

Fault-Tolerance and Security

when opposites attract

Jian Yin & Arun Venkataramani
& Jean-Philippe Martin & Lorenzo Alvisi & Mike Dahlin

University of Texas at Austin

The Problem: Intuition

- Example: Online medical history
- Should we replicate confidential data?
- Tension between:
 - ◇ *Availability*: Service remains functional despite failures
 - ◇ *Confidentiality*: Some data is not accessible to adversaries despite intrusions

The Problem: Fundamentals

(N is the normal case, F represents failures)

- Availability:

Accessible for X in $N \implies$ accessible for X in F

- Confidentiality:

Accessible for X in $F \implies$ accessible for X in N

- Replication-based systems typically only provide the former

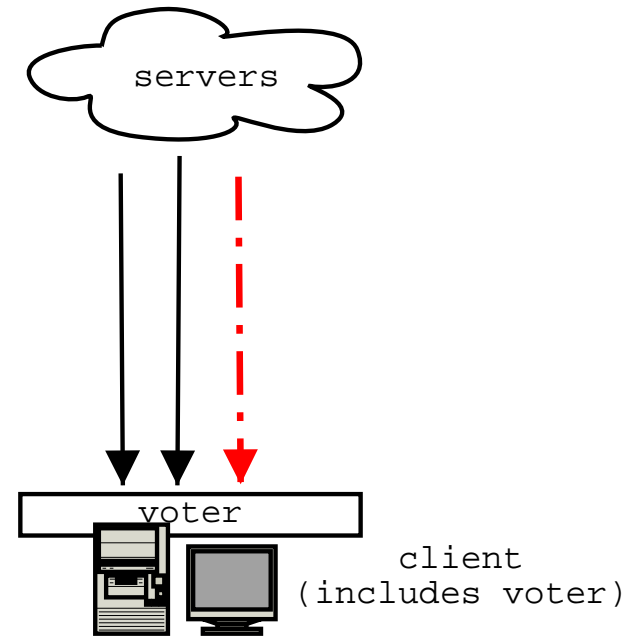
Outline

- The problem: fault-tolerance and security
- Existing techniques that do not fit the bill
- Design space
- Privacy Firewall
- Conclusion

Byzantine Fault Tolerance...

...and why it's not enough

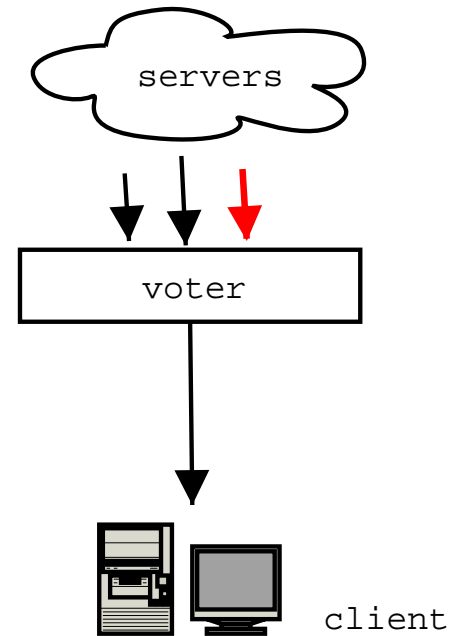
- Principle:
 - ◇ Query sent to several servers
 - ◇ Majority rule
- Provides:
 - ◇ Availability and integrity despite break-ins
- Shortcoming:
 - ◇ Client sees all answers
 - ◇ Attacker can still access confidential data



Byzantine Fault Tolerance...

...and why it's not enough

- Principle:
 - ◇ Query sent to several servers
 - ◇ Majority rule
- Provides:
 - ◇ Availability and integrity despite break-ins
- Shortcoming:
 - ◇ What if the voter fails?



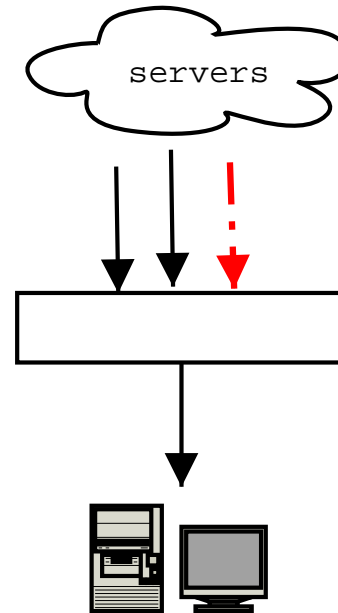
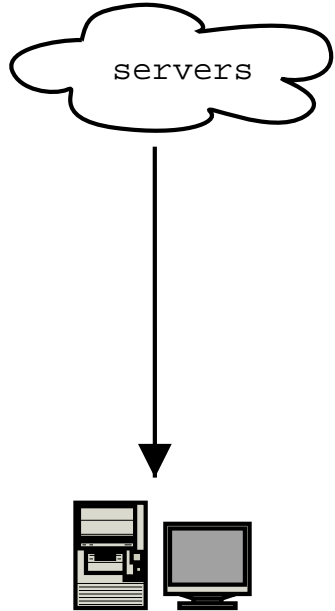
Design Space: Basis

Confidential Fault-Tolerant Service

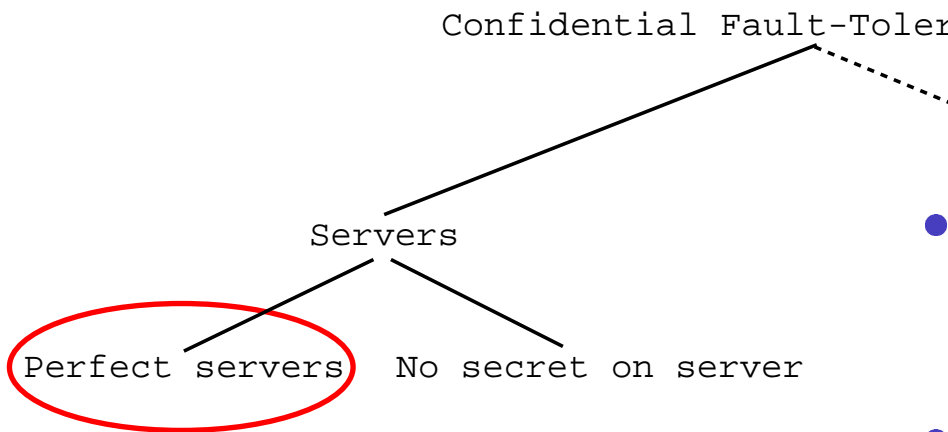
Servers

OR

Interface

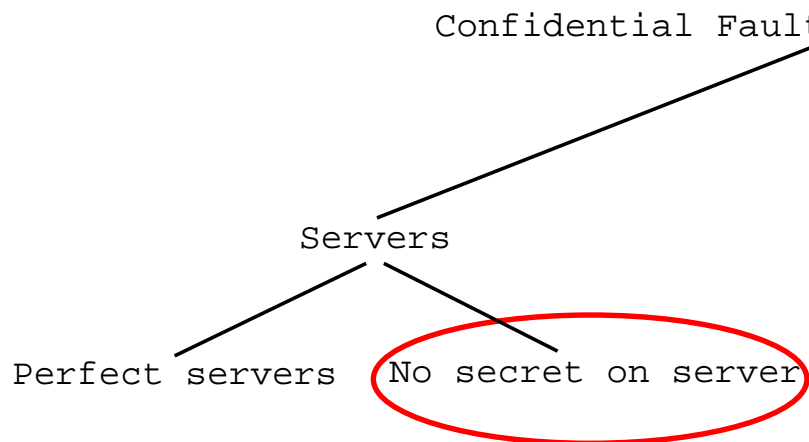


Design Space: Server Side



- A server with no bugs will not leak secrets
- Formal verification
- Need to verify the operating system as well

Design Space: Server Side

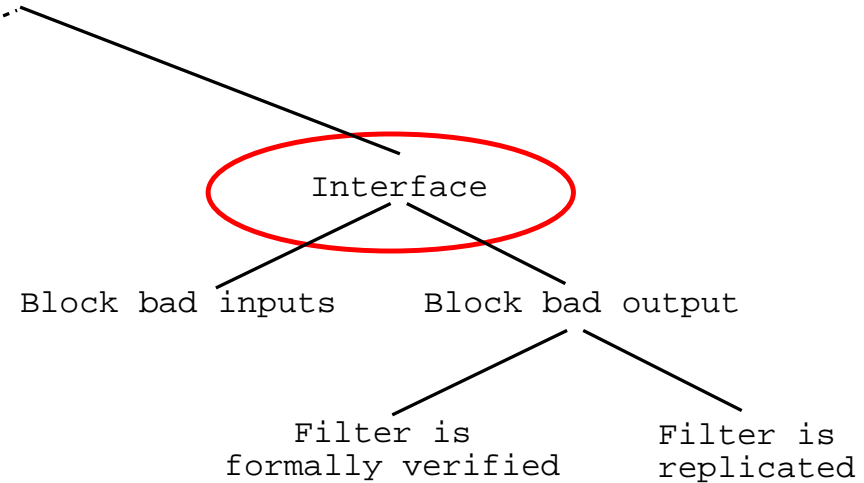


- Byzantine storage with encrypted data (Malkhi, Reiter)
- COCA (Zhou, Schneider, van Renesse)
- Service-specific

Design Space: Interface Side

- Assume BFT back end
- Applicable to all existing services
- Commodity filter

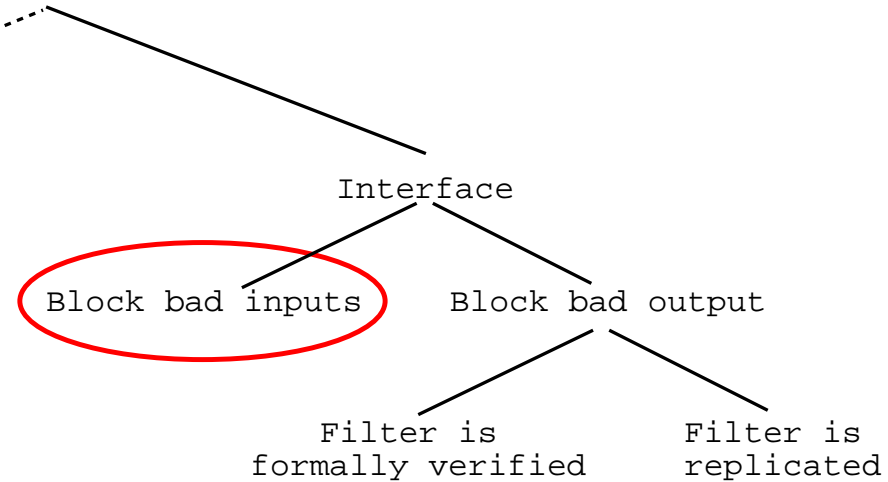
Confidential Fault-Tolerant Service



Design Space: Interface Side

- Example: firewall
 - ◊ Block “bad” inputs
- Need to know all the bugs
- Service-specific

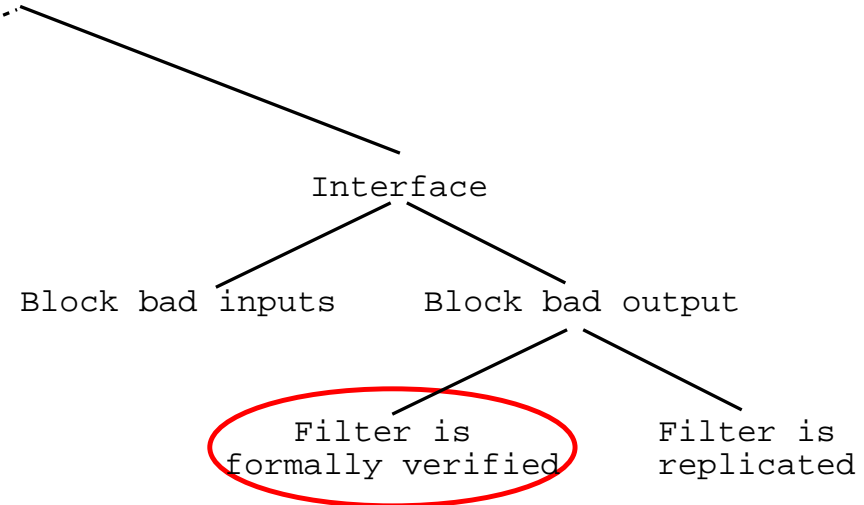
Confidential Fault-Tolerant Service



Design Space: Interface Side

- Promising
- Again, need to verify OS as well
- Service independent

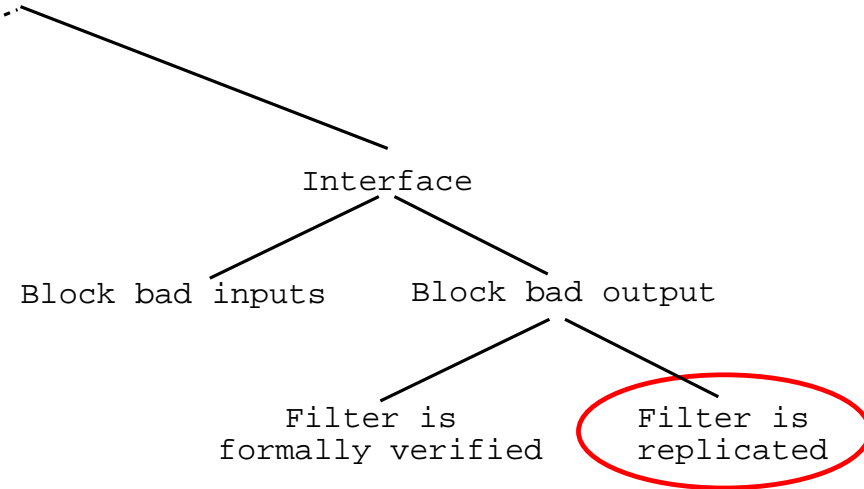
Confidential Fault-Tolerant Service



Design Space: Interface Side

- Same as replicating servers?
- No: easier because there is no confidential data
- Service independent
- Our approach

Confidential Fault-Tolerant Service



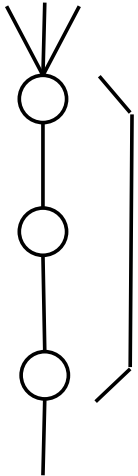
Our Contribution

- Result: Privacy Firewall
 - ◇ Confidentiality through replication
 - ◇ Accessible in $N \iff$ Accessible in F
- Adversary can:
 - ◇ Take control of a threshold of machines
 - ◇ Authenticated asynchronous channels
- Currently evaluating a prototype

Exploring the Privacy Firewall Idea

- Service guarantees availability and integrity
- Privacy Firewall has no confidential data

Servers
(no other access to the outside)



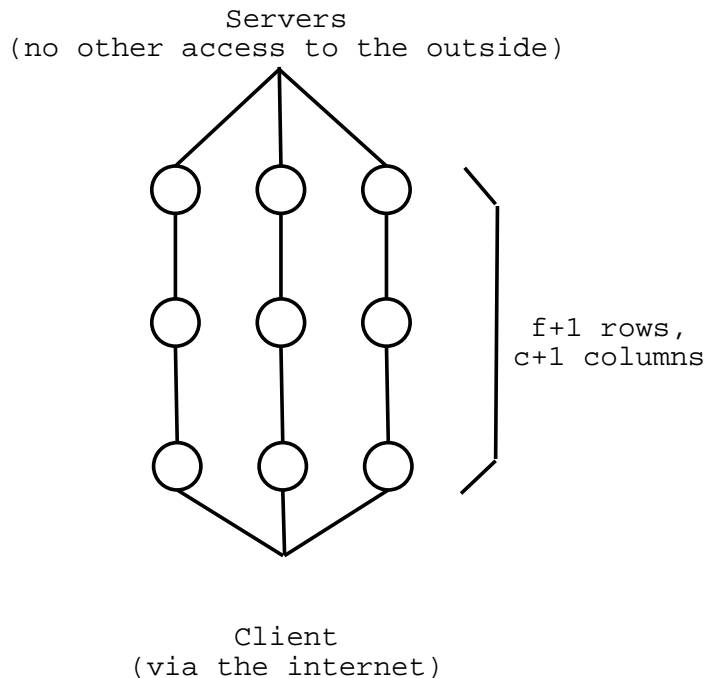
Client
(via the internet)

- Machines verify that answers have been vouched for by $s+1$ servers
- No confidential information gets through
- What if firewall machines fail?

Privacy Firewall Protocol

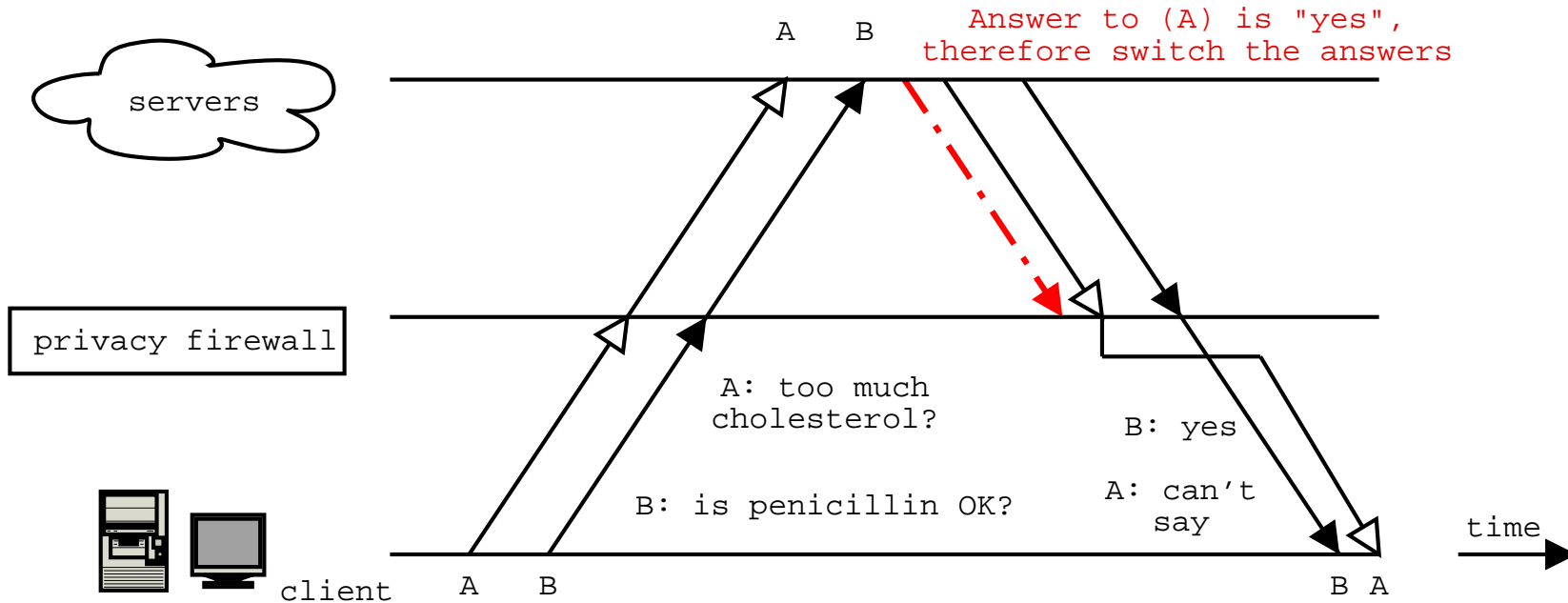
- Intuition: Each PF machine is an identical filter
- On query:
 1. Forward upwards
- On reply:
 1. Check order
 2. Check that it is vouched for by $s+1$ servers
 - ◇ Threshold signature
 3. Forward downwards

Privacy Firewall: Availability



- Confidential for up to f Byzantine failures
- Available for c Byzantine failures
- About size:
 - ◇ Hardware is cheap
 - ◇ Minimal number of machines
 - ◇ f small
- Covert channels?

Privacy Firewall: Covert Channel



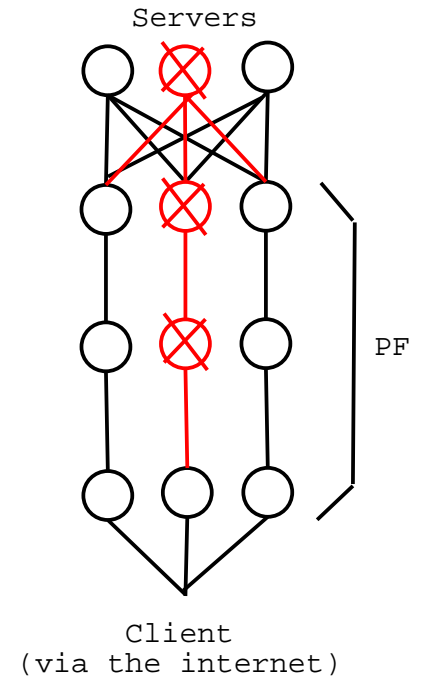
- Reordering allows “Yes/No” secret answers
- Correct protocol must forbid reordering
- Simple example of a hard problem

Privacy Firewall: Confidentiality

- Risks: Covert channels to leak information
- Solution: Ensure protocol leaves no choice
- Avoid:
 - ◇ Unchecked fields in the header
 - ◇ Reordering of requests or replies or signatures
 - ◇ Dropping replies
 - ◇ Non-determinism (already necessary)
 - ◇ Separate signatures

Intuition of Correctness

- Availability
 - ◇ Always one entirely correct chain
- Confidentiality
 - ◇ Always one correct PF machine in each chain
 - ◇ Above: private data but no choice
 - ◇ Below: choice but no private data



Conclusion

- “Always on” services need Fault-Tolerance and Security
- Fault-Tolerance and Security: a challenge
 - ◇ Replication generally hurts confidentiality
- Privacy Firewall: new approach
 - ◇ Harnesses replication to improve confidentiality
- Confidentiality is an important requirement
... but often forgotten!

Discussion Starter

- Privacy Firewall
- Other possible approaches:
 - ◇ Perfect servers
 - ◇ No secrets on servers
 - ◇ Filter inputs
 - ◇ Filter outputs
 - ▷ Formally verified filter
 - ▷ Replicated filter
 - ◇ ...