A Critical Analysis of Models for Fault-Tolerant and Secure Computation

Mike Burmester* Yvo Desmedt*
Department of Computer Science
Florida State University
E-mail: {burmester, desmedt}@cs.fsu.edu

Yongge Wang
Department of Software and Information Systems
University of North Carolina at Charlotte
E-mail: yonwang@uncc.edu

ABSTRACT
We consider the problem of fault-tolerant dependable computation with multiple inputs. Although the traditional model assumes that the number of faults is relatively small when the enemy has limited resources, this assumption is unreasonable when some faults may be interdependent. Indeed, a computation system may have several replicated components and the adversary may exploit a common weakness of these so as to cause simultaneous failure. In this paper we introduce models for secure distributed computation with multiple inputs that tolerate dependent faults. In particular, we consider AND/OR graphs with colored vertices and show that they can be used to model dependable computations with an appropriate level of abstraction. We then apply this model to show that fault-tolerant dependable computation with multiple inputs can only be achieved if an appropriate number of color-disjoint solution subgraphs is found. Finding such subgraphs is **NP**-hard. This is in contrast to the problem of dependable computation with single inputs that requires finding vertex-disjoint paths, for which there is a polynomial time algorithm.

KEY WORDS
Dependable computation, fault tolerance, security

1 Introduction

Redundancy (see for example, [4, 5]) has been used to achieve reliability in the context of fault-tolerant computation, reliable communication, and reliable networks. While reliability is solely concerned with accidental errors, survivability must deal with malicious faults (see for example, [12, 13]). A redundant computation system consists, essentially, of several linked processors, such as servers, hardware units, software programs, etc. A basic requirement is that the system should tolerate faults. Several models can be used for such systems, but these usually deal with faults that follow a random pattern and are not malicious. It is not clear whether redundancy is sufficient in itself to guarantee that a computation will be secure (even if conventional cryptographic tools are used), and therefore it is not clear whether these models are adequate to describe a malicious scenario in its generality.

*This research supported by the National Science Foundation under grant CCR-0209092.

In this paper we shall consider models for redundant computation that will tolerate random faults and malicious faults, using the traditional setting of computation theory. These are high level models which allow for the most general settings appropriate for such computations. The specific nature of the processors involved, or the nature of their links is abstracted out. For example, the processors might be servers, software modules, or even steel plants. A hardware module might be linked to a software package, and so on. These aspects will not concern us.

2 Background

In the context of survivable computer systems one can distinguish two types of malicious faults: *independent* and *dependent* faults. As an illustration consider a redundant computation system for which the hardware and software components have been developed independently. For such a system, a fault in one of its components will not necessarily extend to the rest of the system. That is, independent faults are (usually) restricted to subsystems. On the other hand if software components have been replicated, then (malicious) faults may be duplicated. Such faults are dependent.

2.1 Independent faults

In the context of reliability, faults occur in a probabilistic way and are independent of each other and of the overall state of the system at their origin. A typical such fault in a circuit occurs when the output of a gate is independent of its input, for example when its output is a random string (noise). The impact of such faults can be controlled by using redundancy. However such an approach cannot deal with strongly dependent faults.

2.2 Dependent faults

In the traditional faults model, the usual scenario is that the adversary controls the faulty processors according to some plan which may exploit the possible weakness of the system. The adversary has at least as much power and knowledge about the state of the system (excluding the secret keys) as the non-faulty processors, and possibly more. For example, the adversary may know the structure of the system whereas the non-faulty processors may not. Such

an adversary might try to use this information and forward misleading inputs to non-faulty processors in an attempt to cause the system to fail.

It is reasonable to assume when dealing with an attack with limited resources, that the number of faults will also be limited. However this assumption is unreasonable for a scenario in which the faults are strongly dependent. In particular, when the same faulty software has been replicated, and the same component is used throughout the system (the 1988 Morris Internet Worm exploited the vulnerabilities in UNIX).

2.3 The requirements for a model

Achieving processor cooperation in the presence of faults is a major problem with distributed systems. Popular paradigms such as Byzantine agreement (see for example, [10]) have been studied extensively. Dolev [4] (see also, Dolev et al. [5]) has shown that Byzantine agreement is achievable only when the number of faulty processors in the system is less than one-half of the connectivity of the system's network (for Byzantine agreement, the number of faulty processors must be less than a third of the total number of processors). Hadzilacos [7] has shown that even in the absence of malicious failures $(k+1)$-connectivity is required for agreement in the presence of k faulty processors. These results refer to processors with one type of input, while in practice most processors have several types of input (see Section 3.2.1 for details).

Modelling a scenario in which the adversary is malicious should allow for a dynamic topology in which changes in the system may take place without the (non-faulty) processors being aware of these. It should allow for the most general type of processor. This could be a simple gate, a software package, or even a powerful computer. It should also allow for memory, and the ability to perform complicated operations. The model should describe the structure of the system at the appropriate level of abstraction: it must distinguish those aspects that are relevant to the computation and abstract out the aspects that are not essential. Such a model should offer the maximum flexibility to the designer.

3 Current models

3.1 Models for fault-tolerant communication

Several models have been proposed in the literature for authenticated communication in large-scale open systems (see for example, Reiter and Stubblebine [17]). In this section we overview these models. Each of these makes use of a directed graph to capture the notion of "certification".

We shall use the term *entity* to indicate someone or something that possesses and makes use of a private/public key pair. This could be a person, an authentication server, or a certification authority. The *user* is the person applying the model for the purpose of gaining assurance in the name-to-key binding.

The Beth-Borcherding-Klein model [1] This uses a set of trust relationships that can be represented by a directed graph whose vertices are the entities. There are two types of edges: "directed edges" denoted by $A \rightarrow B$ and "recommendation edges" denoted by $A \Rightarrow B$. Edge $A \rightarrow B$ indicates that entity A believes it can authenticate (i.e., it has the public key of) entity B. Edge $A \Rightarrow B$ indicates that entity A trusts entity B to authenticate other entities, or recommend other entities to authenticate, or further recommend.

The Mauer model [11] This also uses a directed graph. As with the Beth-Borcherding-Klein model, the vertices are entities and there are two types of edges, "directed" and "recommendation". However, there is a subtle difference in the semantics for these edges. In the Mauer model, the edges represent syntactic constructs, e.g., certificates. In particular, the directed edge $A \rightarrow B$ indicates that user A "holds a certificate for user B's public key (allegedly) issued and signed by A", while a recommendation edge $A \Rightarrow B$ indicates that the user A possesses a recommendation (recommending or authenticating) for B (allegedly) signed by entity A.

The Reiter-Stubblebine model [16] This model also uses a directed graph, but again differs from the Beth-Borcherding-Klein model and the Mauer model. In this case, the vertices are public keys (actual keys, with no references to any entities), and an edge $K_1 \rightarrow K_2$ indicates that the user has a certificate signed with the private key corresponding to K_1 (so K_1 can be used to verify the signature) that assigns attributes to K_2. These attributes are assumed to assert K_2's owner (among other things, perhaps) and are included as a label for edge $K_1 \rightarrow K_2$.

The Zimmermann model [21] This is used in PGP [21]. Zimmermann's model resembles (but precedes) the Reiter-Stubblebine model. Its nodes are keys, and its edges $K_1 \rightarrow K_2$ are labeled by attributes and are represented by a certificate that binds the attributes to K_2 and that can be verified with the key K_1. The graph differs from the Reiter-Stubblebine graph in that the user can assign to each vertex a *trust value* in the set $\{$ *unknown*, *untrusted*, *marginally trusted*, *fully trusted* $\}$.

The Franklin-Yung model [6] This uses hypergraphs (a kind of special directed graphs) to model communications in broadcast channels. In this model, the message of the sender is received by all receivers designated by the sender with authenticity and privacy.

3.2 Models for fault-tolerant computation

Our models are based on the traditional setting of computation theory. In particular all the entities and the adversary have limited resources (polynomially bounded in the description of the application).

There are several models that can be used for fault-tolerant computation systems. However as we shall see from the analysis below, these are not adequate to describe fault-tolerant computation systems with multiple inputs. The theory of computing, following Turing, is based on representing data by sequences of symbols, typically bits, and performing operations selected from a small set

of primitive operations (such as ANDs, ORs, etc). Systems which use such primitives can be modelled by circuits. This is the classical model for fault-tolerant (VLSI) design with redundancy (e.g., von Neuman restoring). Circuits are best suited for Boolean type computations in which the structure of the system is well defined, and usually are not suitable for open settings. A somehow similar argument applies to models based on finite state machines. These models are not appropriate for distributed systems, and therefore are not suitable for our purpose.

In programming languages, a function or a computation process is modelled by a flow graph. This model allows one to characterize a computation with multiple inputs at an appropriate level of abstraction, although it does not model the redundancy in fault-tolerant computations. We can model a redundant computation system with one type of input (see for example [15]) by a directed graph (or multi-graph) in which the vertices are the processors and the edges are the links. Incoming edges of a vertex indicate which units can be used as alternative servers to provide the input. This is a high level model and appropriate for some applications, for example networks applications. However it does not describe scenarios in which most computation processes have more than one type of input.

3.2.1 Multiple inputs aspects

Our analysis above shows that none of the existing models is appropriate for fault-tolerant computation system with multiple inputs. Such systems include most of the national infrastructures whose components require computations with multiple inputs. To illustrate this let us consider some examples.

In an electrical power distribution system, the generation stations, the transmission and distribution networks must be integrated, and all the components must be reliable so as to achieve and maintain nominal functionality for the end-user. Furthermore, the generation stations are themselves systems that consist of components with multiple inputs, e.g. the control systems of a generation station need data from several sensors. Another example is an air traffic control system. This also consists of several components with multiple inputs. For example, the processors of an aviation control system need data from several sources such as the speed of the airplane, its position, etc. There are many more examples of such applications. With most of these, we often need the output of more than one sensor as an input to a computation. In the next section, we will present several models for such kinds of systems.

We have given a brief survey of models used in the context of fault-tolerance and survivability. However as we have seen, none of these are sufficient to model fault-tolerant and secure distributed computation with multiple inputs. The models for fault-tolerant communications do not capture the multiple inputs aspect, while the the models for fault-tolerant computations do not capture the redundancy aspects. In the following section we shall describe models for fault-tolerant and secure distributed computations with multiple inputs.

4 Models for independent faults

In this section, we discuss models for independent faults.

4.1 A directed multi-graph with colored edges model

We model a redundant computation system with multiple inputs by a directed multi-graph with colored edges. The different colors of the edges indicate the different types of inputs. The graph must have at least one input vertex and one output vertex. There are several possible applications for this model. For example, subroutines whose inputs have the same color need only use one input (when there are no faults). If the colors are different then the processor must use one input for each of the input colors to carry out its computation (or whatever it is supposed to do).

Definition 4.1 *A* directed multi-graph with colored edges $G(V, INPUT, output; E; COLORS)$ *is a directed graph with a set of* vertices V*, a set* $INPUT$ *of* input vertices*, one* output vertex*, and a set of* colored directed edges E*. The input vertices have no incoming edges and the output vertex has no outgoing edges.*

This model is equivalent to the AND/OR graph model introduced by Burmester, Desmedt, and Wang [2], which we shall now describe.

4.2 An AND/OR graph model

AND/OR graphs have been used to model problem solving processes in artificial intelligence (see, e.g., Nilsson [14]). In [2], Burmester, Desmedt, and Wang used AND/OR graphs to model fault-tolerant computations with multiple inputs. An *AND/OR graph* is a directed graph with two types of vertices, labelled $\wedge$*-vertices* and $\vee$*-vertices*. The graph must have at least one input (source) vertex and one output (sink) vertex. The output vertex may be regarded as an $\vee$-vertex (without loss of generality). More specifically, we have the following definition.

Definition 4.2 [2] *An* AND/OR graph $G(V_\wedge, V_\vee, INPUT, output; E)$ *is a directed graph with a set* $V_\wedge$ *of* $\wedge$-vertices*, a set* $V_\vee$ *of* $\vee$-vertices*, a set* $INPUT$ *of* input vertices*, an* output vertex $output \in V_\vee$*, and a set of* directed edges E*. The vertices with no incoming edges are input vertices and the vertex with no outgoing edges is the output vertex.*

It can be shown that this model is equivalent to the directed graph model with multi-colored edges considered in Section 4.1, in the sense that there is a polynomial time reduction from one to the other. The application given in Section 4.1 can also be used for this model. In this case, for processors which need all their inputs in order to operate are represented by $\wedge$-vertices, whereas processors which can choose (using some kind of voting procedure) one of their "redundant" inputs are represented by $\vee$-vertices. In

the replicated agent computation with voting (see Schneider [18]), a **meet** operation may be considered as an $\wedge$-vertex and the operation of choosing one agent from its replicas may be considered as an $\vee$-vertex.

5 Models for dependent faults

Current computing platforms, as well as communications infrastructure and software, are largely homogeneous. Computing platforms are quite uniform in the operating system they run and the instruction-set architecture they support. Furthermore, display, network interface, and disks are made uniform by adherence to either government or manufactures standards or are presented to application software as common interfaces by operating systems software in the form of device drivers and hardware adaptation layers. The similarity of the interfaces of a collection of homogeneous systems implies that these systems will share vulnerabilities. A successful attack on one system is then likely to succeed on other systems as well. This picture is further complicated by the role of technical standards. Standards enable interoperability of components, and attacks that exploit the vulnerability of the components can be reused in a variety of settings where standards prevail. The models in the previous section do not reflect these finer aspects of dependent faults.

5.1 An AND/OR graph with colored vertices

We can model a redundant computation system with dependent faults by an AND/OR graph with colored vertices.

Definition 5.1 *An* AND/OR graph with colored vertices *is an array:* $G(V_\wedge, V_\vee, INPUT, output; E; COLORS, C)$ *with the following properties:*

1. $G(V_\wedge, V_\vee, INPUT, output; E)$ *is an AND/OR graph;*
2. $COLORS$ *is a set of colors;*
3. $C : V_\wedge \cup V_\vee \rightarrow COLORS$ *is a function which assigns a color for each vertex.*

The main advantage of AND/OR graphs with colored vertices is that they model the dependent faults at an appropriate level of abstraction and are a powerful mathematical tool for the study of dependent faults.

They are several possible applications for this model. For example, processors with the same standards can be marked with the same color. We illustrate this by an example of a simplified version of an aviation control system. Assume that four computers A, B, C, and D are used to process data from the speed sensors of an airplane, and four computers E, F, G, and H are used to process data from the position sensors of the airplane. The results from the eight computers are fed to the computer I to compute the next position of the airplane. In addition, assume that A, B, E, and F use Intel Pentium 3 processors and run Microsoft Window 2000, C, D, G, and H use Pentium 4 processors and run Microsoft Window XP, and I is a Macintosh computer which runs MacOS. The aviation control system can be modelled by the AND/OR graph in Figure 1. In this graph, S and P represent the speed sensor and the position sensor respectively. The data from the computers A, B, C, and D is fed to one $\vee$-vertex and the data from the computers E, F, G, and H is fed to another $\vee$-vertex. The colors of the vertices are as follows: $C(A) = C(B) = C(E) = C(F) = \{Pentium3, win2000\}$, $C(C) = C(D) = C(G) = C(H) = \{Pentium4, winXP\}$, and $C(I) = \{M68000, MacOS\}$. (The colors of the $\vee$-vertices, the vertex S, and the vertex P are defined according to the systems used.) When a vertex fails due to one of its colors, then all vertices with the same color will fail. In a fault-tolerant computation system, all vertices with the same colors will have the same failure probability. Indeed, an operating system is a collection of correlated and independent components. One component failure does not necessarily mean that all other components will fail at the same time.

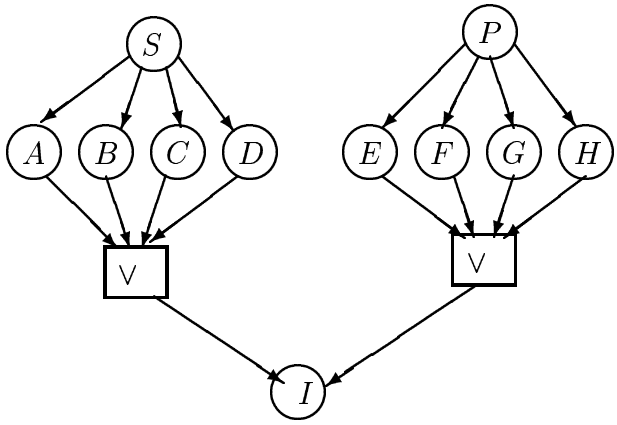

Figure 1. The aviation control system AND/OR graph

5.2 An AND/OR graph with a partial order on the colors of the vertices

The AND/OR graphs with colored vertices model dependent faults in a natural way. However they do not describe some of the finer aspects of dependent faults, such as weak dependencies. In the previous model we identified different types of vertices by a color. A color could correspond to an operating system, or to the microprocessor used, etc. The operating system could be replicated, with different replications corresponding to different vertices. In many instances there is a hierarchy for the type of failures. For example, if the hardware of a computer has a design flaw, all operating systems that use that hardware may also fail. Furthermore, if the operating system fails, all application programs requiring that operating system will fail. So one has an hierarchy of types of vertices. This additional aspect can (more generally) be expressed by using a partial order on the colors. We model such a redundant computation system by an AND/OR graph with colored vertices which in addition has a partial order on the colors.

Definition 5.2 *An* AND/OR graph with a partial or-

der on the colors *is an array:* $G(V_\wedge, V_\vee, INPUT, output; E; COLORS, C; PO)$ *for which:*

1. $G(V_\wedge, V_\vee, INPUT, output; E; COLORS, C)$ *is an AND/OR graph with colored vertices;*
2. PO *is a partial order (or a lattice structure) on* $COLORS$.

For example, the IE6.0 for Windows XP must run on a computer with Windows 2000 or Windows XP as one of its operating system. When an AND/OR graph is used to model computer systems, the IE6.0 for Windows XP should be assigned a color whose order is less than that of the color for Windows XP and that of the color for Windows 2000. The applications given in Section 5.1 can also be used for this model. For example, in the AND/OR graph of Figure 1, we may define a partial order on the colors as follows: $win2000 \leq Pentium3$ and $winXP \leq Pentium4$. Note that generally we do not have $win2000 \leq winXP$, since Window XP may have bugs that Window 2000 does not have. In this case, when a vertex with a certain color fails, all vertices with colors "lower" than the failed processor's color also fail. Indeed, all models introduced in the previous sections are equivalent in the sense that there is a polynomial time algorithm that transforms one to the other.

As an example of this equivalence we show how an AND/OR graph with colored vertices can be simulated by an AND/OR graph. Let $G(V_\wedge, V_\vee, INPUT, output; E; COLORS, C)$ be an AND/OR graph with colored vertices. It is straightforward to show that the following AND/OR graph G' is functionally equivalent to the AND/OR graph G with colored vertices. Define $G'(V'_\wedge, V'_\vee, INPUT', output; E')$ by letting

1. $INPUT' = INPUT \cup \{I_{color}\}$;
2. $V'_\vee = V_\vee \cup \{v_c : c \in COLORS\}$;
3. $V'_\wedge = V_\wedge \cup \{v' : v \in (V_\vee \cup V_\wedge) \setminus (INPUT \cup \{output\})\}$;
4. E' consists of the following edges:
   - For each $c \in COLORS$, there is an edge $I_{color} \to v_c$;
   - For each vertex $v \in V_\vee \cup V_\wedge$ such that $C(v) = c$, there is an edge $v_c \to v'$;
   - For each vertex $v \in (INPUT \cup \{output\})$, there is an edge $v \to v'$;
   - For each edge $(u \to v) \in E$, there is an edge in E': $u' \to v$.

6 Models for general fault structures

More generally, the faults of a dependable system can be modelled by an *adversary structure* $\mathcal{Z}$ [8, 9]. This structure consists of a monotone family of subsets of the set of components of the system. Formally, if X is the set of components, then "$\mathcal{Z} \subset 2^X$ such that, $Z \in \mathcal{Z} \wedge Z' \subset Z \Rightarrow Z' \in \mathcal{Z}$". If a graph is used to model the system then $X = V$ is the node set (for multyparty computation systems X is the set of parties, or "players"). The fault sets Z of $\mathcal{Z}$ are precisely those sets of components that the adversary can corrupt.

For the particular case AND/OR graphs with colored vertices, the fault sets Z consist of nodes that have the same color. For AND/OR graphs with an order on the colors of their vertices, the fault sets consist consist of nodes of the same color, and nodes of a lower color. In general, the adversary structure model can be used to describe the faults of any colored graph (in this way). The converse is also true if we allow for multicolored nodes (including nodes with no color).

7 Applications

Redundancy plays a key role in achieving reliability. It is used for reliable communications (with error-correcting codes), for network reliability and for fault-tolerant computation. Assume that we use an AND/OR graph to model a fault-tolerant computation. Then information (for example, mobile codes) flows from the input vertices of the graph to the output vertex. A valid computation in an AND/OR graph can be described by a *solution graph* (the exact definition will be given below). However, if insider vertices are faulty or even malicious, then the output vertex cannot be certain that the result will be authentic or correct. If we assume that at maximum k vertices will be malicious, then the theory of fault-tolerant computation (see, e.g., [4, 5]) tells us that, if there are $(2k + 1)$ vertex-disjoint paths of information flow then the output vertex will always succeed in getting at least $(k + 1)$ identical results computed from the input vertices through vertex-disjoint solution graphs, provided that the output knows the layout of the graph. This implies that if output vertex knows the layout of the graph then it can use a majority vote to decide whether the result is correct or not. It follows that in order to achieve dependable computation with redundancy, it is necessary to find an appropriate number of vertex-disjoint solution graphs in a given AND/OR graph. A path in the traditional sense is not useful in our contexts, because we need all the inputs of an $\wedge$-vertex to carry out the computation. We therefore give a formal definition of solution graphs for AND/OR graphs.

Definition 7.1 *Let* $G(V_\wedge, V_\vee, INPUT, output; E; COLORS, C; PO)$ *be an AND/OR graph with a partial order on its colored vertices. A* solution graph $P = (V_P, E_P)$ *is a minimal subgraph of* G *with:*

1. $output \in V_P$.
2. *For each* $\wedge$*-vertex* $v \in V_P$*, all incoming edges of* v *in* E *belong to* E_P.
3. *For each* $\vee$*-vertex* $v \in V_P$*, there is exactly one incoming edge of* v *in* E_P.

4. *There is a sequence of vertices* $v_1, \ldots, v_n \in V_P$ *such that* $v_1 \in INPUT, v_n = output$, *and* $(v_i \rightarrow v_{i+1}) \in E_P$ *for each* $i < n$.

Moreover, two solution graphs P_1 *and* P_2 *are* color disjoint *if* $(C(V_{P_1}) \cap C(V_{P_2})) \subseteq C(INPUT \cup \{output\})$.

Note that every vertex of a solution graph may have some outgoing edges due to the minimality of a solution graph. Since an AND/OR graph without $\wedge$-vertices may be considered as a normal digraph, the solution graphs in such AND/OR graphs are the same as the standard paths in digraphs. It is easy to see that there is only one solution graph in an AND/OR graph that has only one $\vee$-vertex (that is, the output vertex) and in which the $output$ vertex has only one incoming edge.

Lemma 7.2 *If two solution graphs* P_1 *and* P_2, *in an AND/OR graph with colored vertices, are* color disjoint, *then there are vertex-disjoint, that is* $((V_{P_1}) \cap C(V_{P_2})) \subseteq (INPUT \cup \{output\})$.

Proof. This follows directly from the definitions. Q.E.D

Though there are polynomial time algorithms for finding vertex-disjoint paths in networks of processors, the corresponding problem for computation systems with multiple inputs is **NP**-hard.

Theorem 7.3 *Given an AND/OR graph* G *with a partial order on its colored vertices and a positive number* k, *it is* **NP***-hard to decide whether there are* k *color-disjoint solution graphs in* G.

Proof. The results in [2] show that it is **NP**-hard to find a vertex separator of a given AND/OR graph. This implies that it is **NP**-hard to decide whether there are k vertex-disjoint solution graphs in G. The result follows by observing that, a set of k color-disjoint solution graphs in G is also a set of k vertex-disjoint solution graphs. Q.E.D.

We foresee other applications. For example, these models may be used to identify the most critical tasks in redundant computations and to allocate the available resources to the most critical tasks. These models may also be used to analyze the flows in computation systems with multiple inputs and may eventually be used to analyze the performance of a manufacturing system.

References

[1] T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. In: *ESORICS 94, LNCS 875*, pp. 236–274, Springer Verlag, 1995.

[2] M. Burmester, Y. Desmedt and Y. Wang. Using Approximation hardness to achieve dependable computation. In: *Proc. 2nd RAND, LNCS 1518*, pp. 172–186, 1998.

[3] Y. Desmedt, Y. Wang, and M. Burmester. Complete Characterization of Adversaries Tolerable in Secure Message Transmissions. To appear.

[4] D. Dolev. The Byzantine generals strike again. *J. of Algorithms*, **3**, pp. 14–30, 1982.

[5] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *J. of the ACM*, **40**(1), pp. 17–47, 1993.

[6] M. Franklin and M. Yung. Secure hypergraphs: privacy from partial broadcast. In: *Proc. ACM STOC, '95*, pages 36–44, ACM Press, 1995.

[7] V. Hadzilacos. *Issues of Fault Tolerance in Concurrent Computations*. PhD thesis, Harvard University, Cambridge, MA, 1984.

[8] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. In *Proc. of the 16th ACM PODC*, 1997.

[9] M. Hirt and U. Maurer. Player Simulation and General Adversary Structures in Perfect Multiparty Computation. Journal of Cryptology 13(1): 31-60 (2000)

[10] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *J. of the ACM*, **32**(2), pp. 374–382, 1982.

[11] U. Mauer. Modeling a public-key infrastructure. In: *Computer Security—ESORICS 96, Lecture Notes in Computer Science 1146*, Springer Verlag, 1996.

[12] P. G. Neumann. Are dependable systems feasible? *Commun. of the ACM*, **36**(2), pp. 146, 1993.

[13] V. F. Nicola, M. K. Nakayama, P. Heidelberger, and A. Goyal. Fast simulation of highly dependable systems with general failure and repair processes. *IEEE Trans. on Computers*, **42**(12):1440-1452, 1993.

[14] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.

[15] D. Pradhan. *Fault-Tolerant Computing : Theory And Techniques*. Prentice Hall, 1986.

[16] M. Reiter and S. Stubblebine. Path independence for authentication in large-scale systems. In: *Proc. 4th ACM Conference on Computer and Communication Security*, ACM Press, 1997.

[17] M. Reiter and S. Stubblebine. Towards acceptable metrics of authentication. In: *Proc. 1997 IEEE Symposium on Security and Privacy*, IEEE Press, 1997.

[18] F.B. Schneider. Towards fault-tolerant and secure agentry. In: *Proc. WDAG '97, LNCS 1320*, pp. 1–14, Springer Verlag, 1997.

[19] Y. Wang and Y. Desmedt. Secure broadcast in broadcast channels. In: *Eurocrypt'99*, pp. 443–455, LNCS 1592.

[20] Y. Wang, Y. Desmedt, and M. Burmester. Models for dependable computation with multiple inputs and some hardness results. *Fundamenta Informaticae*, **42**(1):61–73, 2000.

[21] P. Zimmermann. *PGP User's Guide*, Volumes I and II, 1994. Included in the PGP 8.0 distribution.