

Byzantine Fault-Tolerant Confidentiality

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

1 Introduction

As the world becomes increasingly interconnected, more and more important services such as business transactions are deployed as *access-anywhere services* – services that are accessible by remote devices through the Internet and mobile networks. Such services often must access confidential data to provide service. For example, an online bank service must access a user’s checking account to process an online transfer request. In such a scenario, guarantees of availability, integrity, and confidentiality are essential. By availability, we mean that services must provide service 24/7 without interruption. By integrity, we mean that services must process clients’ requests correctly. By confidentiality, we mean that services must restrict who sees what data.

Unfortunately, access anywhere services may contain bugs that hackers can exploit to disrupt service or steal confidential data. Most software today contains bugs, and although the perfect solution would be to write bug-free software, this is currently so difficult that a more promising solution is to use redundancy to harden services against bugs. A traditional Byzantine fault-tolerant (BFT) system runs different implementations of the same service on several replicas and ensures that correct computation is performed by enough correct replicas to mask incorrect replicas [1, 2, 4, 13, 14, 16, 17]. Recent research has shown that BFT systems can be practical for several important services as they can be implemented with low overhead compared to the unreplicated services [3].

Although traditional BFT systems improve availability and integrity through redundancy, existing BFT architectures make such systems more prone to compromising confidentiality. In a traditional BFT system, replicas send all replies directly to clients. Thus, if a hacker manages to compromise one of the replicas, he can steal confidential data. Moreover, in traditional BFT systems, there is a fundamental tradeoff between increasing availability and integrity on one hand and strengthening confidentiality on the other. BFT systems use several replicas to provide availability, based on the reasoning that the replicas are different and it is therefore unlikely that they all fail simultaneously. However, because it is sufficient for an attacker to compromise a single replica, this approach also increases the chance that at least one replica contains an exploitable bug, allowing the attacker to gain access to the confidential data that the service uses.

This paper discusses how to use redundancy to simultaneously improve availability, integrity, and confidentiality. We call this problem Confidential BFT (CBFT) and propose the privacy firewall architecture to solve it. Service replicas connect to the privacy firewall and can send messages to the outside world only through it. The firewall runs a majority voting algorithm just like the clients of a traditional BFT system and filters out faulty messages that may contain confidential data.

This approach has several advantages, stemming from the fact that the privacy firewall is a separate component from the replicated service. The first advantage is simply the generality of the approach: since virtually all replicated services can be modeled as a replicated state machine, the privacy firewall can protect almost any replicated service that exists today. Second, once built, a privacy firewall can easily be used for a variety of replicated services, with only minor modification to the service (or none at all). Thus, CBFT’s privacy firewall can be adapted to legacy applications, providing them with confidentiality after the fact. The third advantage is that the effort spent into building the privacy firewall can be amortized over several replicated services: once the privacy firewall is built, it can easily be duplicated to protect additional services. This versatility makes it imaginable that companies would build privacy firewalls and then sell turnkey solutions.

The firewall system has to be correct to provide confidentiality. Even though the firewall is simple, building a formally verified bug-free firewall may not be feasible. However, redundancy can be used to improve the robustness of the firewall. Such a firewall system consists of a group of nodes that are interconnected such that any path from a service replica to the outside world is longer than a threshold, f . Thus, as long as there are f or fewer faulty firewall nodes, any communication from any service replica to the outside world must go through at least one correct node. Moreover, a correct node in a firewall chain can independently ensure that a unique sequence of replies results from a sequence of requests just as if this sequence of requests were processed by a single correct server. Thus, faulty machines are prevented from using steganography to leak confidential data.

In summary, this paper investigates how to build available, high-integrity, and confidential access-anywhere services by using redundancy and it outlines the architecture of one implementation, the Privacy Firewall. The key challenges

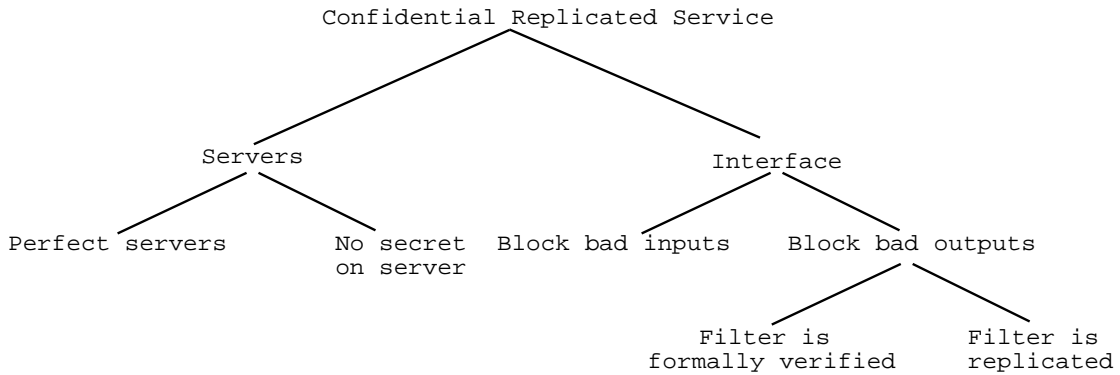


Figure 1: BFTC Design Space

ahead are to evaluate this new design and to develop a deeper understanding of the range of solutions to this problem.

2 Design space and related work

Redundancy techniques to improve availability and integrity without considering confidentiality have been extensively studied. Typically, in such systems all service replicas send replies to clients, who use the answer with the most votes. There is a significant body of work on BFT quorum systems [1, 13, 14, 15] in which a group of service replicas use intersection properties of quorum sets to guarantee that clients retrieve only those values of variables written previously. Byzantine fault tolerance for arbitrary services captured by state machines has been studied in both theoretical and practical settings [2, 3, 4, 16, 17]. In particular, Castro and Liskov [3, 4, 16] have shown that BFT systems can be practical because i) the throughput and latency of several BFT services implemented by them are comparable to unreplicated service [3] and ii) multiple implementations of the same legacy services by different vendors can be used to achieve failure independence of replicas [16].

In the previous paragraph we discuss how to provide only fault-tolerant integrity and availability. We now explore techniques that provide privacy as well.

When designing a system with privacy guarantees such as BFTC, one can either focus on the service itself or add a “filter” between the service and the client to remove the risk of privacy leak. These are the two main branches of the design space shown in Figure 1 that we will explore in more detail in this section.

To eliminate privacy risks in the service, one can use bug-free servers. To reach a high level of confidence, the whole system (hardware, system software, and application) must be formally verified [11].

Another service-centric approach, represented by the second leaf in the design space diagram (Figure 1), consists in making sure that no confidential data can be retrieved from the servers. This goal can be achieved in one of two ways. First, the data may be encrypted so as to be of no use to attackers. In this case traditional BFT designs can be used, but the servers are not able to process the data, limiting the generality of such services. Second, the confidential data can be split in such a way that even if a number of fragments are compromised no useful data can be recovered [5, 6, 7, 9, 10, 12, 18, 19]. This approach is only applicable to scenarios where we know how to process this split information, for example in certificate authorities [19] or in cases where the service is not expected to do any processing on the data, for example opaque data storage or transmission.

In general, the main limitation of server-based approaches is that they must be constructed on a service-by-service basis. When working with the interface, instead, one can build a single filter that will work for a wide range of services with little or no modification.

The classical approach to working on the interface is to put a firewall between the service and the clients. The mission of this firewall is to block all inputs that could cause the servers to fail. For example a corporate firewall might block all incoming traffic to all machines in the corporation except to public services (e.g. HTTP and email) on some machines. Firewalls can block a large class of invalid inputs, but they are not able to detect malformed requests to a service that exploit bugs in the service’s implementation. If these problems are known then one can write proxies to filter the requests, but this approach is again system-specific and must be done for every service to protect.

Instead of blocking inputs, the filter at the interface can be designed to filter bad outputs. This design takes advantage of the fact that the service itself is already redundant for fault tolerance: the filter sends the query to several servers and compares their answers. Different answers are a sign that

the query caused one of the servers to behave incorrectly, and the answers are therefore blocked to prevent any private information from leaking out. The discrepancy also allows detection of faulty behavior by servers.

The advantage of this design is that it can be applied to a wide range of services with little or no modification. We call it the “Privacy Firewall” by analogy to the ease of use of traditional firewalls. The key difference between the two is that traditional firewalls need to know bad inputs a priori. The privacy firewall uses discrepancies across the output of redundant servers to identify what it must filter.

This filter may of course also be the subject of attacks and must be fault-tolerant and maintain confidentiality. One approach is to verify the filter and its underlying operating system formally. This method is preferable to verifying the service itself because the filter can be used with several services with little or no modification. Furthermore, the filter is likely to be much simpler than the service and therefore easier to verify.

While this approach is promising, in this paper we explore another possibility: we use replication to make the filter fault-tolerant and maintain the confidentiality of the system.

3 A CBFT system

The traditional state machine approach for implementing Byzantine fault-tolerant services delegates the responsibility of combining the outputs of the ensemble of state machine replicas to the client. To quote Schneider [17], “*the voter - a part of the client - is faulty exactly when the client is, so the fact that an incorrect output is read by the client due to a faulty voter is irrelevant*” because a faulty voter is then synonymous with a faulty client. Although such a model is appropriate for services where availability and reliability are the goals, it fails to address confidentiality for access-anywhere services. In particular, if a hacker manages to compromise one replica in such a system, the system has no mechanisms to prevent the compromised replica from sending confidential data back to the hacker.

As shown in Figure 2(b), traditional approaches to tolerate Byzantine faults in replicas attempt to emulate the abstract notion of a single correct server as in Figure 2(a). Here R is a request sent by the client, P the response the client receives, and P' the set of responses filtered by the voting component V to eliminate incorrect information I received by V from a faulty replicas in the BFT. I can be used by a malicious replica as a channel to transmit confidential data. The obvious solution is to move the voter, V , away from clients into the service provider’s domain as shown in in Figure 2(c). Unfortunately, doing so is difficult since V is a potential point of vulnerability for availability, integrity, and

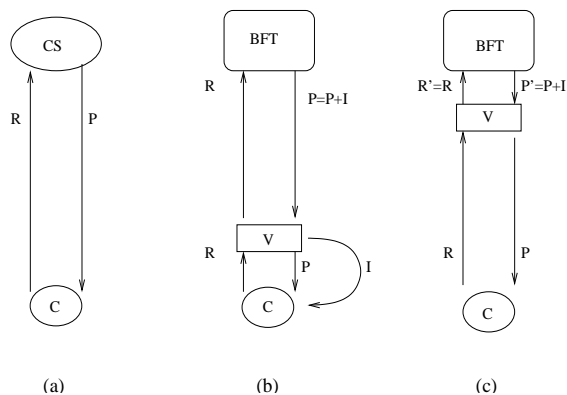


Figure 2: Fault-tolerance models

confidentiality. In particular, Schneider’s fault-sharing argument above no longer applies – faults in the voter V can hurt availability and integrity of correct clients. Furthermore, a fault in the voter can reintroduce a covert channel that allows the information I to escape to clients.

One solution is to appeal to software engineering and formal methods to construct a perfect voter. Another approach is to use redundancy to construct a highly reliable voter out of imperfect components. A threshold number of the components can suffer from Byzantine faults, but we group them to build a system that, as a whole, acts as a firewall and protects confidentiality without compromising availability. This is a hard problem, especially in the face of sophisticated attacks that could compromise confidential information by delaying certain replies, omitting requests, spoofing requests not generated by clients, etc.

To provide intuition, we first describe a simplified firewall that filters some incorrect information I , but is vulnerable to poor availability and in-band signaling that can leak confidential data. We then outline how to address these limitations. The system consists of two components as illustrated in Figure 3(a):

- a) A backend BFT consisting of a set of replicated servers U to tolerate s Byzantine faults.
- b) A firewall network V , organized as a chain of $f + 1$ machines, at most f of which are faulty. Voters can communicate directly only with their neighbors. Only f_c at the bottom of the chain can communicate directly with clients, and only the topmost machine f_s can communicate with the server replicas.

Clients send encrypted, signed requests to f_c . Each machine in the chain forwards the request up the chain towards f_s which broadcasts it to all servers in U . Servers in U are expected to respond with encrypted, signed replies to f_s . On receiving $s + 1$ identical replies with verifiable correct signatures for a given request, f_s forwards the reply and the

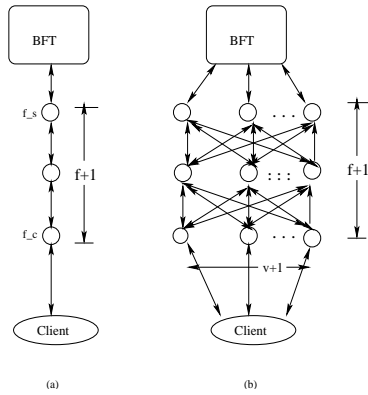


Figure 3: The privacy firewall

set of $s + 1$ signatures to the machine just below it in the chain. Each correct firewall thereafter independently verifies the $s + 1$ signatures and forwards them further down the chain, until finally f_c sends them to the client. If any firewall other than f_s detects that the replies do not match, it discards the reply and triggers a fault-detection alarm.

This design guarantees that the client only receives correct replies, because every response is verified by $f + 1$ machines in the firewall (at least one of which is correct) before reaching the client. In other words, replies will get through the firewall in the presence of up to f firewall and server failures if and only if they would have been returned by the service when no failures occur. This design guarantees that data considered confidential by the service will remain confidential in the presence of failures.

The system described above, though simple, has the following four drawbacks: (1) The system has no availability guarantee. If one of the machines in the firewall is compromised, it could drop requests or replies. (2) A faulty firewall machine might leak confidential information by altering the membership of the set of correct signatures vouching for a reply. (3) A faulty firewall machine might leak confidential information by inserting, omitting, or reordering requests or replies. (4) The system is susceptible to timing attacks, i.e., the malicious servers and/or the machines in the firewall could use the delay in sending responses to encode confidential information.

Availability of the system can be improved simply by augmenting the firewall with multiple interlinked chains each $f + 1$ in length as shown in Figure 3(b). The figure shows how the chains are linked together: this topology guarantees that faulty servers cannot transmit information by opting not to answer queries. Assuming at most f faulty machines, $f + 1$ such chains are sufficient to guarantee 100% availability. Since each chain only sends correct replies, the client can choose any reply that it receives. In general if v machines are susceptible to crash failures, it is

a straightforward graph-theoretical problem to prove that a minimum of $(f + 1)(v + 1)$ machines are necessary to protect confidentiality and ensure availability in the face of f Byzantine or v crash failures.

The second problem can be easily rectified by using threshold signatures [8] with a threshold of $s + 1$. Each of the replicas sends its reply along with its partial signature to the top firewall f_s . f_s waits for $s + 1$ matching replies, combines them to form one message and the matching combined signature, and forwards them down the chain. Each machine now independently verifies the reply before sending it further down.

One solution to (3) is for each firewall machine to ensure that it transmits replies in exactly the order that it receives requests. If there exists at least one correct firewall CF on a data path, then clients or firewall machines nearer to clients than CF can gain no confidential information from request ordering, insertion, or omission.

Timing attacks are difficult to tackle in an airtight manner without affecting the performance of the system. Timing attacks can be countered in two ways: i) we can reduce the ability of faulty nodes to affect latency, for example by having correct nodes insert delays deliberately, ii) we can examine unusual latency fluctuations at some machines to detect possible intrusion and recover proactively.

With the above additions, it can be shown that the privacy firewall emulates the single correct server abstraction as depicted in Figure 2. We show the equivalence in a single chain firewall by considering a sequence of requests seen at the first correct machine from the bottom in a chain and by showing that it produces the unique sequence of replies that the abstract single correct server would generate. By restricting the flexibility of the information sent out by a correct machine in the firewall we remove the means for malicious machines to hide confidential information therein, thereby preventing steganography. It is straightforward to extend the argument to multiple chains.

4 Conclusion

In this abstract we argue that availability, integrity, and confidentiality are essential for access-anywhere services and that traditional replicated Byzantine fault tolerant systems, although they strengthen integrity and availability, weaken confidentiality. A Confidential BFT system (CBFT) differs from traditional BFT systems in that it increases confidentiality guarantees instead of reducing them, while maintaining the availability and integrity guarantees provided through redundancy. The approach we propose in this paper is general because it applies to any state machine. We are currently

implementing a prototype privacy firewall and evaluating its performance.

References

- [1] R. Bazzi. Synchronous Byzantine quorum systems. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing (PODC '97)*, August 1997.
- [2] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. In *Journal of the Association for Computing Machinery*, October 1995.
- [3] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, February 1999.
- [4] M. Castro and B. Liskov. Proactive recovery in a Byzantine-fault-tolerant system. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, October 2000.
- [5] Y. Deswarte, L. Blain, and J. Fabre. Intrusion tolerance in distributed computing systems. In *Proc. Symp. on Research in Security and Privacy*, pages 110–121, Oakland, CA, USA, 1991. IEEE Computer Society Press.
- [6] M. Franklin and R. N. Wright. Secure communication in minimal connectivity models. In *Advances in Cryptology – EUROCRYPT '98*, pages 346–360, 1998.
- [7] M. Franklin and M. Yung. Secure hypergraphs: Privacy from partial broadcast (extended abstract). In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 36–44, 29 May–1 June 1995.
- [8] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Theory and Application of Cryptographic Techniques*, number Theory, pages 354–371, 1996.
- [9] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk. Robust and efficient sharing of RSA functions. *Journal of Cryptology*, 13(2):273–300, Spring 2000.
- [10] B. Randell J-C. Fabre. An object-oriented view of fragmented data processing for fault and intrusion tolerance in distributed systems. In *Second European Symposium on Research in Computer Security (ESORICS 92)*, pages 193–208, 1992.
- [11] M. Kaufmann, P. Manolios, and J. Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, June 2000.
- [12] J. Kubiawicz, D. Bindel, P. Eaton, Y. Chen, D. Geels, R. Gummadi, S. Rhea, W. Weimer, C. Wells, H. Weatherpoon, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. *ACM SIGPLAN Notices*, 35(11):190–201, November 2000.
- [13] D. Malkhi and M. Reiter. Byzantine quorum systems. In *Distributed Computing*, 1998.
- [14] D. Malkhi, M. Reiter, and A. Wool. The load and availability of Byzantine quorum systems. In *SIAM Journal on Computing*, December 2000.
- [15] J-P. Martin, L. Alvisi, and M. Dahlin. Small Byzantine quorum systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 374–383, June 2002.
- [16] R. Rodrigues, M. Castro, and B. Liskov. BASE: Using abstraction to improve fault tolerance. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001 October.
- [17] F. Schneider. Implementing Fault-tolerant Services Using the State Machine Approach: A tutorial. *Computing Surveys*, 22(3):299–319, September 1990.
- [18] T. Wu, M. Malkin, and D. Boneh. Building intrusion-tolerant applications. In *Proceedings of the 8th USENIX Security Symposium (SECURITY-99)*, pages 79–92, Berkely, CA, August 23–26 1999. Usenix Association.
- [19] L. Zhou, F. Schneider, and R. Renesse. COCA: A secure distributed on-line certification authority. In *Technical Report TR2000-1828, Computer Science Department, Cornell University*, 2000.