Byzantine Failures and Security: Arbitrary is not (always) Random

Felix C. Gärtner
Swiss Federal Institute of Technology (EPFL)
School of Computer and Communication Sciences
Distributed Programming Laboratory
CH-1015 Lausanne, Switzerland
fcg@acm.org

Abstract: The Byzantine failure model allows arbitrary behavior of a certain fractionof network nodes in a distributed system. It was introduced to model and analyze the effects of very severe hardware faults in aircraft control systems. Lately, the Byzantine failure model has been used in the area of network security where Byzantine-tolerance is equated with resilience against malicious attackers. We discuss two reasons why one should be careful in doing so. Firstly, Byzantine-tolerance is not concerned with secrecy and so special means have to be employed if secrecy is a desired system property. Secondly, in contrast to the domain of hardware faults, in a security setting it is difficult to compute the assumption coverage of the Byzantine failure model, i.e., the probability that the failure assumption holds in practice. To address this latter point we develop a methodology which allows to estimate the reliability of a Byzantine-tolerant solution exposed to attackers of different strengths.

1 Introduction

A group of scientists from all over the world regularly gathers for scientific meetings do discuss recent advances in their field. The meetings are organized by one particular member of this group who—when necessary—fixes a date and a place for the meeting and informs all other members of the group. However, the good spirit which usually prevailed in this group deteriorated during the most recent meeting because some members hadn't shown up. In fact, exactly those members didn't show up which had severely criticized the results presented by the organizer during the last meeting. The organizer, however, confirmed that he had informed everybody about the meeting. Contacted via telephone, the missing participants stated that they had in fact been notified about the meeting but they thought the meeting was to be held at a different date. They accused the organizer of not having given them the correct information because he was reluctant to discuss his results with them again. The organizer protested against these accusation and said that this was a conspiracy to bring him into discredit. The participants now discussed how to avoid this Problem for the next meeting.

The problem description above (which is taken from [Bo97]) is an instance of what is known as the *Interactive Consistency Problem* [LSP82]. Interactive Consistency is one of the most fundamental tasks in fault-tolerant distributed computing and a solution to this problem has many applications. For example, a critical (database) server for example is replicated several times to ensure availability even when one of the servers fails. However, all servers must update their state in the same way so that accesses to the replicated data do not return inconsistent values. The server which is about to initiate an update to its state plays the role of the conference organizer in the scenario above. The other servers resemble the conference participants. A protocol which solves the Interactive Consistency Problem allows to prevent the confusion which happened above and ensures that all (well-behaving) participants consistently agree on the same update.

Interactive Consistency is sometimes also known as the *Byzantine Generals Problem* [LSP82] which was originally presented in a more militaristic setting where the conference participants are generals of the Byzantine army which want to agree on whether to attack an enemy town or not. The generals are either loyal or traitors. Traitors can act in arbitrary ways, i.e., they can send conflicting messages to the other generals during decision making. From this story, the assumption that computer components can act in arbitrary ways is now called the *Byzantine failure assumption*. It originated from work in the area of fault-tolerant aircraft control systems [WLG$^+$78] and was intended to model worst case faulty behavior.

Recently, we are seeing an increasing use of the Byzantine failure model in the area of security [Re96, KM99, CKS00, CLNV02, Ku02, YMV$^+$02, ZSvR02]. This is because the worst case assumption of arbitrary behavior can also be regarded as malicious behavior of an attacker of the system. Hence, a Byzantine-tolerant system can also be regarded as secure against malicious attackers.

There are two main concerns indicating that the Byzantine failure model should be used with care in the context of secure systems. The first concern is that Byzantine-tolerant solutions (for example to Interactive Consistency) are not concerned with *secrecy*. For example, if a database is fully replicated to maintain availability in the presence of Byzantine failures, an attacker compromising at least one database server will know the entire contents of the database. If file contents are required to be kept secret, then standard algorithms for Interactive Consistency from the fault-tolerance area (e.g., the classic ones of [LSP82] or [BO83]) can only be applied if specific firewall architectures are used [YMV$^+$02].

If secrecy of replicated data is required, mechanisms from cryptography have to be applied, usually some instance of *secret sharing*. In this approach, data is stored in a distributed way so that it needs a certain fraction of nodes to collaborate in the reconstruction of the data. If this fraction is higher than the number of nodes which may be faulty, then the Byzantine nodes cannot by themselves reconstruct arbitrary data. However, if standard agreement protocols are used, nothing can prevent a Byzantine node from receiving information about what has been proposed as the database update by individual processes.

Interestingly, a similar problem to Interactive Consistency where individual process inputs can be kept secret has been studied for years in the area of cryptography under the heading of *secure multi-party computation* (see Goldwasser's invited lecture [Go97] or Goldreich's survey [Go02] for an overview). In a secure multi-party computation, a set of processes wants to compute a deterministic function of all their inputs without trusting each other. A common example is the millionaires' problem: Two millionaires want to know who of them has more money but they don't want to tell each other how much they really have. A multi-party computation can compute *any* function even though a certain fraction of nodes exhibits Byzantine behavior. It does this in a way so that a Byzantine process learns nothing about the inputs of other processes to the computation (apart from what is derivable from the result of the function). It is obvious that Interactive Consistency can be formulated as an instance of secure multi-party computation and often researchers in fault-tolerance are not aware of this fact leading to multiple re-inventions of the wheel.

The second concern suggesting care in using Byzantine failures in security is related to measures of reliability. The Interactive Consistency problem (as well as secure multi-party computation) is provably impossible if at least one third of the processes can behave in a Byzantine fashion [PSL80]. If we denote by f the maximum number of faulty processes and by n the total number of processes, this means that we need $f < n/3$ for the problem to be solvable. In practice, we cannot predict with certainty that $f < n/3$ will hold, we (sometimes) can merely give a probability, which is called the *assumption coverage* [Po92]. The best solution to Interactive Consistency is useless in practice if the assumption coverage of $f < n/3$ is 0, i.e., in practical situations where this bound cannot be guaranteed.

In the fault-tolerance domain, calculating assumption coverage is an established area. The relevant input measure is the *reliability* of a component which is defined as the probability that the component is not faulty until a certain time t. Experience from large data collections of component failures shows that the reliability of components can be modeled as a random variable satisfying a certain distribution [Bi99]. The overall reliability of a complex system can then be calculated from the reliability of its components. An important assumption which makes these calculations possible (and which can be justified in practice) is that the failures of the nodes are statistically independent.

In the area of security, there are (up to now) no extensive and standardized databases of security violations (maybe the closest is the recently established Internet Storm Center [Th03]). It is also often argued that Byzantine failures in the security setting are not random events: If a new security vulnerability is published, the probability of a server being compromised rises. But maybe most importantly it is very difficult to argue that the occurrences of security related Byzantine failures are statistically independent (especially if all servers run the same operating system). So although Byzantine failures still result in arbitrary behavior, they are not random events in security.

There has been some work to study the effects of different levels of reliability on Byzantine-tolerant solutions in the area of fault-tolerance [Re85, Ba87, Va93]. For example, Reischuk [Re85] gives a protocol for Interactive Consistency and analyzes the probability of termination given the reliability of the individual nodes and communication channels. All this work assumes statistically independent failures.

In this paper we study ways of measuring the reliability of a system in a security setting.

We argue that in such a setting two types of (often indistinguishable) Byzantine behavior can occur: (a) traditional Byzantine failures through hardware faults, and (b) Byzantine behavior as a result of the server being fully compromised by an attacker. We introduce a work-factor related measure of faults of class (b) and show how it can be related to the traditional measuring techniques of faults of class (a). To the best of our knowledge, this is the first work which enables statements about the reliability of a system in the presence of certain types of attackers.

2 Background

2.1 General System Setting

The general system setting is a group of n servers (sometimes also called nodes) in a fully connected network which communicate via message passing. For simplicity (and to avoid known impossibilities [FLP85]) we assume a synchronous model of computation, i.e., there are known upper bounds on message delivery delays and the relative difference in server execution speeds.

Servers can become faulty by exhibiting arbitrary (Byzantine) behavior due to two different types of reasons:

- Hardware faults: occurrence of these faults is random and statistically independent.
- Server takeovers by an attack: in this case we assume that an adversary has full control over the server.

We denote by f the maximum number of servers which are allowed to behave arbitrarily. If not stated otherwise, we assume that $f < n/3$.

2.2 Measuring Reliability

The *reliability* R of a system is the probability that it will perform its expected function for a given time interval [Bi99]. Taking the time interval as a parameter, the *reliability function* $R(t)$ denotes the probability that the system is non-faulty until time t. In general, we assume that $R(0) = 1$.

The failure-free operating time of a system is generally a random variable [Bi99, p. 3]. The *empirical failure rate* $\hat{\lambda}(t)$ of a system is the number of failures per time unit and computed by looking at the time it takes of a set of statistically identical systems to exhibit faulty behavior. If the number of observed components rises, the empirical failure rate is assumed to converge to the "true" *failure rate* $\lambda(t)$. The failure rate fully determines the reliability function in the following way [Bi99, p. 5]:

$$R(t) = e^{-\int_0^t \lambda(x)dx} \tag{1}$$

In practice it is usually assumed that the failure rate is (nearly) constant and so for all $t \geq 0$ we have that $\lambda(t) = \lambda$. From Eq. 1 follows that

$$R(t) = e^{-\lambda t} \tag{2}$$

In this case the failure-free operating time is exponentially distributed. Typical figures for λ in practice are 10^{-10} to 10^{-9}h^{-1} [Bi99]. The well-known measures of MTTF (mean time to failure) can be calculated from the reliability function.

Given a set of n servers each of which having the same reliability function $R(t)$, we are interested in the overall reliability function $R_s(t)$ of a system where at least k out of n servers remain non-faulty. For the case of statistically independent failures this can be computed as the sum of the probabilities where *exactly* k out of n, exactly $k+1$ out of n, ..., exactly n out of n servers remain non-faulty (note that these cases are mutually exclusive). The individual probabilities (i.e., where exactly k out of n are non-faulty) can be calculated in the standard way using the binomial coefficient $\binom{n}{k}$ giving the number of all distinct combinations of k elements from a set of n elements. This results in the following formula:

$$R_s(k, n, t) = \sum_{i=k}^{n} \binom{n}{i} R^i(t) \big(1 - R(t)\big)^{n-i} \tag{3}$$

Eq. 3 will be important later to evaluate the reliability of the system in the presence of attacks.

2.3 Measuring Security

How to correctly assess the security of a given system is still very much an open problem (see for example the many different views on this subject in [Ap01]). Schneier [Sc01] even argues that developing useful security related measures is not possible now, and might never be.

An example from the area of physical security which is often mentioned in this context is the rating standard of safes by the company Underwriters Laboratories [Un00]. The standard distinguishes between the methods and the time needed to break into a certain safe. Ratings can be "tool resistant" (TL), i.e., resistant against hand and power tools (hammers, drills), "torch resistant" (TR), i.e., resistant against oxyacetylene cutting torch, and "explosive resistant" (TX). An overall rating of TL-30 for example means that the safe is tool resistant for at least 30 man-minutes. The notable difference between this measure and the reliability metric used in fault-tolerance is that it is an *effort related measure*. Effort is usually associated with the time and the "force" which an adversary must invest to perform the attack. Effort related measures also prevail in cryptography where security is often proven with respect to an adversary who is computationally bounded (e.g., by an arbitrary polynom). In security, effort (or work-factor) related measures seem to be more appropriate than mere probabilities. However, the problems (given by Schneier [Sc01]) for practical security measures, like time- and context-dependencies, have limited concrete proposals (maybe the most advanced are Schneier's attack trees [Sc99]) and resulted in some rather theoretical statements like Jacob's security orderings [Ja92] or Meadows' denial-of-service framework [Me98].

3 Coverage Calculations in a Security Context

3.1 Defining Attacker Classes

Because of the difficulties in defining concrete security measures which were explained in Sect. 2, we define a set of effort-related general attacker classes with ideas borrowed from complexity theory. Similar to the measurements of reliability we regard the system which is put into operation at a time $t = 0$ and observe its behavior if it is exposed to an adversary from a certain attacker class. The attacker classes basically define how fast an attacker is able to compromise individual servers of the system.

We define an *attacker function* $C(t)$ which returns the number of servers which have been fully compromised until time t. We make several simplifying assumptions here which have analogies to the area of reliability measurements. Firstly, we do not consider "partial failure", i.e., either the attacker has fully compromised the system or not. In practice this means that we just consider attacks in which at some point an attacker gains superuser (root) privileges and had no control over the server before. Secondly, we do not consider "repair", i.e., once a server has been compromised there is no action to restore it in an uncompromised state. In practice this means that the system administrator does not reboot the server from a "clean" image of the operating system.

We now define and motivate four different classes of attackers. All classes are parametrized by some value p.

- Constant time attacker with parameter $p > 0$.

  Such an attacker needs a constant amount of time, namely p, to fully compromise *all* servers in the system. This models scenarios in practice where all servers run the same operating system and an attacker exploits some (un)known vulnerability.

  The attacker function of the constant time attacker with parameter p is defined as follows:

$$C(t) = \begin{cases} 0 & \text{if } t \leq p, \\ n & \text{otherwise.} \end{cases} \tag{4}$$

- Linear time attacker with parameter $p > 0$.

  Such an attacker compromises servers one after the other, i.e., he needs the same amount of time, namely p, anew to break in to a server. This models scenarios in practice where operating system diversity is applied to prevent constant time attacks, e.g., every server is running a different version of the operating system in the hope that not all exhibit the same vulnerability.

The attacker function of the linear time attacker with parameter p is defined as follows:

$$C(t) = \min\{\lfloor t \cdot p \rfloor, n\} \tag{5}$$

- Logarithmic time attacker with parameter $p > 1$.

  Such an attacker needs a certain amount of time to break in to the first server. The time to break in to the second server follows a logarithmic curve, i.e., it takes increasingly longer to compromise each attacked server. This models scenarios in practice where the system administrators have discovered that their system is under attack and are constantly strengthening the defenses of the remaining (uncompromised) servers, making it increasingly hard for the attacker to break in.

  The attacker function of the logarithmic time attacker with parameter p is defined as follows:

$$C(t) = \min\{\lfloor \log_p t \rfloor, n\} \tag{6}$$

- Polynomial time attacker with parameter $p > 0$.

  Such an attacker gets increasingly faster in breaking in to the servers of the system. This models scenarios in practice where the attacker "learns" secret information about the attacked system from breaking in to servers. For example, breaking in to the first server makes it easier to exploit vulnerabilities originating from file sharing mechanisms.

  The attacker function of the polynomial time attacker with parameter p is defined as follows:

$$C(t) = \min\{\lfloor t^p \rfloor, n\} \tag{7}$$

More classes can be defined analogously.

3.2 Relating Reliability to Attacker Classes

To give an estimate of the system reliability in the presence of attacks, we need to do two things:

1. Choose an attacker class which is reasonable for the given setting and instantiate the parameter p. From this, derive the attacker function $C(t)$. For now, we have to defer concrete rules of how to do this to future work. Note however that these classes can be (partially) ordered according to their strength, i.e., assuming a constant time attacker is clearly more pessimistic than assuming a linear time attacker.

2. Choose a reliability function $R(t)$ with respect to hardware failures for an individual server. This can be done with established methodologies mentioned in Sect. 2.

The idea of the following calculations is that the attacker function limits the "remaining" redundancy for hardware faults. For example, if no server has been compromised, for

$f = 3$ and $n = 10$, the reliability of the entire system is calculated from the Eq. 3 for a "k out of n" system where $k = n - f$. However, if a single server has already been compromised, this server is not part anymore of the system redundancy used to tolerate hardware faults. This means the reliability decreases to that of a "k out of $n - 1$" system.

In general, the overall reliability R_C for f and n in the presence of attacker $C(t)$ calculates to the reliability of a "k out of $n - C(t)$" system. Taking Eq. 3 this can be formalized as:

$$\begin{aligned} R_C(f,n,t) &= R_s(n-f, n-C(t), t) \\ &= \sum_{i=n-f}^{n-C(t)} \binom{n-C(t)}{i} R^i(t)\big(1-R(t)\big)^{n-C(t)-i} \end{aligned} \quad (8)$$

Note that the reliability drops to 0 once $C(t) \geq f$, i.e., in this case the attacker has used up the entire redundancy of the system. The calculation implicitly assumes that an attacker may not compromise a faulty server. It also makes the worst case assumption that faults do not affect compromised servers.

3.3 Examples

Using the definition of R_C above it is now possible to give an account of the reliability of a certain system in the presence of attacks. We have calculated the reliability functions for two different classes of attackers (constant and linear time) with different parameters p. The failure rate λ defining the reliability of an individual server was constantly set to 10^{-2}. The number of servers n was set to 10 and the maximum number of failures f was set to 3.

The case of the constant time attacker is plotted in Fig. 1 for a time range t between 0 and 100. The strength of the attacker p varies between 0 and 160. It can be seen that the reliability function R_C follows the original curve but suddenly drops to 0 once p time has elapsed, i.e., once the attacker has compromised the entire system. Note that a lower value of p indicates a faster and therefore stronger attacker. The figure shows that the reliability of the system remains unchanged if p is sufficiently large.

The case of the linear time attacker is plotted in Fig. 2 for a time t between 0 and 100. This time the strength p of the attacker varies between 0 and 0.1. Note that in contrast to the constant time attacker, larger values of p indicate stronger adversaries (more compromised servers per time unit) and so for increasing values of p, the system reliability R_C decreases faster. For example, for $p = 0.05$, the first server is compromised at $t = 20$ leading to a sudden but slight drop of the reliability function. For $p = 0.1$, the first server is compromised at time $t = 10$, the second server at time $t = 20$, the third one at $t = 30$. As soon as the fourth server is compromised at time $t = 40$, the reliability drops to 0. The figure shows that linear time attackers do not necessarily have "linear success" in degrading system reliability: The effect on system reliability depends on the lifetime of

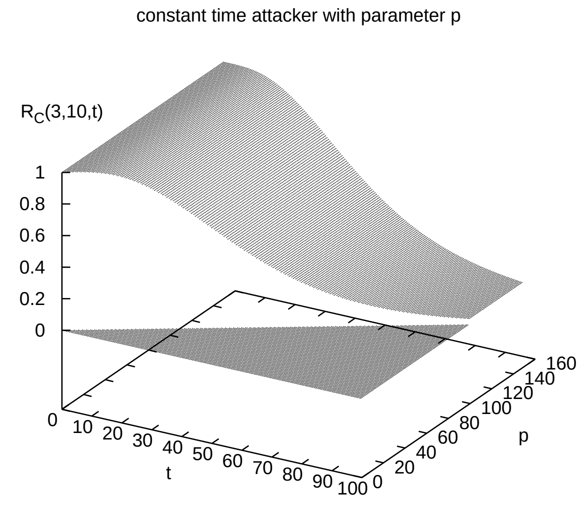

Figure 1: System reliability assuming a constant time attacker of varying strengths.

the system, e.g., in advanced lifetimes every newly compromised server decreases system reliability less than the previous one.

The data plotted in Fig. 2 can also be visualized similar to characteristic curves of transistor elements which are known from electrical engineering. Fig. 3 can be used by an engineer to determine the reliability of his system with respect to a particular strength of an attacker. For example, the reliability of the system at time $t = 40$ is 0.6 for an attacker of strength 0 and 0.025, but 0.4 for an attacker of strength 0.05. Similarly the graph can be used to derive the maximum attacker strength tolerable given a certain target reliability and mission time. For example, for a mission duration of 50 and a minimum reliability of 0.2 the system can sustain an attacker of strength 0.025 and below.

4 Conclusions and Future Work

In this paper we have argued that one should be careful when using the Byzantine failure model in the context of security. The main reason for this is that there is no accepted measure to estimate the "coverage" of the necessary assumption that at most a certain fraction of processes can be fully compromised. We have introduced an effort-related form to model attackers and a corresponding set of attacker classes that allow to attribute some meaning to the statement that "the reliability of the system at time t is x" in the presence of attacks. The attacker classes group different adversaries by distinguishing the "speed" (in a complexity theoretic sense) in which they can compromise servers. This can be seen as a first step to reconcile standard reliability measures from fault-tolerance with

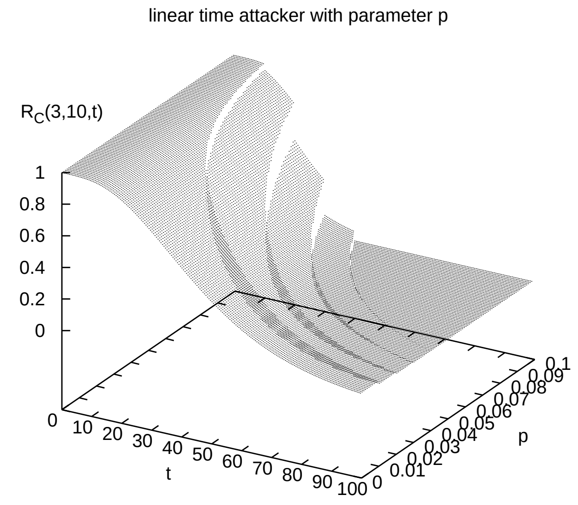

Figure 2: System reliability assuming a linear time attacker of varying strengths.

new effort-related measures from security.

The problem of choosing the right attacker class and instantiating the attacker parameter p remains and leaves room for much future work. From the motivating examples given in Sect. 3 it should be clear that the different attacker classes do have some relation to practice and so it may after all be feasible to arrive at a sensible choice of an attacker class considering a specific application scenario. Such a classification would be helpful in the area of risk management [GLS03] since it would provide some indication of how much company resources should be invested into countermeasures (i.e., reducing the attacker strength or improving the attacker class).

Another line of future work could more closely investigate the interrelations between attacker classes and system reliability. The figures in Sect. 3 show nicely that there are tradeoffs between attacker class, attacker strength and system reliability. For different values of p the system reliability can be higher even if a more severe attacker class is chosen. The simplifying assumptions made in Sect. 3 also invite to explore many other questions, for example to consider attacker models with “repair” and their effect on system reliability.

Acknowledgments

The author wishes to thank Karl-Erwin Großpietsch, Martin Hirt, Klaus Kursawe and Michael Waidner for helpful discussions. Comments by Markus Schumacher and Arshad Jhumka on an earlier version of this paper are also gratefully acknowledged. Work was supported by Deutsche Forschungsgemeinschaft (DFG) as part of the Emmy Noether programme.

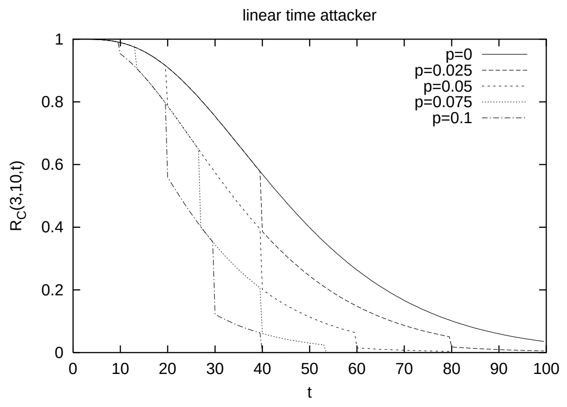

Figure 3: System reliability assuming a linear time attacker of strength p for three different values.

References

[Ap01] Applied Computer Security Associates. Proc. Workshop on Internet Security Systems Scoring and Ranking. Internet: `http://www.acsac.org/measurement/proceedings/wisssr1-proceedings.pdf`. May 2001.

[Ba87] Babaoğlu, Ö.: On the reliability of consensus-based fault-tolerant distributed computing systems. *ACM Trans. Computer Systems*. 5(4):394–416. 1987.

[Bi99] Birolini, A.: *Reliability Engineering: Theory and Practice*. Springer. 3rd Ed. 1999.

[BO83] Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols. In: *Proc. 2nd ACM Symp. on Principles of Distributed Computing*. pp. 27–30. 1983.

[Bo97] Borcherding, M.: *Authentifikationsvoraussetzungen für effiziente byzantinische Übereinstimmung*. PhD thesis. Univ. Karlsruhe, Fak. f. Informatik. 1997. Logos, Berlin.

[CKPS01b] Cachin, C., Kursawe, K., Petzold, F., und Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: *Advances in Cryptology*. LNCS. Springer. 2001.

[CKS00] Cachin, C., Kursawe, K., und Shoup, V.: Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. In: *Proc. Symp. on Principles of Distributed Computing*. pp. 123–132. Portland, Oregon. 2000.

[CLNV02] Correia, M., Lung, L. C., Neves, N. F., und Veríssimo, P.: Efficient Byzantine-resilient reliable multicast on a hybrid failure model. In: *Proc. 21st Symposium on Reliable Distributed Systems*. Suita, Japan. Oct. 2002.

[FLP85] Fischer, M. J., Lynch, N. A., und Paterson, M. S.: Impossibility of distributed consensus with one faulty process. *J. ACM*. 32(2):374–382. Apr. 1985.

[GLS03] Gordon, L. A., Loeb, M. P., und Sohail, T.: A framework for using insurance for cyber-risk management. *Comm. ACM*. 46(3):81–85. Mar. 2003.

[Go97] Goldwasser, S.: Multi party computations: Past and present. In: *Proc. 16th ACM Symp. Principles of Distributed Computing*. pp. 1–6. 1997.

[Go02] Goldreich, O. Secure multi-party computation. Internet: `http://www.wisdom.weizmann.ac.il/~oded/pp.html`. 2002.

[Ja92] Jacob, J.: Basic theorems about security. *J. Computer Security*. 1(4):385–411. 1992.

[KM99] Krings, A. W. und McQueen, M. A.: A Byzantine resilient approach to network security. In: *Digest of FastAbstracts 29th Int. Symp. Fault-Tolerant Computing (FTCS-29)*. Madison, Wisconsin. Jun. 1999. `http://www.crhc.uiuc.edu/FTCS-29/pdfs/krings.pdf`.

[Ku02] Kursawe, K.: Asynchronous Byzantine group communication. In: *Proc. 21st IEEE Symp. Reliable Distributed Systems (SRDS), Workshop on Reliable Peer-to-Peer Distributed Systems*. pp. 352–357. Osaka, Japan. Oct. 2002.

[LSP82] Lamport, L., Shostak, R., und Pease, M.: The Byzantine generals problem. *ACM Trans. Progr. Lang. and Sys.*. 4(3):382–401. Jul. 1982.

[Me98] Meadows, C.: A formal framework and evaluation method for network denial of service. In: *Proc. 1999 IEEE Computer Security Foundations Workshop*. pp. 4–13. 1998.

[Po92] Powell, D.: Failure mode assumptions and assumption coverage. In: *Proc. 22nd Int. Symp. Fault-Tolerant Computing (FTCS '92)*. pp. 386–395. Boston, MA. Jul. 1992.

[PSL80] Pease, M., Shostak, R., und Lamport, L.: Reaching agreements in the presence of faults. *J. ACM*. 27(2):228–234. Apr. 1980.

[Re85] Reischuk, R.: A new solution to the Byzantine generals problem. *Information and Control*. pp. 23–42. 1985.

[Re96] Reiter, M. K.: A secure group membership protocol. *IEEE Trans. Softw. Eng.*. 22(1):31–41. Jan. 1996.

[Sc99] Schneier, B.: Attack trees. *Dr. Dobb's Journal of Software Tools*. 24(12):21–22, 24, 26, 28–29. Dec. 1999.

[Sc01] Schneier, B.: A cyber UL? In: *Crypto-Gram Newsletter*. Counterpane Internet Security, Inc. Jan. 2001. `http://www.counterpane.com`.

[Th03] The SANS Institute. Internet storm center. Internet: `http://isc.incidents.org/`. 2003.

[Un00] Underwriters Laboratory. Standard 687 for burglary-resistant safes. `http://www.ul.com/`. Aug. 2000.

[Va93] Vaidya, N. H.: Degradable agreement with hybrid faults (an algorithm and reliability-safety analysis). Tech. Rep. 93-037. Dept. Comp. Science, Texas A&M Univ. Aug. 1993.

[WLG$^+$78] Wensley, J. H., Lamport, L., Goldberg, J., Green, M. W., Levitt, K. N., Melliar-Smith, P. M., Shostak, R. E., und Weinstock, C. B.: SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proc. IEEE*. 66(10):1240–1255. Oct. 1978.

[YMV$^+$02] Yin, J., Martin, J.-P., Venkataramani, A., Alvisi, L., und Dahlin, M.: Byzantine fault-tolerant confidentiality. In: *Proc. Int. Workshop Future Directions in Distributed Computing*. pp. 12–15. Jun. 2002.

[ZSvR02] Zhou, L., Schneider, F. B., und van Renesse, R.: COCA: A secure distributed on-line certification authority. *ACM Trans. Comp. Sys.*. 20(4):329–368. Nov. 2002.