

Secure Access to Medical Data over the Internet

Ulrich Ultes-Nitsche

Department of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, United Kingdom
uun@ecs.soton.ac.uk

Stephanie Teufel

International Institute of Management in Telecommunications
University of Fribourg
Avenue de Tivoli 3, CH-1700 Fribourg, Switzerland
stephanie.teufel@unifr.ch

Abstract The concept of context-dependent access control has emerged during the last years: Information about the state of a process model of a working environment is combined with general knowledge about a person to grant or revoke access to protected data. Being understood very well in principle, different problems arise when implementing context-dependent access control, in particular on an open network.

This paper reports on an ongoing project on context-dependent access control to support distributed clinical trials. Centrally stored data will be accessed from contributors to the clinical trial over the Internet. We present in this paper how context-dependent access control can be implemented on the Internet in a secure way. Technically we use Java Servlets to implement the access control and SSL to secure communication. The whole framework is built around the Java Webserver. We emphasize the technical aspects of this scenario in this paper.

Keywords Workflow-support for clinical trials, telemedicine, context-dependent access controls, health information systems.

I. INTRODUCTION

In [8] and previous papers [6,7], the concept of a context-dependent access-control has been introduced and discussed exhaustively. A prototype implementation of the concept is described in [11]. The prototype implementation is for local use only and would reveal many security holes if used over an open network: The dynamic link library (DLL) that handles the access control, for instance, would be publicly accessible. In [11], by spying out the DLL code, one could obtain information about the database's administrator log-in procedure, possibly leaving the entire database unprotected. However, using technology different from the one presented in [11] allows to come up with a secure distributed solution to context-dependent access control over the Internet.

In context-dependent access control, information about the state of a business process is combined with general knowledge about a user to grant or revoke access to sensitive data. The basic concepts of such an access control scheme are understood very well [5]. However, when developing a concrete system one faces several problems: existing access control mechanisms of the target platform have to be adapted to support context-dependency, missing features have to be realized in some indirect way, etc.

It is the aim of the current paper to present an implementation concept for context-dependent access control on the Internet. Even though applied to the specific application area of clinical trials, the underlying concepts are general and support all applications for which context-dependent access control is suitable. The paper summarizes a part of the Swiss

National Science Foundation funded project *MobiMed* [1].¹ The system we are going to describe is PC-based (Windows NT) and implemented as a Java Servlet accessing an MS SQL Server database. The Servlet extends the functionality of Java Webserver, which makes it accessible from the Internet. Context information, which is used for checking the authority of an access request, is delivered by an Action Technology workflow system.

This paper emphasizes the novel *technical* solution to accessing medical data over the Internet under a context-dependent access control policy. So we concentrate very much on describing the impact that the use of Java Servlets has. More aspects of context-dependent access controls can be found in these proceedings in [12]. After a short introduction into context-dependent access control, we present the basic implementation concepts. To give a detailed description of the implementation of context-dependent access controls, we give an overview of the components that ensure security. Finally, we discuss the appropriateness of the presented solution.

II. CONTEXT-DEPENDENT ACCESS CONTROL

Role based security approaches fit well the hierarchically structured setting of a hospital [15,16]. Each level in the hierarchy can be mapped to a so-called organizational role that a person at this hierarchical level plays in the hospital (e.g. medical personnel, care personnel, etc.). After an analysis of each role's demands on obtaining particular data to perform work, access rights are assigned to each role. The access rights determine which records in the database that contains information about the patients (patient records) may be read or written by a person playing a particular role. When logging in, a user of the system identifies her- or himself by using a chipcard and a PIN, and then a role is assigned to the person according to user/role lists, determining her or his access rights. A different way to handle role assignment is to store role information on the chipcard itself in ciphered form, which then is read during log in or, alternatively, whenever data records are accessed.

¹ *MobiMed* (Privacy and Efficiency in *Mobile Medical Systems*) is a project of the Swiss Priority Programme (SPP) for *Information and Communications Structures* (ICS; 1996-1999) of the Swiss National Science Foundation (SNSF) aiming at the development of mobile access to data in a clinical environment. Its contributing members are the Universities of Zurich, Oldenburg, and Southampton, and Plattner Schulz Partner AG, Basel.

Simple role based access control mechanisms have the advantage that they can be implemented rather easily but the drawback of certain inflexibilities. A more sophisticated access control technique refining the role based approach takes into account an access request's particular point in time, i.e. the question: "Is it reasonable that a person playing a particular role *needs* access to certain data at the current state of a health-care process?"² Obviously, it is not necessary to have access to all data about a particular patient all the time. The question of "what does one *need to know*?" gave the considered principle its name: the *need-to-know* principle. Moreover, what does one *need to know right now* (at the time of an access request) is the context-dependent access control scheme that we consider in this paper. A possible realization of the need-to-know principle can be achieved by combining user role information with state information of a workflow ("Is it reasonable to grant access to a person (playing a particular role) in the given workflow state?"). A workflow system which can be used to determine context-information is the Action Workflow approach [10].

III. THE ACCESS-CONTROL SYSTEM IN PRINCIPLE

As mentioned above, context-dependent access control combines user information with process state information to compute access rights. Since the database system that we consider --- MS SQL Server --- uses group-based access control, we have to build context dependency around group-based access controls. In group-based access control, a list of groupnames is assigned to a table or table entry of a database. A group is a name to a list of usernames. If a user requests access to a table entry, the group-based access control tries to match the username with a group assigned to the requested data for the type of access request. If the username can be matched, access is granted. In Figure 1, the group *care personnel* has access to table *Department/Room* and group *medical personnel* has access to table *Type of Examination*.

Usually, medical personnel have access to all the data that care personnel can access. By making group *medical personnel* a subgroup of group *care personnel*, *medical personnel* inherits all access rights of *care personnel*. This is sketched in Figure 2.

Subsequently, we describe how the group concept of MS SQL Server can be used to realize context dependency. The first constraint we have to consider is that group assignments in a given hospital should not be changed: The access rights to existing tables for which no context-dependent access control is established should be left unchanged. On the other hand, for new tables in the database, context-dependent access control shall be established, implying that for none of the existing groups access is granted a priori to these new tables.

Access rights in the context-dependent scheme are only given temporarily to users for a single access. We achieve this by creating a not yet existing group, called here *MobiMedDBAccessGranted*. No user and no group is

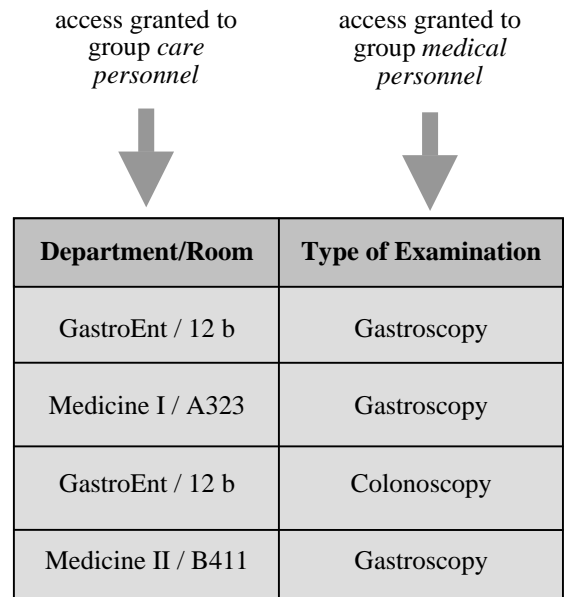


Figure 1. The group concept of MS SQL Server.

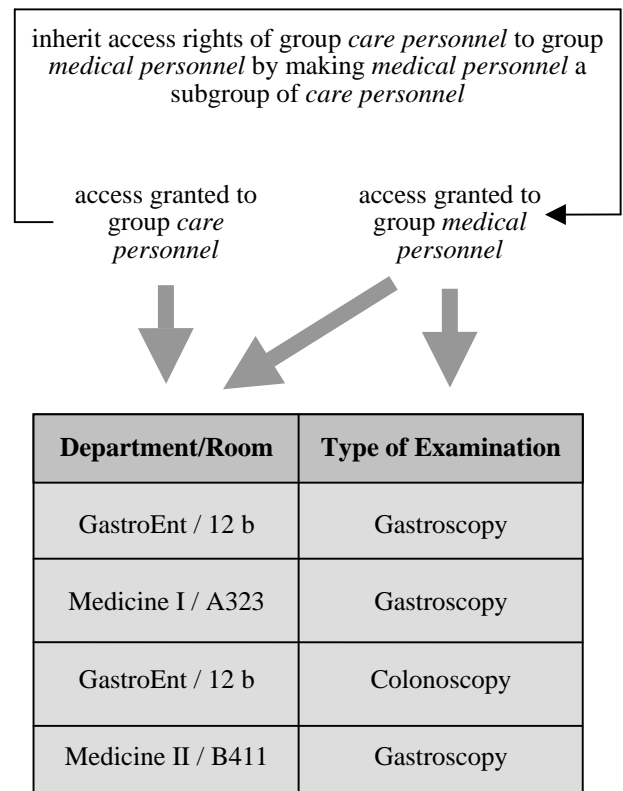


Figure 2. Inheriting access rights.

assigned to *MobiMedDBAccessGranted* initially and permanently. In order to change the group membership of a user temporarily, we can use the SQL stored procedure *sp_change_group*. After checking the *need to know* of a user to perform an access [8], he or she receives temporary membership in the group *MobiMedDBAccessGranted*, the

² A health-care process is a business process describing patient treatment in a hospital. In our context it is the process a patient goes through in a clinical trial.

access is performed, and the user is reset to her/his previous group.

As mentioned earlier, the context information that is used in the evaluation of the authority of an access request is taken from a business process model. In such a model, information about the history of events to a particular instantiation of the process is stored in a process state. In the context of a hospital, the state comprises information about treatment of a patient, and to each patient a single instantiation of the business process is created. The process ID of the instantiation is used as a unique key to all database entries related to the patient. As usual, access-control tables are generated, in the context-dependent framework containing more information than usual. The context-dependent access-control table contains quadruples *stateID*, *groupID*, *table entry*, *access type*. These entries have the meaning that in state *stateID*, a user belonging to group *groupID* may access database entry *table entry* by an access of type *access type*. The access type indicates whether full access, read-only access, or no access at all will be granted.

Based on the concept sketched in this section, we will develop subsequently a realization of the Internet context-dependent access-control that is secure and feasible. To implement it, we use Java Servlets, securing communication using SSL.

IV. JAVA SERVLETS

The past year has seen the rise of server-side Java applications, known as Java Servlets. Servlets are used to add increased functionality to Java-enabled servers in the form of small, pluggable extensions. When used in extending web servers, Servlets provide a powerful and efficient replacement for CGI and offer many significant advantages [14]. These advantages include:

A. Portability

Java Servlets are protocol and platform independent and as such are highly portable across platforms and between servers. The Servlets must conform to the well-defined Java Servlet API, which is already widely supported by many web servers.

B. Performance

Java Servlets have a more efficient life cycle than the either CGI or FastCGI scripts. Unlike CGI scripts, Servlets do not create a new process for each incoming request. Instead, Servlets are handled as separate threads within the server. At initialization, a single object instance of the Servlet is created that is generally persistent and resides in the server's memory. This persistence reduces the object creation overhead. There are significant performance improvements over CGI scripts in that there is no need to spawn a new process or invoke an interpreter [9]. The number of users able to use the system is also increased because fewer server resources are used for each user request.

C. Security

The Java language and Java Servlets have improved security over traditional CGI scripts both at the language level and at the architecture level:

Language Safety

As a language Java is type safe and handles all data types in their native format. With CGI scripts most values are treated and handled as strings which can leave the system vulnerable. For example, by putting certain character sequences in a string and passing it to a Perl script, the interpreter can be tricked into executing arbitrary and malicious commands on the server.

Java has built-in bounds checking on data types such as arrays and strings. This prevents potential hackers from crashing the program, or even the server, by overflowing buffers. For example, this can occur with CGI scripts written in C where user input is written into a character buffer of a predetermined size. If the number of input characters is larger than the size of the buffer, it causes a buffer overflow and the program will crash. This is commonly known as stack smashing.

Java has also eliminated pointers and has an automatic garbage collection mechanism, which reduces the problems associated with memory leaks and floating pointers. The absence of pointers removes the threat of attacks on the system where accesses and modifications are made to areas of server memory not belonging to the service process.

Finally, Java has a sophisticated exception handling mechanism, so unexpected data values will not cause the program to misbehave and crash the server. Instead an exception is generated which is handled and the program usually terminates neatly with a run time error [3].

Security Architecture

Java Servlets have been designed with Internet security issues in mind and mechanisms for controlling the environment in which the Servlet will run have been provided.

CGI scripts generally have fairly free access to the server's resources and badly written scripts can be a security risk. CGI scripts can compromise the security of a server by either leaking information about the host system that can be used in an attack, or by executing commands using untrusted or unchecked user arguments. Java significantly reduces these problems by providing a mechanism to restrict and monitor Servlet activity. This is known as the Servlet sandbox. The Servlet sandbox provides a controlled environment, in which the Servlet can run and uses a security manager to monitor Servlet activity and prevent unauthorized operations. There are four modes of operation that include trusted Servlets, where the Servlet has full access to the server resources, and untrusted Servlets which have limited access to the system.

JDK 1.2 is introducing an extension to its security manager, the access controller. The idea behind the access controller is to allow more fine-grained control over the resources a Servlet can access. For example, instead of allowing a Servlet to have write permission to all files in the system, write permission can be granted for only the files required by the Servlet for execution [9].

However, Java-based servers are still vulnerable to denial of service attacks where the system is bombarded with requests in order to overload the server resources. This approach invokes so many Servlet instances that all the server

resources are allocated. This can impact all the services supported by the server. However, the effects of this can be reduced by specifying an upper limit on the number of threads that can be run concurrently on the server. If all the threads are allocated, that particular service can no longer be accessed, but because the server still has resources left to allocate, the rest of the services are still available.

V. THE SECURE SOCKETS LAYER PROTOCOL

The secure sockets layer protocol (SSL) is designed to establish transport layer security with respect to the TCP/IP protocol stack. Version 3 was published as an Internet draft document [2] by the IETF (Internet Engineering Task Force). We introduce SSL briefly along the lines of [13] and motivate its usage for the MobiMed prototype.

D. The Protocol Stack

The transport layer part of SSL, the SSL record protocol, sits on top of TCP in the Internet protocol stack. It is accessed by an upper layer consisting of the hypertext transfer protocol (http) and different parts contributing to SSL: SSL handshake protocol, SSL change cipher spec protocol, and the SSL alert protocol, used to set up, negotiate, and change particular security settings used by the SSL record protocol. Schematically, the SSL architecture is presented in Figure 3.

SSL Handshake Protocol	SSL Change Cipher Spec Protocol	SSL Alert Protocol	HTTP
SSL Record Protocol			
TCP			
IP			

Figure 3. SSL within the Internet protocol stack [13].

E. Different Security Features of SSL

SSL allows for different security features being chosen. First of all, different encryption algorithms can be used to produce ciphertexts and authentication messages. For authentication, different hash algorithms can be negotiated. SSL can also use X509.v3 peer certification [3]. Using a session identifier, active states of SSL are identified, where a state consists of a number of keys involved in the session, both on the server and on the client side, and sequence numbers to count the messages exchanged. By using these different parameters, SSL sets up a session configuration that then allows for ensuring integrity, confidentiality, and authentication depending on the set up parameters.

F. Use of SSL in MobiMed

Unlike other concepts that secure connections or even only data-packages, SSL includes the concept of a secure session, determined by the parameters mentioned in the subsection above. It is this session concept that makes it appealing for being used in MobiMed. The secure session lasts as long as a user is logged in the system. Since communication with the user will be based on HTML documents sent to and received from the client side using http, the use of SSL will be transparent to the client.

VI. REALIZING THE ACCESS-CONTROL SYSTEM

To perform the SQL stored procedure `sp_change_group`, one needs to have the access rights of the SQL Server's administrator. These access rights could not be granted if the context-dependent access control itself were implemented as an SQL stored procedure due to SQL Server restrictions. Hence we decided to implement it in a Java Servlet. The Servlet offers the only way to access the MobiMedDB database and can be accessed from *any* application on the Internet. The administrator log-in procedure to the SQL Server must be known to the Servlet. By putting it into the private part of the Servlet class, it is securely protected, since this part, by no means, may be sent to the client side. We overcome by this the problem of spying out administrator log-in information, a problem that existed in the local-version prototype of this system [11].

Java Servlets support the *Java DataBase Connectivity* (JDBC) API³ to access databases that support the JDBC API. SQL Server supports the *Open DataBase Connectivity* (ODBC) API that can be linked to the JDBC API. The interfaces within the resulting system are presented in Figure 4.

The client-side application (either HTML or a Java Applet) talks to the Servlet via the Java Webserver. The connection between the client and the webserver is secured using SSL. The Servlet offers to the client application the standard Servlet API. This is implemented by extending the *HttpServlet* class. The Servlet API offers to handle http *get* and *post* requests (the methods of the *HttpServlet* class are *doGet* and *doPost*). The parameters sent to the Servlet are in our case SQL queries that are interpreted by the Servlet. As presented in Figure 4, the Servlet accesses the SQL Server database using the JDBC/ODBC link. From the SQL Server it receives user information as well as the access-control tables for the evaluation of access rights. The Servlet handles the log-in procedure for a user as well as setting and resetting her/him temporarily to the *MobiMedDBAccessGranted* group, enabling her/him to access data under the context-dependent access control scheme.

An example of a successful request under the context-dependent access control scheme is described in seven steps in Figure 5 (AW Administrator is the run-time environment of the workflow system):

³ API = Application Programming Interface.

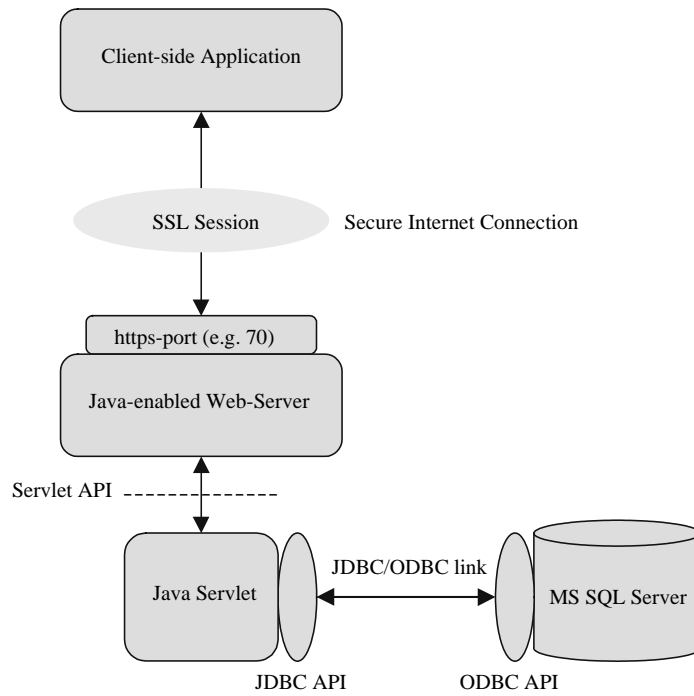


Figure 4. The system interfaces.

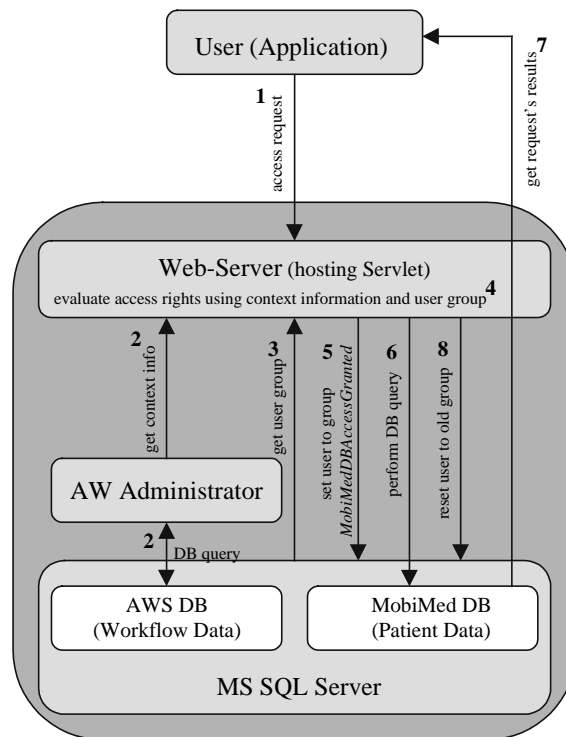


Figure 5. How are access rights assigned to a user?

1. The user (application) (started by a user or triggered by the workflow system) sends an access request to the Servlet.
2. The Servlet requests and receives context information about the workflow of the patient whose record is to be accessed.
3. The Servlet requests and receives group information about the user who is requesting the access.
4. Both, the context information and the role information are combined and the authority of the user to perform the access to the patient record is evaluated (using the access-control table). In the following it is assumed that the user has authority to perform the request (otherwise the request would be rejected at this point).
5. The user is set temporarily to group *MobiMedDBAccessGranted*.
6. The request (query) is performed on the database *MobiMedDB*.
7. The results of the query are delivered to the user.
8. The Servlet resets the user to his/her previous group membership.

It is important to note that by no means is it possible for a user to access the database directly, since she/he does not belong to the group *MobiMedDBAccessGranted*.

In the first stage of the prototype implementation, user communication is handled by means of the Servlet exchanging HTML documents with the browser of the user. In a later stage this will be refined by Servlet/Applet communication, allowing for more sophisticated user interfaces and more flexible applications.

VII. DISCUSSION

We discuss in this section the security as well as the feasibility of the presented approach. To ensure privacy, the SQL Server and the Java Webserver, including the access-control Servlet, must be a single system. This means that, if the database server and the webserver are not running on the same machine, they must at least be connected by an Intranet that does not allow outside access. This is necessary, since webserver and database server communicate using JDBC/ODBC which is not secure if the exchange of data can be seen from outside. The system is implemented in such a way that the Java Webserver is the only component that can be accessed remotely, including the Servlet it is hosting. Data exchange between client and server is held confidential by using SSL. The use of SSL has the additional benefit of supporting authentication of client and server. The client does not risk giving password information to a masqueraded server.

Given that the access-control system is implemented as described, the obvious major threat to the system is that the administrator log-in procedure known to the Servlet becomes publicly accessible. By hosting the sensitive parts of the access control in a private method of the Servlet and taking into account that Servlets are server-sided Java bytecode, neither can a user access the private Servlet methods nor is

bytecode containing sensitive information accessible from the Internet, keeping this critical part of the system secure.

A way a user could try to break the system is by sending multiple access requests, knowing that at least for some of them she/he has authority. The idea is that a request for which she/he has no authority coincides sufficiently in time with a request for which she/he has authority. So both queries could be send to the SQL Server at the time when she/he is temporarily assigned to the group for which access is granted. This situation can only occur, when multiple threads of the access-control Servlet are not synchronized properly. The easiest attempt to tackle this problem is to disable multi-threading for single users.

Finally, is the presented approach is really feasible? The only problem that could arise is that the Servlet is not capable of analyzing an SQL query in order to obtain the information needed to evaluate access rights. We require that all access requests are specific to a given instantiation of the health care process, i.e. it is a specific instantiation for a particular patient. The access request can easily be analyzed to check whether it is a read-only access. Since information about the process-state with respect to a patient is provided by the system and the system's user has authenticated herself/himself at log-in time, all necessary information to perform the context-dependent access-control check is available to the system. Hence the proposed solution is indeed feasible. Since accesses to the system are relatively rare, compared to highly used Internet servers, performance aspects do not matter much. Tests with an early local prototype [11] did not show any problems with performance.

As a last comment, it should be understood that in the setting of a hospital, access to data can be vital. Therefore context-dependent access control will be equipped with simple, group-based bypass mechanisms. However, bypassing the context-dependent access control will have to be logged thoroughly to provide proof of potential misuse of the bypass mechanism.

VIII. CONCLUSION

In this paper we reported on an ongoing project on context-dependent access control to support distributed clinical trials. We concentrated on presenting the technical aspects of the solution, in particular on the use of Java Servlets. The implementation concept comprises a secure distributed solution to context-dependent access control [11] over the Internet. The described system has not yet been implemented. It will be an extension to the current version of the *MobiMed* prototype that is presented in [12]. The new system will as well be PC-based (Windows NT/ Windows 2000), but using a Java Servlet rather than a Perl-script to access the MS SQL Server database (that contains the medical data) in a context-dependent fashion. To secure the communication the secure sockets layer protocol (SSL) is used. The context information for the access control is supplied by a commercial workflow management system. Even though developed for a concrete platform, the underlying security concepts are general and in particular the Internet security issues can be adapted to different platforms and applications [4].

ACKNOWLEDGMENT

We would like to thank Prof. Dr. Kurt Bauknecht and the Swiss National Science Foundation for their support as well as Konstantin Knorr for helpful suggestions for the final version of the paper.

REFERENCES

- [1] H.-R. Fischer, S. Teufel, C. Muggli, and M. Bichsel. Privacy and Efficiency of Mobile Medical Systems (MobiMed). Bewilligtes Forschungsgesuch des SNSF Schwerpunktprogramms Informatikforschung SPP-IuK, Modul: Demonstrator, Nr. 5003-045359, 1995.
- [2] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol version 3.0. In *Internet Draft <http://home.netscape.com/eng/ssl3/draft302.txt>*, Netscape, Transport Layer Security Working Group, November 1996.
- [3] S. Garfinkel and G. Spafford. *Web Security and Commerce*. O'Reilly and Associates, 1997.
- [4] E. Hepworth and U. Ultes-Nitsche. Security aspects of a Java-Servlet-based web-hosted e-mail system. In *Proceedings of the 7th IFIP Working Conference on Information Security Management & Small Systems Security*, 1999. Kluwer Academic Publishers, Boston.
- [5] R. Holbein. *Secure Information Exchange in Organizations*. PhD thesis, University of Zurich, Switzerland, 1996. Shaker Verlag, Aachen.
- [6] R. Holbein and S. Teufel. A context authentication service for role based access control in distributed systems - CARDS. In J. Eloff and S. von Solms, editors, *Information Security - the Next Decade IFIP/SEC'95*. Chapman & Hall, London, UK, 1995.
- [7] R. Holbein, S. Teufel, and K. Bauknecht. The use of business process models for security design in organisations. In S. Katsikas and D. Gritzalis, editors, *Information Systems Security - Facing the Information Society of the 21st Century (IFIP/SEC'96)*. Chapman & Hall, London, UK, 1996.
- [8] R. Holbein, S. Teufel, O. Morger, and K. Bauknecht. A comprehensive need-to-know access control system and its application for medical information systems. In L. Yngström and J. Carlsen, editors, *Information Security in Research and Business (IFIP/SEC'97)*. Chapman & Hall, London, UK, 1997.
- [9] J. Hunter. *Java Servlet Programming*. O'Reilly and Associates, 1998.
- [10] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores. The action workflow approach to workflow management technology. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'92)*, Toronto, 1992. ACM Press.
- [11] U. Nitsche, R. Holbein, O. Morger, and S. Teufel. Realization of a context-dependent access control mechanism on a commercial platform. In G. Papp and R. Posch, editors, *Global IT Security - Proceedings of the 14th International Information Security Conference (IFIP/Sec'98)*, volume 116 of *OCG*, Vienna, Austria, 1998.
- [12] S. Röhrig and K. Knorr. Towards a Secure Web-Based Health Care Application. In: *ECIS'2000* (these proceedings).
- [13] W. Stallings. *Cryptography and Network Security*. Prentice Hall, New York, second edition, 1998.
- [14] Sun Microsystems. *Java Servlet API Whitepaper*. 1998.
- [15] T. Ting, S. Demurjian, and M.-Y. Hu. Requirements, capabilities, and functionalities of user-role based security for an object-oriented design model. In C. Landwehr, editor, *IFIP WG 11.3 Workshop on Database Security*, Sheperdstown, West Virginia, 1991. Elsevier Science Publishers.
- [16] T. Ting, S. Demurjian, and M.-Y. Hu. A specification methodology for user-role based security in an object-oriented design model. In B. Thuraisingham and C. Landwehr, editors, *IFIP WG 11.3 6th Working Conference on Database Security*, Simon Fraser University Burnaby, Vancouver, British Columbia, 1992. Elsevier Science Publishers.