Byzantine Fault-Tolerant Confidentiality

Jian Yin Arun Venkataramani Jean-Phillipe 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, software for providing access anywhere services may contain bugs, and hackers may exploit these bugs to disrupt service or steal confidential data. One solution is to eliminate bugs by using better software engineering practice or formal methods [12]. However, the complexity of real world systems and application software makes it difficult to produce bug-free implementations with existing techniques.

A promising approach 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, 14, 15, 16, 17]. Recent research has shown that BFT systems can be practical for several important services as they can be implemented with low overheads compared to the unreplicated services [3].

However, 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 discuss how to use redundancy to simultaneously improve availability, integrity, and confidentiality. We propose an architecture for such a system - Confidential BFT (CBFT). In CBFT, service replicas connect to a "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 modelled as a replicated state machine, the privacy firewall can protect almost any replicated service that exists today. Second, once built, a privacy firewall can very 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, any communication from any service replica to the outside world must go through at lease one correct node as long as there are fewer than $f + 1$ faulty firewall nodes. 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 outlines the architecture of one implementation of such an architecture, CBFT.

2 Related Work

There is much previous work on how to use redundancy to tolerate Byzantine faults. This work falls into three categories: i) using client voting to improve availability and integrity without considering confidentiality, ii) using secret sharing to improve confidentiality for a limited class of services that do not manipulate confidential data, and iii) making a trade-off and giving the servers enough data to enable some level of processing with the understanding that they do not have enough data for a leak to be damaging.

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, and voting takes place at clients. There is a significant body of work on BFT quorum systems [1, 14, 15] in which a group of service replicas use intersection properties of quorum sets to guarantee that clients retrieve values of variables written previously. Byzantine fault tolerance for arbitrary services captured by state machines have 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]

There are also several studies [6, 7, 8, 10, 13, 19, 20] on sharing secrets among several service replicas that prevent fewer than some threshold number of service replicas from compromising secrecy. However, hiding the unencrypted value of confidential data from servers effectively prevents servers from manipulating confidential data. Thus, such techniques can only be applied to a limited class of services in which the servers only retrieve, use, or transmit opaque data such as storage systems, file systems, distributed certification authority, and message transmissions.

The third approach consists in distributing enough data to the servers to allow some processing, but not enough to make leaks damaging. This approach is known as Fragmented Data Processing [11]. This approach is practical but the level of granularity of the fragmentation must be chosen carefully for each implementation.

Other related work [5] relies on detection to react to failures. This is different from our approach where we do not use any oracle to detect faulty machines; however we can of course detect faulty machines once their deviate from correct behavior in their communication with other machines.

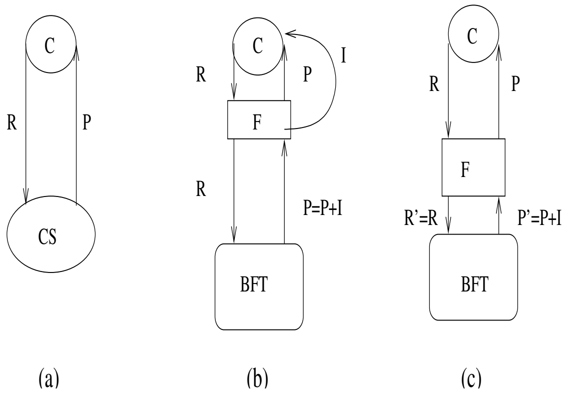

Figure 1: Fault-tolerance models

Our work is similar in spirit to the Ballot Monitors in SITAR [18]. The ballot monitors are designed to perform a majority voting between several replicas in order to identify faulty servers and return a correct reply. Our contribution is to show how such a system can be built since SITAR publications do not discuss any implementation details at the time of this writing. More importantly, privacy firewall are explicitely designed to handle confidentiality. This constraint is not part of the design of SITAR, which instead concentrates on availability. We argue that focusing on combining confidentiality with availability, as we do, is of more value.

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."* Though such a model is appropriate for services deployed locally, 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 replicas from sending confidential data back to the hacker.

As shown in Figure 1(b), traditional approaches to tolerate Byzantine faults in replicas attempt to emulate the abstract notion of a single correct server as in (a). Here R is a request sent by the client and P the response the client receives, and P' the set of responses filtered by the voting component F to eliminate incorrect information I received by F 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, F, away from clients into the service providers' domain as shown in in (c). Unfortunately, this is challenging since F is a potential point of vulnerability for availability, integrity, and confidentiality. In particular, Schneider's "fault sharing" argument above no longer applies – faults in the voter F 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.

To understand the interplay of redundancy and confidentiality, our goal is to design a voting component that acts as a "firewall" to filter responses. This component is susceptible to Byzantine faults, but we group them to build a system that 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.

In order to tolerate faults in the voter, the voter itself must be replicated, otherwise a single failure – of the voter – can prevent the system from producing the correct output or protecting confidentiality. This additional level of replication threatens to mire one recursively in the original problem. In our firewall system, we solve this problem by i) ensuring that every response from the server replica set is verified by at least one correct voter and ii) ensuring that each voter can independently verify the correctness of any reply.

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 2(a):

a) A backend BFT consisting of set of replicated servers U to tolerate s Byzantine faults.
b) A firewall network F, 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 the topmost machine f_s alone 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 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

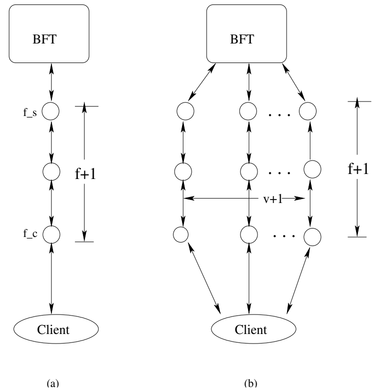

Figure 2: Confidential BFT

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.

P1: Since every response is verified by $f + 1$ machines in the firewall (at least one of which is correct) before reaching the client, the client only receives correct replies.

The system described above, though simple has the following drawbacks:
1. The system has no availability guarantee. Even 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 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 independent chains each $f + 1$ in length as shown in Fig. 2(b). Assuming at most f faulty machines, at most $f + 1$ such chains are sufficient to guarantee 100% availability. Since P1 still holds, 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.

The second problem can be easily rectified by using threshold signatures [9] with a threshold of $s + 1$. Each of the replicas sends its reply along with its partial signature to f_s.

f_s waits for $s + 1$ matching replies, combines them to form one just one message and one 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 a 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 CBFT system emulates the single correct server abstraction as depicted in Fig. 1. 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 The CBFT Protocol

The protocol is presented in the figure below. The protocol starts at the client (Figure 3), where the query is simply forwarded to all the bottom firewall machines. The firewalls (Figure 4) take note of the order of that request (by storing an ordered list of the hash of the requests) and forward the request up the chain until the top privacy firewall machine. This machine (Figure 5) then calls Castro and Liskov's BFT protocol [3] with the query. That protocol implements a replicated state machine.

The top firewall (instead of the client) then receives all the answers and performs the filtering, as per BFT's specification. Note that at this point any divergent answer indicates a faulty server. The protocol for the bottom firewalls is identical to that of the middle firewalls, with the difference that they send their answers back to the requesting client.

$reply$=**Execute**($command$)

1. let n be a nunce.
2. send (QUERY,$command$,n) to all bottom firewalls.
3. **wait until** receive an answer $\{reply, h, n\}_S$, signed by the service's key S, where h is a hash of the query.
4. return $reply$.

Figure 3: *CBFT protocol, executed at the clients*

Query($command, n$) (from below)

1. add Hash($command, n$) to the end of the query queue
2. forward (QUERY,$command$,n) to the firewall machine above us

Reply($\{reply, h, n\}_S$) (from above)

1. if the signature S does not match the service key, drop that message.
2. add the message to the $pending$ list.
3. While h of some message R in the $pending$ list matches the value stored at the head of the query queue then
4. forward ($REPLY, \{reply, h, n\}_S$) to the firewall underneath us.
5. remove the head of the query queue.

Figure 4: *CBFT protocol, executed by middle firewalls*

Query($command, n$) (from below) // requests are executed in sequence

1. execute the BFT protocol with $command, n$.
2. **wait until** receive an answer ($reply, n, pS$) from q servers such that combining the partial signatures pS yields a message $\{reply, n\}_S$ with a valid signature from the service key.
3. send (REPLY, $\{reply, n\}_S$) to the firewall machine underneath us.

Figure 5: *CBFT protocol, executed by the top firewalls*

5 Conclusion

In this abstract we argue that availability, integrity, and confidentiality are essential for access anywhere services and that traditional assumptions behind Byzantine Fault Tolerant state machine replication do not protect confidentiality when clients are not trusted. 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 by the redundancy. The approach we propose in this paper is very general because it applies to any state machine. We are currently in the process of implementing a prototype of CBFT to evaluate 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] M. Cukier, J. Lyons, P. Pandey, H. V. Ramasamy, W. H. Sanders, P. Pal, F. Webber, R. Schantz, J. Loyall, R. Watro, M. Atighetchi, and J. Gossett. Intrusion tolerance approaches in ITUA. In *The 2001 International Conference on Dependable Systems and Networks, Goeteborg, Sweden*, pages B–64 to B–65, July 2001.

[6] 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.

[7] M. Franklin and R. N. Wright. Secure communication in minimal connectivity models. In *Advances in Cryptology – EUROCRYPT '98*, pages 346–360, 1998.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] M. Kaufmann, P. Manolios, and J. Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, June 2000.

[13] J. Kubiatowicz, D. Bindel, P. Eaton, Y. Chen, D. Geels, R. Gummadi, S. Rhea, W. Weimer, C. Wells, H. Weatherspoon, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. *ACM SIGPLAN Notices*, 35(11):190–201, November 2000.

[14] D. Malkhi and M. Reiter. Byzantine quorum systems. In *Distributed Computing*, 1998.

[15] D. Malkhi, M. Reiter, and A. Wool. The load and availability of Byzantine quorum systems. In *SIAM Journal on Computing*, December 2000.

[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] R. Wang, F. Wang, and G. T. Byrd. SITAR: A scalable intrusion tolerance architecture for distributed server. In *IEEE 2nd SMC Information Assurance Workshop, West Point, New York*, 2001.

[19] 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.

[20] L. Zhou, F. Schneider, and R. Renesse. COCA: A secure distributed on-line certification authority. In *Technical Report TR2000-1828, Computer Science Department, Cornell Univesity*, 2000.