Building Trustworthy Distributed Services*

Fred B. Schneider
Department of Computer Science
Cornell University
Ithaca, New York 14853
U.S.A.

*Joint work with Lidong Zhou, Tom Roeder, and Riccardo Pucella.
Trustworthy Services

A trustworthy service...

- tolerates component failures
- tolerates attacks intended to compromise:
  - availability
  - integrity
  - confidentiality
State Machine Approach

The basic recipe ...

- **Servers:**
  - deterministic state machines
  - assumed to fail independently

- **Clients:**
  - make requests
  - synthesize service response from individual server responses
State Machine Approach

Supports:
- Confidentiality
- Integrity
- Availability

of whatever service is provided by a single replica.
State Machine Approach Internals

Internals:

- **Agreement protocol** so all correct servers process requests in same order.
State Machine Approach Internals

Internals:

- **Agreement protocol** so all correct servers process requests in same order.
- **Authentication protocol** so client can distinguish and synthesize responses from different servers.
Revisiting the “Fine Print”

- Agreement protocol for replica coordination.
  - But such protocols involve assumptions, and assumptions are vulnerabilities.
    - Timing assumptions versus Denial of Service.
    - Non-assumption: Asynchronous System Model.
Revisiting the “Fine Print”

- **Agreement protocol for replica coordination.**
  - But such protocols involve assumptions, and assumptions are vulnerabilities.
    - Timing assumptions versus Denial of Service.
    - Non-assumption: Asynchronous System Model.

- **Authentication protocol requires servers to keep secrets (viz private keys).**
  - Secret refresh is not scalable for clients. Solution:
    - (Asynchronous) Proactive secret sharing +
    - threshold digital signatures
Revisiting the “Fine Print”

- **Agreement protocol for replica coordination.**
  - But such protocols involve assumptions, and assumptions are vulnerabilities.
    - Timing assumptions versus Denial of Service.
    - Non-assumption: Asynchronous System Model.

- **Authentication protocol requires servers to keep secrets (viz private keys).**
  - Secret refresh is not scalable for clients. **Solution:**
    - (Asynchronous) Proactive secret sharing +
    - threshold digital signatures

- **Replica failures are independent.**
  - But attacks invalidate this. **Solution:**
    - Proactive obfuscation
Compromised Components

Correct component satisfies specification.

Compromised component does not.

- Caused by failure or attack.
- Adversary might control a compromised component.
  - Adversary learns secrets stored at compromised component. These might allow other components to be compromised.
    - E.g. Cryptographic keys to support secure channels.
Proactive Recovery

**recovery protocol**: transforms component compromised → correct

- **Proactive** recovery defends against undetected failures/attacks.
  - tolerates t compromises over *lifetime*
- **versus** -
  - tolerates t compromises in *window of vulnerability*
Nature of the Adversary

Denial of Service attacks can lengthen a window of vulnerability.

Possible restriction on adversary power:

- Adversary’s ability to compromise depends on time available.

- **versus** -
  - Adversary’s ability to compromise depends on intrinsic aspects of servers.
Server Response Authentication:
Service Key versus Server Keys

t+1 servers *speak for* the service.

Desire:

- Any set of t+1 servers can cooperate to sign a response.
- No set of t or fewer servers can contrive to sign a response.

Client accepts response “signed by service”.
Server Response Authentication:
Implementing Service Key

- (n,t) secret sharing [Shamir, Blakley]:
  - Secret s is divided into n shares.
  - Any t or more shares suffice for reconstructing s.
  - Fewer shares convey no information about s.

- Threshold cryptography:
  - Perform cryptographic operations piecewise using shares of private key; result is as if private key was used.
    
    Example: Threshold digital signatures.
Server Response Authentication:
Defense Against Mobile Adversary

Mobile adversary: Attack, compromise, and control one replica for a limited time.

- Adversary accumulates shares of secret key.
- Defense: Re-share service’s private key as part of proactive recovery.
  - Create new, independent sharing of service key.
  - Replace old shares with new shares.
  - Protocol must not materialize service key.
Independence for Secrets

Proactive Secret Sharing

\[ s = s_1 + s_2 + s_3 \]

old share: \( S_i \)
Independence for Secrets

Proactive Secret Sharing

\[ s = s_1 + s_2 + s_3 \]

old share: \( S_i \)

split:

\[ =s_{i1}+s_{i2}+s_{i3} \ldots \]
Independence for Secrets

Proactive Secret Sharing

\[ s = s_1 + s_2 + s_3 \]

**split:**

\[ = s_{i1} + s_{i2} + s_{i3} \ldots \]

**reconstruct:**

\[ s_{1i} + s_{2i} + s_{3i} \ldots \]

= new share: \( s_i' \)
State Machine Approach

The basic recipe …

- **Servers:**
  - deterministic state machines
  - assumed to fail independently

- **Clients:**
  - make request
  - synthesize service response from individual server responses
Replica Independence

Eschewing Shared Design / Code

Solution: Diversity!

- Expensive or impossible to obtain:
  - Development costs
  - Interoperability risks

- Leverage what diversity exists.

- Mechanically create “artificial diversity”.
  ... Employ a program obfuscator.

Diagram:
- Server Code
- Random key: 0110101100...
- Obfuscator
- server replica
Semantics-preserving random program rewriting…

**Goals:** Attacker does not know:
- address of specific instruction subsequences.
- address or representation scheme for variables.
- name or service entry point for any system service.

**Options:**
- Obfuscate source (arglist, stack layout, …).
- Obfuscate object or binary (syscall meanings, basic block and variable positions, relative offsets, …).
- All of the above.
Replica Independence

Successful Attacks on Morphs

All morphs implement the same interface.

- **Interface attacks.** Obfuscation cannot blunt attacks that exploit the semantics of that (flawed) interface.
- **Implementation attacks.** Obfuscation can blunt attacks that exploit implementation details.

**Def.** implementation attack: An input for which all morphs (in some given set) don’t all produce the same output.
Replica Independence

Effectiveness of Obfuscation

**Ultimate Goal:** Determine the probability that a majority of morphs generate the same output for a set of attacks?

**Modest goal:** Understand how effective obfuscation is as compared with other defenses?

- Obvious candidate: Type checking
Type checking: Process to establish that all executions satisfy certain properties.

- Static: Checks made prior to exec.
  - Requires a decision procedure
- Dynamic: Checks made as exec proceeds.
  - Requires adding checks. Exec aborted if violated.

Probabilistic dynamic type checking: Some checks are skipped on a random basis.
Replica Independence

Obfuscation versus Type Checking

**Thesis:** Obfuscation and probabilistic dynamic type systems can “defend against” the same attacks.

From “thesis” → “theorem” requires fixing:
- a language
- a type system
- a set of attacks

**Progress:**
- Completed: Address space rewriting → pointer de-ref sanity.
- In progress: Renaming → strong typing.
- Open Problem: Characterize the probabilities.
Replica Independence
Implementing Proactive Obfuscation

Challenges:
- State recovery
- Protect Obfuscator
- Protect Egg-timer
- Tolerate server outage

Random key:
0110101100...

Server Code

Obfuscator

Egg timer

server replica
Putting it Together: CoPrOF

Cornell Proactive Obfuscation Firewall

Specification:
- Unlikely that attacker can gain control of the service.
- A steady stream of attacks might block service. (But service is restored once that stream is terminated.)

Server:
- Receives messages from “outside”.
- Manages state (encodes history of messages seen).
- Forward subset of messages to “inside”.
Theory → Practice

CoPrOF: Prototype

Inside Traffic

Server Code

Controller

Outside Traffic

N = 7 = 3(t+1) + 1 servers
Theory → Practice

CoPrOF: In the Flesh

Processors:
3 GHz Pentium 4
OpenBSD 4.0
Theory → Practice

CoPrOF: Service Bandwidth
Theory → Practice

CoPrOF: Ultimate Prototype

Inside Traffic

Server 0

Server 1

Server 6

Outside Traffic

N = 7 = 3(t+1) + 1 servers
Recapitulation

- Independence is no longer a matter of location.
  - Proactive obfuscation for code
    - Landscape of attacks vs defenses still unexplored.
  - Proactive secret sharing for keys
    - Add multi-party crypto to your toolbox.

- Assumptions matter. Weaker ones are better.
  - Assumptions = Vulnerabilities.
  - Timing assumptions $\rightarrow$ Denial of Service.
  - Use protocols for **asynchronous** (not synchronous) model.

- I + A but what about C?
  - Multiparty computation?
Concluding Remarks

Fault-tolerance *without* security risks irrelevance.

Fault-tolerance *with* security raises fundamental new questions. (And that’s good if you are a distributed systems researcher.)

Independence has become ripe for study.