A shorter version of this paper appears in *Proc. Intl. Conference on Dependable Systems and Networks (DSN-2001)*, Gothenborg, Sweden, IEEE, 2001.

Distributing Trust on the Internet

Christian Cachin

IBM Research
Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
cca@zurich.ibm.com

March 8, 2001

Abstract

This paper describes an architecture for secure and fault-tolerant service replication in an asynchronous network such as the Internet, where a malicious adversary may corrupt some servers and control the network. It relies on recent protocols for randomized Byzantine agreement and for atomic broadcast, which exploit concepts from threshold cryptography. The model and its assumptions are discussed in detail and compared to related work from the last decade in the first part of this work, and an overview of the broadcast protocols in the architecture is provided. The standard approach in fault-tolerant distributed systems is to assume that at most a certain fraction of servers fails. In the second part, novel general failure patterns and corresponding protocols are introduced. They allow for realistic modeling of real-world trust assumptions, beyond (weighted) threshold models. Finally, the application of our architecture to trusted services is discussed.

1 Introduction

Distributed systems running in error-prone and adversarial environments must rely on trusted components. In today's Internet these are typically directory and authorization services, such as the domain name system (DNS), Kerberos, certification authorities, or LDAP-based secure directories. Building such centralized trusted services has turned out to be a valuable design principle for computer security because the trust in them can be leveraged to many, diverse applications that all benefit from centralized management. Often, a trusted service is implemented as the only task of an isolated and physically protected machine.

Unfortunately, centralization introduces a single point of failure. Even worse, it is increasingly difficult to protect any single system against the sort of attacks proliferating on the Internet today. One established way for enhancing the fault tolerance of centralized components is to distribute them among a set of servers and to use replication algorithms for masking faulty servers. Thus, no single server has to be trusted completely and the overall system derives its integrity from a majority of correct servers.

In this paper, we describe an architecture for distributing trusted services among a set of servers that guarantees liveness and safety (or equivalently, availability and integrity) of the services despite some servers being under control of an attacker or failing in arbitrary malicious ways. Our system model is characterized by a static set of servers, completely asynchronous point-to-point communication, and the use of modern cryptographic techniques. Trusted applications are implemented by deterministic state machines replicated on all servers and initialized to the same state. Client requests are delivered by an atomic broadcast protocol that imposes a total order on all requests and guarantees that the servers perform the same sequence of operations; such an atomic broadcast can be built from a randomized protocol to solve Byzantine agreement. We use efficient and provably secure agreement and broadcast protocols that have recently been developed.

In the first part of the paper (Section 2), we provide a detailed discussion of these assumptions, compare them to related efforts from the last decade, and argue why we believe that these choices are adequate for trusted applications in an Internet environment.

In Section 3, a brief overview of our architecture is given. Our main tool is a recent protocol for atomic broadcast, which builds on reliable broadcast and multi-valued Byzantine agreement.

Of course, distributing a central service to a set of servers enhances its fault tolerance only if there is enough diversity in that set such that common failure modes can be ruled out. For example, if the same simple attack succeeds for all servers, not much has been gained by distribution. It is thus crucial for this approach to make sense that the servers vary in their configuration, operating system, physical location, load etc. Placing them in different administrative domains also eliminates corruptible system administrators as one path of attack.

Nevertheless, no particular distinction is made by the protocols themselves, and the traditional assumption of this approach to fault-tolerant distributed systems is that at most a certain fraction of homogeneous nodes fails. We generalize this model in the second part of this paper. Based on recent progress in distributed systems and cryptography, we introduce in Section 4 systems that tolerate a family of novel failure patterns, which allow for more realistic modeling of real-world trust assumptions. For example, they allow a distributed system running at multiple sites to continue operating safely even if all hosts at one site are unavailable or corrupted, no matter how many there are. One may construct a distributed system that maintains its safety despite the corruption of a majority of its servers in this way. We discuss two concrete instantiations of this general principle by giving the corresponding linear secret sharing schemes.

Finally, we sketch some applications of our architecture in more detail in Section 5: A certification authority and directory service, which is needed by all public-key infrastructures (PKIs), and a notary and time-stamping service that acts as a secure document registry with a logical clock.

2 Model

Our system consists of a static set of n servers, of which up to t may fail in completely arbitrary ways, and an unknown number of possibly faulty clients. All parties are linked by asynchronous point-to-point communication channels. Without loss of generality we assume that all faulty parties are controlled by a single adversary, who also controls the communication links and the internal clocks of all servers.

Faulty parties are called *corrupted*, the remaining ones are called *honest*. Thus, any statement about the common state of the system can rely only on the honest parties, and they proceed only to the extent that the adversary delivers messages faithfully. In short, *the network is the adversary.*

Furthermore, there is a trusted dealer that generates and distributes secret values to all servers once and for all, when the system is initialized. The system can process a practically unlimited number of requests afterwards. Sometimes, it is possible to bootstrap security from a PKI, e.g., to establish secure point-to-point channels. Since we use specialized keys for which no distributed key generation protocols are currently known, and since our goal is to protect the heart of the PKI itself, an external mechanism is needed.

This model falls under the impossibility result of Fischer, Lynch, and Paterson [16] of reaching consensus by deterministic protocols. Many developers of practical systems seem to have avoided this model in the past for that reason and have built systems that are weaker than consensus and Byzantine agreement. However, Byzantine agreement *can be solved by randomization* in an expected constant number of rounds only [10, 8]. Although the first randomized agreement protocols were more of theoretical interest, their practical relevance has been recognized by now. For example, Guerraoui et al. [20] argue that solving consensus is central for building asynchronous distributed systems tolerating crash failures; we pursue the same approach in the model with Byzantine faults.

The recent Byzantine agreement protocol of Cachin, Kursawe, and Shoup [8] is based on modern, efficient cryptographic techniques with provable security, withstands the maximal possible corruption, and is also quite practical given current processor speed. (Its security proof uses the random oracle model, see below.)

In our architecture we use Byzantine agreement as a primitive for implementing atomic broadcast, which in turn guarantees a total ordering of all delivered messages. Note that atomic broadcast is equivalent to Byzantine agreement in our system model [12] and thus considerably more expensive than reliable broadcast, which only provides agreement of the delivered messages, but no ordering (Section 3).

Below we elaborate on the three key features of our model (cryptography, asynchronous communication, static server set) and then compare it to related efforts.

2.1 Cryptography

Cryptographic techniques such as public-key encryption schemes and digital signatures are crucial already for many existing secure services. For distributing trusted services, we also need distributed variants of them from *threshold cryptography.*

Threshold cryptographic schemes are non-trivial extensions of the classical concept of secret sharing in cryptography. Secret sharing allows a group of n parties to share a secret such that t or fewer of them have no information about it, but $t+1$ or more can uniquely reconstruct it. However, one cannot simply share the secret key of a cryptosystem and reconstruct it for decrypting a message because as soon as a single corrupted party knows the key, the cryptosystem becomes completely insecure and unusable.

A *threshold public-key cryptosystem* looks similar to an ordinary public-key cryptosystem with distributed decryption. There is a single public key for encryption, but each party holds a *key share* for decryption (all keys were generated by a trusted dealer). A party may process a decryption request for a particular ciphertext and output a decryption share together with a proof of its validity. Given a ciphertext resulting from encrypting some message and more than t valid decryption shares for that ciphertext, it is easy to recover the message; this property is called *robustness.* The scheme must also be secure against *adaptive chosen-ciphertext attacks* in order to be useful for all conceivable applications. The formal security definition can be found in the literature [36]; essentially, it ensures that the adversary cannot obtain any meaningful information from a ciphertext unless she has obtained a corresponding decryption share from at least one honest party.

In a *threshold signature scheme*, each party holds a *share* of the secret signing key and may generate shares of signatures on individual messages upon request. The validity of a signature share can be verified for each party. From $t+1$ valid signature shares, one can generate a digital signature on the message that can later be verified using the single, publicly known signature verification key. In a secure threshold signature scheme, it is infeasible for a computationally bounded adversary to produce $t+1$ valid signature shares that cannot be combined to a valid signature and to output a valid signature on a message for which *no* honest party generated a signature share.

Another important cryptographic algorithm is the *threshold coin-tossing scheme* in the randomized Byzantine agreement protocol of Cachin, Kursawe, and Shoup [8] that provides arbitrarily many unpredictable random bits.

Threshold-cryptographic protocols have been used for secure service replication before, e.g., by Reiter and Birman [33]. However, a major complication for adopting threshold cryptography to our asynchronous distributed system is that many early protocols are not robust and that most protocols rely heavily on synchronous broadcast channels. Only very recently, non-interactive schemes have been developed that satisfy the appropriate notions of security, such as the threshold cryptosystem of Shoup and Gennaro [36] and the threshold signature scheme of Shoup [35]. Both have non-interactive variants that integrate well into our asynchronous model. However, they can be proved secure only in the so-called *random oracle model* that makes an idealizing assumption about cryptographic hash functions [2]. This falls short from a proof in the real world but gives very strong heuristic evidence for their security; there are many practical cryptographic algorithms with proofs only in this model.

2.2 No Timing Assumptions

We do not make any timing assumptions in the design of our protocols and work in a completely asynchronous model. Asynchronous protocols are attractive because in a synchronous system, one would have to specify timeout values, which is very difficult when protecting against arbitrary failures that may be caused by a malicious attacker.

It is usually much easier for an intruder to block communication with a server than to subvert it. Prudent security engineering also gives the adversary full access to all specifications, including timeouts, and excludes only cryptographic keys from her view. Such an adversary may simply delay the communication with a server longer than the timeout and the server appears faulty to the others.

Time-based failure detectors [12] can easily be fooled into making an unlimited number of wrong failure suspicions about honest parties like this. The problem arises because one crucial assumption underlying the failure detector approach, namely that the communication system is stable for some longer periods when the failure detector is accurate, does not hold against a malicious adversary. A clever adversary may subvert a server and make it appear working properly until the moment at which it deviates from the protocol—but then it may be too late. Heuristic predictions about the future behavior of a server are pointless in security engineering.

Of course, an asynchronous model cannot guarantee a bound on the overall response time of an application. But the asynchronous model can be seen as an elegant way to abstract from time-dependent peculiarities of an environment for proving an algorithm correct such that it satisfies liveness and safety under *all* timing conditions. By making no assumption about time at all, the coverage of the timing assumption appears much bigger, i.e., it has the potential to be justified in a wider range of real-world environments. For our applications, which focus on the security of trusted services, the resulting lack of timeliness seems tolerable.

A variation of the asynchronous model is to assume probabilistic behavior of the communication links [5, 27], where the probability that a link is broken permanently decreases over time. But since this involves a timing assumption, it is essentially a probabilistic synchronous model (perhaps it should also bear that name) and suffers from all the problems mentioned before. The model investigated by Moser and Melliar-Smith [27] assumes, additionally, a fairness property and a partial order imposed by the underlying communication system, but such assumptions seem also difficult to justify on the Internet.

A promising alternative to time-free models is to rely on a minimal, trusted time service provided by a specialized subsystem, as proposed by Veríssimo, Casimiro, and Fetzer [38]. But apart from the open question of how to implement such a "timely computing base" on the Internet, this approach is so recent that its implications for secure state machine replication in a Byzantine environment are not yet fully understood.

2.3 Static Server Set

Distributing a trusted service among a static set of servers leverages the trust in the availability and integrity of each individual server to the whole system. This set is to remain fixed during the whole lifetime of the system, despite observable corruptions. The reason is that all existing threshold-cryptographic protocols are based on fixed parameters (e.g., n and t) that must be known when the key shares are generated.

A corrupted server cannot be resurrected easily because the intruder may have seen all its cryptographic secrets. Unless specialized "proactive" protocols [9] are used to refresh all key shares periodically, the only way to clean up a server is to redistribute fresh keys. However, dynamic groups and proactively secure cryptosystems in asynchronous networks are an open area of research (see Section 6).

The alternative is to remove apparently faulty servers from the system. This is the paradigm of view-based group communication systems in the crash-failure model (see the survey in [30]). They offer resilience against crash failures by eliminating non-responding servers from the current view and proceeding without them to the next view. Resurrected servers may join again in later views.

The Rampart toolkit [32] is the only group communication system that uses views and tolerates arbitrary failures. But since it builds on a membership protocol to agree dynamically on the group's composition, it easily falls prey to an attacker that is able to delay honest servers just long enough until corrupted servers hold the majority in the group. Because the maintenance of security and integrity is the primary application of our protocols for trusted services, we cannot tolerate such attacks and use a static group instead (but again, see Section 6).

2.4 Related Work

The use of cryptographic methods for maintaining consistent state in a distributed system has a long history and originates with the seminal work of Pease, Shostak, and Lamport [28].

One of the first attempts to build secure and robust replicated services was *DELTA-4*, an EU-funded research project [14]. DELTA-4 developed a general architecture for dependable distributed systems. It provides distributed, intrusion-tolerant services for data storage, authentication and authorization. Secrecy is supported via client-side encryption and data fragmentation, and availability via data replication for the fragments. Secret sharing is supported, but no computations on shared secrets or robust protocols are implemented. DELTA-4 assumes a synchronous communication network, and for the security services a static, threshold-based adversary structure.

The pioneering work of Reiter and Birman [33] (abbreviated RB94 henceforth) introduces secure state machine replication in a Byzantine environment and a broadcast protocol based on threshold cryptography that maintains causality among the requests. Similar to our architecture, it uses a static set of servers, who share the keys of a threshold signature scheme and a threshold cryptosystem. Thus, clients need only know the single public keys of the service, but not those of individual servers.

In order to obtain a fully robust system for an asynchronous model with malicious faults, however, RB94 must be complemented with robust threshold cryptography and secure atomic broadcast protocols, which were not known at that time. Our work builds on this and attempts to close this gap.

Subsequent work by Reiter on *Rampart* [32] shares our focus on distributing trusted services, but assumes a different model as explained in the previous sections: it implements atomic broadcast on top of a group membership protocol that dynamically removes apparently faulty servers from the set.

The broadcast protocols of Malkhi, Merritt, and Rodeh [24] work again with a static group in a model similar to ours, but implement only reliable broadcast and do not guarantee a total order, as needed for maintaining consistent state.

The *e-Vault* prototype for secure distributed storage [18] addresses a subset of the applications considered here, namely storing and retrieving immutable data. It works in a synchronous environment, though, and is not directly applicable to wide-area networks.

Castro and Liskov [11] (called CL99 below) present an interesting practical algorithm for distributed service replication that is very fast if no failures occur. It requires no explicit timeout values, but assumes that message transmission delays do not grow faster than some predetermined function for an indefinite duration. Since the CL99 protocol is deterministic, it can be blocked by a Byzantine adversary (i.e., violating liveness), but it will maintain safety under all circumstances. In contrast, our approach satisfies both conditions because it is based on probabilistic agreement.

The *Fleet* architecture of Malkhi and Reiter [26] supports loose coordination in large-scale distributed systems and shares some properties of our model. It works in a Byzantine environment and uses quorum systems and threshold cryptography for implementing a randomized agreement protocol (in the form of "consensus objects"). However, the servers do not directly communicate with each other for maintaining distributed state and merely help clients carrying out fault-tolerant protocols. Close coordination of all servers is also not a primary goal of Fleet. Implementing distributed state machine replication on top of Fleet is possible, in principle, but needs additional steps.

The *Total* family of algorithms for total ordering by Moser and Melliar-Smith [27] implements atomic broadcast in a Byzantine environment, but only assuming a benign network scheduler with some specific probabilistic fairness guarantees. Although this may be realistic in highly connected environments with separate physical connections between all machines, it seems not appropriate for arbitrary Internet settings.

SecureRing [22] and the very recent work of Doudou, Garbinato, and Guerraoui [15] (abbreviated as DGG00) are two examples of atomic broadcast protocols that rely on failure detectors in the Byzantine model. They encapsulate all time-dependent aspects and obvious misbehavior of a party in the abstract notion of a failure detector and permit clean, deterministic protocols (see also [1]). However, most implementations of failure detectors will use timeouts and actually suffer from some of the problems mentioned above. It also seems that Byzantine failure detectors are not yet well enough understood to allow for precise definitions.

A comparison of systems for secure state machine replication is shown in Figure 1. The

Reference	Timing	Servers	BA?	Remark
RB94 [33]	async.	static	yes$^{(1)}$	crash-failures only
Rampart [32]	async.	dynamic	no	FD for liveness and safety
Total alg. [27]	prob. async.	static	no	needs causal order on links
CL99 [11]	async.	static	no	FD for liveness
Fleet [26]	async.	static	yes$^{(2)}$	no state machine replication
SecureRing [22]	async.	static	yes$^{(3)}$	"Byzantine" FD
DGG00 [15]	async.	static	yes$^{(3)}$	"Byzantine" FD
this paper	async.	static	yes$^{(4)}$	general adversaries (Q^3)

Figure 1: Systems for secure state machine replication (Fleet supports only loose coordination). All systems achieve optimal resilience $t < n/3$. The column entitled "BA?" notes if a system solves Byzantine agreement (BA); those who do build (1) on an (assumed) atomic broadcast protocol, (2) on randomization and threshold signatures, (3) on a failure detector or "muteness detector" in the Byzantine model, or (4) on a cryptographic coin [8] in the underlying Byzantine agreement protocol. Some systems need a failure detector (FD); all except Total need a trusted dealer for setup. Fleet can also tolerate adversaries of Byzantine Quorum systems, our system tolerates general adversaries (Q^3-adversaries, see Section 4).

cryptographic model with randomized Byzantine agreement seems both practically and theoretically attractive, although it appears to have been somewhat overlooked in the past. (The fact that randomized agreement protocols have non-terminating runs does not matter because their probability is negligible; moreover, if a protocol involves *any* cryptography, and the practical protocols mentioned above do so, a negligible probability of failure remains anyway.) Remarkably, during the two decades since the question of maintaining "interactive consistency" was first formulated [28], no secure system in our asynchronous model has been designed until now.

3 Secure Asynchronous Broadcast Protocols

This section presents a short overview of the broadcast protocols used in our architecture. Detailed descriptions can be found in companion papers [8, 7].

We need protocols for basic broadcasts (reliable and consistent broadcast), atomic broadcast, and secure causal atomic broadcast; they can be described and implemented in a modular way as follows, using multi-valued Byzantine agreement and randomized binary Byzantine agreement as primitives.

<table>
<tr><td colspan="2">Secure Causal Atomic Broadcast</td></tr>
<tr><td colspan="2">Atomic Broadcast</td></tr>
<tr><td colspan="2">Multi-valued Byzantine Agreement</td></tr>
<tr><td>Broadcast Primitives</td><td>Byzantine Agreement</td></tr>
</table>

All our broadcast and agreement protocols work under the optimal assumption that $n > 3t$.

Byzantine agreement requires all parties to agree on a binary value that was proposed by an honest party. The protocol of Cachin et al. [8] follows the basic structure of all randomized solutions (e.g., [3]) and terminates within an expected constant number of asynchronous rounds. It achieves the optimal resilience $n > 3t$ by using a robust threshold coin-tossing protocol, whose security is based on the so-called Diffie-Hellman problem. It requires a trusted dealer for setup, but can process an arbitrary number of independent agreements afterwards. Threshold signatures are further employed to decrease all messages to a constant size. As mentioned before, its security proof relies on the random oracle model.

Another primitive is *multi-valued Byzantine agreement*, which provides agreement on values from larger domains. Multi-valued agreement requires a non-trivial extension of binary agreement. The difficulty in multi-valued Byzantine agreement is how to ensure the "validity" of the resulting value, which may come from a domain that has no a priori fixed size. Our approach to this is a new, "external" validity condition, using a global predicate with which every honest party can determine the validity of a proposed value. The protocol guarantees that the system may only decide for a value acceptable to honest parties. This rules out agreement protocols that decide on a value that no party proposed. Our implementation of multi-valued Byzantine agreement uses only a constant expected number of rounds.

A basic broadcast protocol in a distributed system with failures is *reliable broadcast*, which provides a way for a party to send a message to all other parties. Its specification requires that all honest parties deliver the same set of messages and that this set includes all messages broadcast by honest parties. However, it makes no assumptions if the sender of a message is corrupted and does not guarantee anything about the order in which messages are delivered. The reliable broadcast protocol of our architecture is an optimized variant of the elegant protocol by Bracha and Toueg [5]. We also use a variation of it, called *consistent broadcast*, which is advantageous in certain situations. It guarantees uniqueness of the delivered message (thus the name consistent broadcast), but relaxes the requirement that all honest parties actually deliver the message—a party may still learn about the existence of the message by other means and ask for it. A similar protocol was used by Reiter [31].

An *atomic broadcast* guarantees a total order on messages such that honest parties deliver all messages in the same order. Any implementation of it must implicitly reach agreement whether or not to deliver a message sent by a corrupted party and, intuitively, this is where Byzantine agreement is needed. The basic structure of the atomic broadcast protocol follows the atomic broadcast protocol of Chandra and Toueg [12] for the crash-failure model: the parties proceed in global rounds and agree on a set of messages to deliver at the end of each round, using multi-valued Byzantine agreement as follows.

First, every party digitally signs the message it proposes for the current round and sends this to all other parties. Every party then proposes a list of $n - t$ properly signed messages for the multi-valued agreement. The external validity condition ensures that all messages in the decided list come with valid signatures, so that at least $n-2t \geq t+1$ of the messages come from honest parties. All messages in the agreed-on list are then delivered according to a fixed order. This atomic broadcast protocol guarantees liveness and fairness, i.e., a message broadcast by an honest party cannot be delayed arbitrarily by the adversary once it is known to at least $t+1$ honest parties.

A *secure causal atomic broadcast* is an atomic broadcast that also ensures a causal order among all broadcast messages, as put forward by Reiter and Birman [33]. It can be implemented by combining an atomic broadcast protocol that tolerates a Byzantine adversary with a robust threshold cryptosystem. Encryption ensures that messages remain secret up to the moment at which they are guaranteed to be delivered. Thus, client requests to a trusted service using this broadcast remain confidential until they are scheduled and answered by the service. The threshold cryptosystem must be secure against adaptive chosen-ciphertext attacks to prevent the adversary from submitting any related message for delivery, which would violate causality in our context. Maintaining causality is crucial in the asynchronous environment for replicating services that involve confidential data.

The protocol for secure causal atomic broadcast follows the basic idea of Reiter and Birman's protocol. By using the robust atomic broadcast mentioned before and the recent, non-interactive threshold cryptosystem of Shoup and Gennaro [36], it is actually the first secure implementation that we are aware of, and it can be proved secure in the random oracle model.

4 Generalized Adversary Structures

The common approach in fault-tolerant distributed systems is that at most a fraction of all servers fail. This model is based on the assumption that faults occur independently of each other and affect all servers equally likely. For random and uncorrelated faults within a system as well as isolated external events this seems adequate.

However, faults that represent malicious acts of an adversary may not always match these assumptions. This causes a conceptual obstacle for using the replication-based approach to achieve security in adversarial environments. In our setting, for example, if all servers in the system have a common vulnerability that permits a successful attack by an intruder, the integrity of the whole system may be violated easily. The independence assumption applies here only to the extent that the work needed for breaking into a server is the same for each machine. With the sophisticated tools, automated exploit scripts, and large-scale coordinated attacks found on the Internet today, this assumption becomes increasingly difficult to justify.

4.1 Concept

Our solution for this problem is to use *generalized adversary structures*. They can accommodate a strictly more general class of failures than any weighted threshold structure. In the Byzantine model, a collection of corruptible servers is also called an *adversary structure*. Such an adversary structure specifies the subsets of parties that may be corrupted at the same time.

We describe two concrete instantiations of such general adversary structures that are based on a classification of all servers according to one or more independent attributes with at least four values each.

Generalized adversary structures for secure fault-tolerant computing are also used in Byzantine Quorum systems [25] and the synchronous Byzantine agreement protocol of Fitzi and Maurer [17]; for combining them with threshold cryptography, we are restricted to those that correspond to linear secret sharing schemes [13].

Let $\mathcal{P} = \{1, \ldots, n\}$ denote the index set of all parties $P_1, \ldots, P_n$. The *adversary structure* $\mathcal{A}$ is a family of subsets of $\mathcal{P}$ that specifies which parties the adversary may corrupt. $\mathcal{A}$ is monotone (i.e., $S \in \mathcal{A}$ and $T \subset S$ imply $T \in \mathcal{A}$) and uniquely determined by the corresponding maximal adversary structure $\mathcal{A}^*$, in which no subset contains another one. In the traditional threshold model, the adversary may corrupt up to t arbitrary parties, and $\mathcal{A}^*$ contains all subsets of $\mathcal{P}$ with cardinality t.

Most protocols impose certain restrictions on the type of corruptions that they can tolerate. For a threshold adversary in an asynchronous distributed system model, $n > 3t$ is in general a necessary and sufficient condition. The analogous condition for protocols with a general adversary structure $\mathcal{A}$ is the so-called Q^3 *condition* [21]: no three of the sets in $\mathcal{A}$ cover $\mathcal{P}$. (Note that $n > 3t$ is a special case of this.)

The adversary structure specifies the (maximally) corruptible subsets of parties. Its complement is called the *access structure* and specifies the (minimally) qualified subsets that are needed to take some action. For example, it is used in secret sharing in cryptography [37], where it denotes the sets of parties who may reconstruct the shared secret. The access structure is usually the more important tool for the protocol designer than the adversary structure. In the example of the threshold system above, all sets of $t+1$ or more parties belong to the access structure.

4.2 Designing Protocols for General Adversary Structures

Every adversary structure can also be described by a Boolean function g on n variables that represent a subset of $\mathcal{P}$ as follows. We associate a subset of $\mathcal{P}$ with its *characteristic vector*, whose ith element is 1 if and only if P_i is in the subset. Defining g on all subsets of $\mathcal{P}$ in this way, g outputs 0 on all elements of the adversary structure (i.e., for all sets of parties that might be corrupted by the adversary), and 1 otherwise. To represent g, we use n-ary threshold gates Θ_k^n that output 1 if and only if at least k of their n inputs are 1 (note that AND and OR gates correspond to the special cases Θ_n^n and Θ_1^n). For instance, $g(S) = \Theta_{t+1}^n(S)$ in the threshold example.

All threshold-cryptographic protocols used by our architecture (cf. Section 3) can be extended to a generalized Q^3 adversary structure $\mathcal{A}$ for which the corresponding secret sharing access structure can be implemented by a linear secret sharing scheme. This requires changing some details in the cryptographic operations, but there are no essential difficulties. The agreement and broadcast protocols need to be changed as follows:

- Where a set of $n-t$ values is required, take all values in $\mathcal{P} \setminus S$ for some $S \in \mathcal{A}^*$.
- Where $2t+1$ values are needed, take all values in $S \cup T \cup \{i\}$ for any $S, T \in \mathcal{A}^*$ with $S \cap T = \emptyset$ and $i \notin S \cup T$.
- Where $t+1$ values are needed, take all values in $S \cup \{i\}$ for any $S \in \mathcal{A}^*$ and $i \notin S$.

4.3 Differentiating Servers by Attributes

Suppose there is an attribute of all parties in the system that takes on at least four different values. If the characteristics of corrupting a party vary with the attribute, then this classification can be exploited directly to design a system in which all parties in the same class may be corrupted simultaneously. With $n = 4$ this reduces to the threshold case. For example, the servers in a wide-area distributed system may vary by physical location, logical domain, system management personnel, type of operating system, other applications running on the same machine, and implementation of the protocols. All of these seem suitable attributes.

We do not make any further distinctions between the attribute values here; this leads naturally to a threshold failure model in the attribute dimension. However, arbitrary and complex relations between the parties can be modeled as long as there exists a corresponding linear secret sharing scheme. For simple linear relations, traditional weighted thresholds may already be enough, which can be obtained by allocating several logical parties to one physical party.

We give two examples of generalized adversary structures by describing the linear secret sharing schemes on which they are based. The first one shows how to combine attribute classification with the traditional threshold model, the second one is based on combining two separate classifications by independent attributes.

Before we continue, we need to introduce some notation. Let *class* denote the name of an attribute and also the set of attribute values. For $c \in \mathit{class}$, let $\chi_c : 2^{\mathcal{P}} \to \{0,1\}$ be the characteristic function of the attribute on a set of parties, i.e., $\chi_c(S) = 1$ if and only if $\mathit{class}(i) = c$ for some $i \in S$.

Example 1. Consider a system of nine servers and one attribute $class = \{a, b, c, d\}$ that satisfies

$$\begin{aligned} class(1) = \cdots = class(4) &= a \\ class(5) = class(6) &= b \\ class(7) = class(8) &= c \\ class(9) &= d. \end{aligned}$$

This could be the operating system of a server, for example, with class a representing a common but not very secure operating system and class d representing the most secure one of the four *class* values.

We want to design a system that tolerates the corruption of at most two arbitrary servers or all servers in any particular class; its adversary structure $\mathcal{A}_1$ is given by

$$g(S) = \overline{\Theta_3^9(S)} \vee \overline{\Theta_2^4(\chi_a(S), \chi_b(S), \chi_c(S), \chi_d(S))},$$

where Θ_k^n denotes a k-out-of-n threshold gate as before. $\mathcal{A}_1^*$ consists of $\{1, \ldots, 4\}$ and of all pairs of servers that are not both of class a. The corresponding access structure for secret sharing is given by

$$\overline{g(S)} = \Theta_3^9(S) \wedge \Theta_2^4(\chi_a(S), \chi_b(S), \chi_c(S), \chi_d(S)).$$

In other words, secrets may be reconstructed by coalitions of servers of size at least three that also cover at least two different classes.

One may readily verify that $\mathcal{A}_1$ satisfies the Q^3 condition. The corresponding linear secret sharing scheme follows directly from the expression for $\overline{g(S)}$ using the standard construction of Benaloh and Leichter [4]. The agreement and broadcast protocols can be adapted according to the modifications sketched above.

Example 2. The classification method works simultaneously for an arbitrary number of *independent* attributes and attribute values. We illustrate this for two attributes with four values each, denoted by $class_1 = \{a, b, c, d\}$ and $class_2 = \{\alpha, \beta, \gamma, \delta\}$. Assume all combinations of $class_1$ and $class_2$ exist so that $\mathcal{P}$ contains at least sixteen servers.

To be concrete, think of a distributed system of sixteen servers implementing a secure directory service for a large multi-national company that is running on nodes in New York (USA), Tokyo (Japan), Zurich (Switzerland), and Haifa (Israel) and consists of servers running the AIX, Windows NT, Linux, and Solaris operating systems. Thus, $class_1$ corresponds to the location and $class_2$ to the operating system of a server.

We can now obtain a distributed system that, for example, tolerates the *simultaneous* corruption of all servers with a particular operating system and all servers in one location. Its adversary structure is characterized by

$$g(S) = \overline{\Theta_2^4(x_a, x_b, x_c, x_d)} \vee \overline{\Theta_2^4(y_\alpha, y_\beta, y_\gamma, y_\delta)},$$

where

$$x_v = \Theta_2^4(\chi_v(S) \wedge \chi_\alpha(S),\ \chi_v(S) \wedge \chi_\beta(S),\ \chi_v(S) \wedge \chi_\gamma(S),\ \chi_v(S) \wedge \chi_\delta(S))$$

for $v \in class_1$, and

$$y_\nu = \Theta_2^4(\chi_a(S) \wedge \chi_\nu(S),\ \chi_b(S) \wedge \chi_\nu(S),\ \chi_c(S) \wedge \chi_\nu(S),\ \chi_d(S) \wedge \chi_\nu(S))$$

for $\nu \in class_2$.

This adversary structure satisfies the Q^3 condition, as can be verified easily. The corresponding secret sharing scheme is characterized by the negated expression, $\overline{g(S)} = \Theta_2^4(x_a, x_b, x_c, x_d) \wedge \Theta_2^4(y_\alpha, y_\beta, y_\gamma, y_\delta)$. Intuitively, the sharing introduces two secret values (one for each class) at the top level that must both be known to recover the secret. To reconstruct the first one, at least two of the four $class_1$ points $x_a, \ldots, x_d$ must be known; each point in turn can again only be reconstructed by a subset that covers at least two $class_2$ values *and* the corresponding $class_1$ value. In other words, x_a is shared among the four parties with a $class_1$ value of a using a two-out-of-four scheme, etc. The top-level secret for $class_2$ is distributed analogously.

The resulting distributed system maintains liveness and safety as long as there are servers with three operating systems at three locations that are uncorrupted; but one location may be unreachable and one operating system could contain easily exploitable vulnerabilities so that a maximum of seven servers could have failed at any moment. Note that all solutions based on thresholds can tolerate at most five corruptions among the 16 servers.

5 Applications

Our distributed trusted services are based on secure state machine replication in the Byzantine model (following [34, 33]). Requests to a particular service are delivered by the broadcast protocols mentioned in Section 3. A broadcast is started when the client sends a message containing the request to a sufficient number of servers. In general, the client must send the request to more than t servers or the corrupted servers could simply ignore the message; alternatively, one could postulate that one server acts as a gateway to relay the request to all servers and leave it to the client to resend its message if it receives no answer within the expected time.

Depending on whether it needs to maintain causality among client requests, a service may use atomic broadcast directly or secure causal atomic broadcast otherwise. If the client requests commute, reliable broadcast suffices.

Each server returns a partial answer to the client, who must wait for at least $2t + 1$ values before determining the proper answer by majority vote. Since atomic broadcast guarantees that all servers process the same sequence of requests, the client will obtain the same answer from all honest servers. If the application returns a digital signature, the answers may contain signature shares from which the client can recover a threshold signature.

We sketch two applications here and refer to [6] for more details; an authentication service and a trusted party for fair exchange are also described there.

5.1 Certification Authority and Directory Service

A *certification authority* (CA) is a service run by a trusted organization that verifies and confirms the validity of public keys. The issued *certificates* usually also confirm that the real-world user defined in a certificate is in control of the corresponding private key. A certificate is simply a digital signature under the CA's private signing key on the public key and the identity (ID) claimed by the user.

The CA has published its own public key of a digital signature scheme. When a user wants to obtain a certificate for his public key, he sends it together with his ID and credentials to the CA. The ID might consist of name, address, email, date of birth, and other data to uniquely identify the holder. Then the CA verifies the credentials, produces a certificate if they pass, and sends the answer back to the user. The user can verify his certificate with the public key of the CA. For its certificates to be meaningful, the CA must have a clearly stated and publicized policy that it follows for validating public keys and IDs; this policy might change over time. We ignore revocation and other issues for the moment.

A *secure directory* service maintains a database of entries, processes lookup queries, and returns the answers authenticated by a signature under its private signing key. The corresponding signature verification key is available to all clients. Several examples of secure directories exist in distributed systems today and more are needed in the future, such as DNS authentication.

Internally, a secure directory works much like a CA: it receives a query, retrieves some values from the stored database, generates a digital signature on the result, and sends both back to the client. Additional functionality is needed for updating the database.

Both services can be implemented in our distributed system architecture. Requests must be delivered by atomic broadcast to ensure that all servers return the same answers. Updates to the database must be treated in the same way. The digital signature scheme of the service is replaced by the corresponding threshold signature scheme, which requires minimal changes to the clients in the case of [35]. In the server code, computing the digital signature is replaced by generating a signature share.

Note that atomic broadcast is crucial for delivering any request that changes the global state; only if a CA never changes its policy and all of its certificates are independent of each other does it suffice to use reliable broadcast.

5.2 Notary Service

In its simplest form, a *digital notary service* receives documents, assigns a sequence number to them, and certifies this by its signature. Such a service could, e.g., be used for assigning Internet domain names or registering patent applications. A notary must process requests sequentially and atomically; it updates its internal state for each request.

In many notary applications, the content of a request must remain confidential until the notary processes it in a single atomic step. For example, a competitor in the patent application scenario might try to file a related patent application, have it processed first, and claim the invention for himself.

A distributed notary can be realized readily using our architecture since it involves a simple state machine to be replicated. Client requests must be disseminated by secure causal atomic broadcast to rule out any violation of their confidentiality. For if no encryption were used, a corrupted server could see the contents of the request during atomic broadcast and arrange that the service schedules and processes a related request of the adversary before the original one. The same attack is possible if the cryptosystem is not secure against adaptive chosen-ciphertext attacks.

As the answer of the notary service is a digitally signed message, clients obtain their receipt as described before in the CA example.

6 Extensions

We mention some extensions and improvements of our architecture. Although we have strived for a secure and fault-tolerant system in the given environment, the security could be strengthened by using "proactive" protocols, allowing for dynamic group changes, or using hybrid failure structures (not to be confused with generalized ones). Our atomic broadcast protocols involve a considerable overhead, in particular for large n, because security has been our primary design principle. Among the various possible optimizations, it seems most promising to design "optimistic" protocols, which run very fast if no corruptions occur but may fall back to a slower mode if necessary.

Proactive Protocols. Proactive security is a method to protect threshold-cryptographic schemes against a mobile adversary that can corrupt all parties during the lifetime of the system, but never more than t at once (see [9] for a survey). Proactive protocols divide time into epochs. All parties "reshare" their cryptographic secret keys between two epochs and delete all old key material. The model assumes an external mechanism for detecting corruptions and "cleaning up" a party. Because all secrets that the adversary has seen in the past become useless by resharing, the adversary never knows enough secret information to compromise the whole system.

Proactively secure protocols for our asynchronous system model are currently not known. One issue to be addressed first is how to integrate epochs into the asynchronous system model.

Dynamic Groups. The static nature of our system model may pose a problem for practical systems. Real systems evolve over time and grow or shrink together with the organizations that use them. Every such change would require a fresh setup of the complete system by a trusted dealer. Ideally, a system should reconfigure itself and dynamically increase or decrease the group size and the thresholds. However, special care is needed to ensure the safety of all keys during the changes; thus, at least some resharing of keys will be needed as in proactive protocols.

Note that similar motivation has led to the important idea of view-based group communication systems that tolerate crash failures. But dynamically changing the group seems much harder in the Byzantine model when cryptographic secrets are involved; this is currently an open problem.

Hybrid Failure Structures. Another interesting direction is to treat crash failures separately from corruptions and adapt the protocols to such hybrid failure structures. After all, crashes are more likely to occur than intrusions and they are much easier to handle than Byzantine corruptions. For coping with transient server outages, the crash-recovery model seems also plausible (see references in [20]). Protocols in hybrid failure models have been investigated before [19] so that we expect this to be feasible.

Optimistic Protocols. Optimistic protocols run very fast if no malicious adversary is at work and all messages are delivered promptly. If a problem is detected (typically because liveness is violated), they may switch into a more secure mode using protocols that guarantee progress. This idea is quite common in the literature [29, 11]. In our Byzantine context, one has to make sure that safety is never violated, though. Kursawe and Shoup [23] recently designed a protocol along these lines.

Acknowledgments

This paper reflects the results of many discussions with Klaus Kursawe, Frank Petzold, Jonathan Poritz, Victor Shoup, and Michael Waidner; I am grateful for all their contributions.

This work was supported by the European IST Project MAFTIA (IST-1999-11583), but represents the view of the author. The MAFTIA project is partially funded by the European Commission and the Swiss Department for Education and Science.

References

[1] R. Baldoni, J.-M. Helary, and M. Raynal, "From crash fault-tolerance to arbitrary-fault tolerance: Towards a modular approach," in *Proc. International Conference on Dependable Systems and Networks (FTCS-30/DCCA-8)*, pp. 273–282, 2000.

[2] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. 1st ACM Conference on Computer and Communications Security*, 1993.

[3] M. Ben-Or, "Another advantage of free choice: Completely asynchronous agreement protocols," in *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC)*, 1983.

[4] J. Benaloh and J. Leichter, "Generalized secret sharing and monotone functions," in *Advances in Cryptology: CRYPTO '88* (S. Goldwasser, ed.), vol. 403 of *Lecture Notes in Computer Science*, pp. 27–35, Springer, 1990.

[5] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *Journal of the ACM*, vol. 32, pp. 824–840, Oct. 1985.

[6] C. Cachin, ed., *Specification of Dependable Trusted Third Parties.* Deliverable D26, Project MAFTIA IST-1999-11583, Jan. 2001. Also available as Research Report RZ 3318, IBM Research.

[7] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols." Cryptology ePrint Archive, Report 2001/006, Mar. 2001. `http://eprint.iacr.org/`.

[8] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography," in *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 123–132, 2000. Full version available from Cryptology ePrint Archive, Report 2000/034, `http://eprint.iacr.org/`.

[9] R. Canetti, R. Gennaro, A. Herzberg, and D. Naor, "Proactive security: Long-term protection against break-ins," *RSA Laboratories' CryptoBytes*, vol. 3, no. 1, 1997.

[10] R. Canetti and T. Rabin, "Fast asynchronous Byzantine agreement with optimal resilience," in *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 42–51, 1993. Updated version available from `http://www.research.ibm.com/security/`.

[11] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. Third Symp. Operating Systems Design and Implementation (OSDI)*, 1999.

[12] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[13] R. Cramer, I. B. Damgård, and U. Maurer, "General secure multi-party computation from any linear secret sharing scheme," in *Advances in Cryptology: EUROCRYPT 2000* (B. Preneel, ed.), vol. 1087 of *Lecture Notes in Computer Science*, Springer, 2000.

[14] Y. Deswarte, L. Blain, and J.-C. Fabre, "Intrusion tolerance in distributed computing systems," in *Proc. 12th IEEE Symposium on Security & Privacy*, pp. 110–121, 1991.

[15] A. Doudou, B. Garbinato, and R. Guerraoui, "Abstractions for devising Byzantine-resilient state machine replication," in *Proc. 19th Symposium on Reliable Distributed Systems (SRDS 2000)*, pp. 144–152, 2000.

[16] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, pp. 374–382, Apr. 1985.

[17] M. Fitzi and U. Maurer, "Efficient Byzantine agreement secure against general adversaries," in *Proc. 12th International Symposium on Distributed Computing (DISC)*, vol. 1499 of *Lecture Notes in Computer Science*, pp. 134–148, Springer, 1998.

[18] J. A. Garay, R. Gennaro, C. Jutla, and T. Rabin, "Secure distributed storage and retrieval." Proc. 11th International Workshop on Distributed Algorithms (WDAG), 1997.

[19] J. A. Garay and K. J. Perry, "A continuum of failure models for distributed computing," in *Proc. 6th International Workshop on Distributed Algorithms (WDAG)*, vol. 647 of *Lecture Notes in Computer Science*, pp. 153–165, Springer, 1992.

[20] R. Guerraoui, M. Hurfin, A. Mostefaoui, R. Oliveira, M. Raynal, and A. Schiper, "Consensus in asynchronous distributed systems: A concise guided tour," in *Advances in Distributed Systems* (S. Krakowiak and S. Shrivastava, eds.), vol. 1752 of *Lecture Notes in Computer Science*, pp. 33–47, Springer, 2000.

[21] M. Hirt and U. Maurer, "Player simulation and general adversary structures in perfect multi-party computation," *Journal of Cryptology*, vol. 13, no. 1, pp. 31–60, 2000.

[22] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "The SecureRing protocols for securing group communication," in *Proc. 31st Hawaii International Conference on System Sciences*, pp. 317–326, IEEE, Jan. 1998.

[23] K. Kursawe and V. Shoup, "Optimistic asynchronous atomic broadcast." Cryptology ePrint Archive, Report 2001/022, Mar. 2001. `http://eprint.iacr.org/`.

[24] D. Malkhi, M. Merritt, and O. Rodeh, "Secure reliable multicast protocols in a WAN," *Distributed Computing*, vol. 13, no. 1, pp. 19–28, 2000.

[25] D. Malkhi and M. K. Reiter, "Byzantine quorum systems," *Distributed Computing*, vol. 11, no. 4, pp. 203–213, 1998.

[26] D. Malkhi and M. K. Reiter, "An architecture for survivable coordination in large distributed systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 2, pp. 187–202, 2000.

[27] L. E. Moser and P. M. Melliar-Smith, "Byzantine-resistant total ordering algorithms," *Information and Computation*, vol. 150, pp. 75–111, 1999.

[28] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, pp. 228–234, Apr. 1980.

[29] F. Pedone and A. Schiper, "Optimistic atomic broadcast," in *Proc. 12th International Symposium on Distributed Computing (DISC)*, 1998.

[30] D. Powell (Guest Ed.), "Group communication," *Communications of the ACM*, vol. 39, pp. 50–97, Apr. 1996.

[31] M. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in Rampart," in *Proc. 2nd ACM Conference on Computer and Communications Security*, 1994.

[32] M. K. Reiter, "Distributing trust with the Rampart toolkit," *Communications of the ACM*, vol. 39, pp. 71–74, Apr. 1996.

[33] M. K. Reiter and K. P. Birman, "How to securely replicate services," *ACM Transactions on Programming Languages and Systems*, vol. 16, pp. 986–1009, May 1994.

[34] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys*, vol. 22, Dec. 1990.

[35] V. Shoup, "Practical threshold signatures," in *Advances in Cryptology: EUROCRYPT 2000* (B. Preneel, ed.), vol. 1087 of *Lecture Notes in Computer Science*, pp. 207–220, Springer, 2000.

[36] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," in *Advances in Cryptology: EUROCRYPT '98* (K. Nyberg, ed.), vol. 1403 of *Lecture Notes in Computer Science*, Springer, 1998.

[37] D. R. Stinson, *Cryptography: Theory and Practice.* CRC Press, 1995.

[38] P. Veríssimo, A. Casimiro, and C. Fetzer, "The timely computing base: Timely actions in the presence of uncertain timeliness," in *Proc. International Conference on Dependable Systems and Networks (FTCS-30/DCCA-8)*, pp. 533–542, 2000.