longer considered too secure.
  - The authors fail to address why their system is better than SSL In a technical way.
- Classmates Input.

References

- [1] Mazieres, Kaminsky, Kaashoek, Witchel. Separating Key Management from File System Security. MIT LCS, 1999
- [2] Fu, Kamisky, Mazieres. Using SFS for a Secure Network File System. ;login: The magazine of USENIX & SAGE, volume 27, number 6, December 2002.