

Security Analysis of the Utilization of Corba Object References as Authorization Tokens

Christoph Becker
PrismTech
cbecker@prismtech.com

Sebastian Staamann
PrismTech
staamann@prismtech.com

Ralf Salomon
University of Rostock
ralf.salomon@uni-rostock.de

Abstract

In object-oriented real-time computing scenarios, particularly where Corba is used in embedded systems with resource constraints, developers and system architects often utilize Corba object references as authorization tokens. This paper investigates the security of this method in principle, and it presents the results of the analysis of the work and computing effort necessary for a potential attacker to fabricate Corba object references to existing objects for the purpose of illegitimately gaining access to these objects at the instance of seven widely used Corba products.

1. Introduction

Many software developers and system architects in the field of object-oriented real-time computing technology use Corba [1] or Real-time Corba [2] as the middleware basis for their distributed applications. It is common practice to utilize the Corba object references they have to deal with anyway as a means of implicit authorization too. Implicit authorization means that handing over an object reference to a client object does not only provide the necessary addressing and context information but also authorizes the client entity to access the respective object. An illustration for the widespread utilization of implicit authorization and prescription in de-facto standards for distributed, real-time, embedded, and object-oriented software systems is the Software Communication Architecture (SCA) for software defined radios [3] of the US DoD's JTRS program; an example that illustrates that the idea has been adopted even in environments with very high security requirements. The Security Supplement to the SCA spec. [4] mandates implicit authorization for the prevention of unauthorized access to Corba objects.

The attraction of implicit authorization to system designers, particularly for real-time and embedded systems with hard execution time constraints and often

limited computing resources, is that it avoids the need for the implementation of an additional logic of explicit access rights which actually just restates the logic of the authorized interactions of the Corba objects.

An assumption with implicit authorization is that entities that have not legitimately received an object reference for the target object cannot access the object. The matter to be investigated here is if and how the available Corba technology guarantees that the implicit authorization mechanism cannot be circumvented. This question is fundamental for any reasoning about the usefulness of implicit authorization.

This paper discusses the ways in which external attackers or rogue software can illegitimately obtain object references and access objects using these references, thus undermining implicit authorization. Direct attacks on the assumed secrecy of the object references like eavesdropping of object references and fake inquiries at Name Services are only the obvious ways to illegitimately get hold of other parties' object references. Indirect attacks like the fabrication of object references from easily available context information, such as the Corba product and version, the transport address, and object adapters, do not even require the direct access to the object references.

For a representative group of COTS Corba products this paper has analyzed the work and computing effort needed for an attacker (or for corrupted software modules) to intelligently guess and probe references of active objects. The results show that attackers using methods of intelligent guessing can in many cases fabricate object references for valid objects with modest computational effort and time needed. The feasibility of such attacks impairs the security of a number of Corba- and RT-Corba-based systems already in use, often in distributed real-time embedded (DRE) environments of significant criticality.

Section 2 of this paper discusses the utilization of Corba object references as authorization tokens and the idea of implicit authorization in general. Section 3

analyzes the implicit assumptions of this method and categorizes the different types of attacks possible. Section 4 presents a detailed analysis of the work and computing effort needed to intelligently guess and probe references of active objects with available Corba products. Section 5 gives an overview on possible counter measures, and Section 6 concludes this paper with some reasoning about the applicability of implicit authorization in the light of the results presented.

2. Implicit authorization

Right from its inception, one of the guiding principles for Corba, the common object request broker architecture, has been location transparency, i.e., the expansion of the object oriented programming paradigm to scenarios where the objects belonging to an application do not anymore have to necessarily reside in the same process and addressing space nevertheless can still all be used by the application programmer and in the application code as if they did.

As a key concept for location transparency, Corba has introduced the concept of Interoperable Object References (IORs). An IOR is a specialized object reference. It is used by an application in the same way that an ordinary local object reference is used. An IOR allows an application to make remote method calls on a Corba object. Once an application obtains an IOR, it can access the respective Corba object via IIOP¹, the protocol used between the Corba infrastructures on both sides to exchange the request and reply messages of the method invocation on the remote object. The Corba object can be implemented with any Corba compliant Object Request Broker (ORB) product that supports IIOP. The application that obtains the IOR can be developed with a different ORB. The application constructs a GIOP message and sends it. The IOR contains all the information needed to route the message directly to the appropriate server. Figure 1 illustrates the internal structure of an IOR, which is revised in Section IVb.

Corba as *the* dominant distributed objects technology has almost perfectly achieved its original goal of location transparency of any single object. However, the boundaries between process and addressing spaces of distributed objects, which are now so easily bridged by Corba and which can almost be ignored by the application programmers, are very often also boundaries with some security relevance, e.g., between domains of different ownership, different

¹ IIOP is the realization of the Generic Inter-ORB Protocol, GIOP, on top of TCP. The discussion in this paper is at the instance of IIOP. The issues discussed, however, apply to GIOP in general.

administration, different code origin, and different governing security policies. This is a significant difference to the boundaries between co-local objects in the same process and addressing space.

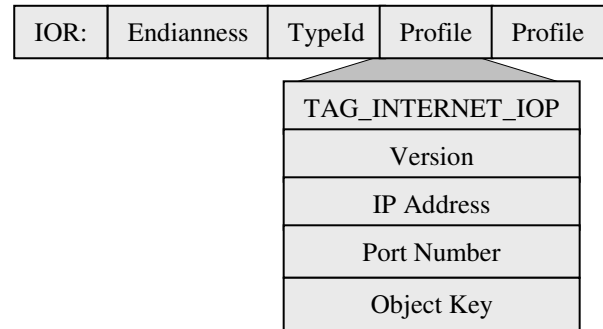


Figure 1. Interoperable Object Reference

The security perspective on this is naturally concerned with the implementation of security controls at boundaries with security relevance. From this point of view, it has to be noted that architects and developers of distributed applications, encouraged by the location transparency enabled by the use of Corba as the distribution middleware, have very often developed distributed object systems as if there were no security relevant boundaries between the distributed objects and as if all parties that technically have access to the system (including the parties that may access the data exchanged on the transport link) were of the same trust level, a level of almost complete mutual trust.

However, it is a common case that systems that have been designed and built in this way are deployed in environments that in fact do have security and trust boundaries. The justification with regards to potential threats to the system is very often that only parties that have been given the necessary IORs can access the Corba objects exposed across a security boundary. This way of thinking means that handing out an IOR to a third party is actually an act of implicit authorization.

3. Attacks on implicit authorization

Behind any implicit authorization of a party to access a Corba object through the handing out of an IOR to this object to this party stands a fundamental assumption, an assumption that needs to be investigated. It is widely believed that IORs could only be obtained in legitimate ways from a party that hands out IORs in a responsible way in line with the security policy (typically other parts of the server side

application that the Corba object is a part of) and that there was no other way of getting hold of these IORs.

The ultimate goal of all types of attack on the implicit authorization method is to illegitimately invoke an operation on a Corba object (thus violating the confidentiality of the information managed by the object or the integrity of the system by changing information and/or state) or make the object unavailable (e.g., crashing unstable servers by sending syntactically incorrect messages, e.g., containing fabricated object keys). For all types of attack, this paper assumes that the attacker has access to a network over which he/she can send messages to the ORB servers hosting Corba objects that may attract attempts to illegitimately access them. Furthermore, this paper assumes that in most cases, an attacker can have already learned the operations available from the objects to be attacked before the actual access attempt, e.g., because they are defined in published interface types (for instance in the IDL definitions of publicly available Corba based standards as often seen in telecomms) or from the interface repository. This paper also assumes that the attackers are able to create IIOP compliant request messages into which they put the operation name and the object key they may have learned by one of the attacks explained below.

Implicit authorization can, depending on the protection characteristics of the environment (e.g., exposure of the communication link to eavesdropping or message injection), be attacked in one of the following two ways:

1. An attacker may obtain Corba object references directly, e.g., by eavesdropping of communication links or getting such IORs from a conspiring third party that has received these references legitimately.
2. An attacker may fabricate Corba object references from addressing information he/she can learn comparatively easily (IP addresses and TCP port numbers) using standard network scanning techniques (as they are widely used by Internet hackers) and object keys intelligently guessed (see Subsection 4.3).

Once the attacker has learned a Corba object reference (and the operations offered), he/she can illegitimately invoke these operations on the target object.² It is worth mentioning here that even

² Actually, the attacker does not need to create the object reference as a data structure; it is sufficient to have the information

encryption of the communications link (e.g., using SSL) often does not protect against such attacks because many Corba servers in operation accept the establishment of SSL connections without checking whether the client is actually authorized by the server operator to establish such a connection (or even do not authenticate the client at all).

4. Fabrication of Corba object references

In a practical study, we have investigated in detail the feasibility of attack 2 listed above with a number of Corba implementations, including ORBs for which the source code is accessible (e.g., Open Source and ORBs for which source code is available) as well as ORBs where it cannot be assumed that an attacker may easily get access to the source code. This section reports the results of the analysis of seven Corba implementations.³

4.1. Test set and test environment

The test set includes Java and C++ ORBs. The members of the sample are characterized as follows:

- ORB1 is a C/C++ open-source ORB, mostly used in applications, such as embedded systems, where a small footprint is required.
- ORB2 is the Java member of a small-footprint low-cost ORB suite.
- ORB3 is the Java member of a suite of full-fledged enterprise ORBs.
- ORB4 is a widely used Java ORB, perhaps the most widely used Java ORB at all.
- ORB5 is the Java member of another suite of full-fledged enterprise ORBs.
- ORB6 is the ORB part of one of the most widely used Java enterprise application server product suites.
- ORB7 is the ORB part of another very widely used Java enterprise application server product suite.

It is worth noting that most of the Java ORBs analyzed have an equivalent C++ ORB from the same manufacturer with the same language-independent design concepts and internal structure. For the analysis, the Java versions have been chosen in order to ease the

contained at hand to create and send an IIOP request message attempting the illegitimate access.

³ The names of these products are not revealed here; however, the authors are convinced that these products are fairly representative for the real-world use of Corba nowadays.

testing procedure and in order to re-use the program code written for the tests to the maximum extent. All tests have been performed in the same testbed. It consisted of an AMD Athlon XP1800+ with 512 MB of memory, with the Ubuntu 6.06 LTS distribution package of Linux as the operating system. All time measurements were taken on this platform. The guessing attacks reported below have been performed either running the ORB and the guessing-attack pseudo-clients on the same machine, or running just the ORB on this platform and the guessing-attack pseudo-client on a Dual-Pentium III with 700 MHz and 1 GB memory (also running Ubuntu 6.06 LTS), the two machines connected by a 100 MBit Ethernet.

4.2. The structure of object keys

IORs are generated by the server-side ORB, in most cases when the target Corba object is created. For the client application, the IOR is an opaque data item. The client-side ORB, however, must be able to interpret most constituents of an IOR. Thus, most constituents also have a standardized internal structure. They can be learned or guessed by attackers comparatively easily, using common hacker techniques and tools, such as network address plans and port scanners [5]. The only non-standardized part is the object key part of the profile in an IOR. The object key is not interpreted on the client side, neither by the application nor by the client-side-ORB. It is the only part of an IOR that may be difficult to be learned by an attacker. The main part of our analysis is thus concerned with the intelligent guessing of object keys. The first step for an attacker for intelligent guessing is to determine the structure of the object keys of the ORB to be attacked

The internal structure of object keys is specific for each Corba product. However, the structure of object keys reflects in most cases (even in different forms, specific for the ORB product) more or less directly how ORBs work and how they are organized. Typically, an object key consists of both a (cardinal) number to identify the object and the name of the Portable Object Adapter (POA) that is responsible for this object. The name of the POA may also be a cardinal number. Many ORBs try to achieve the necessary uniqueness of the object keys through the incorporation of pseudo-random data, such as timestamps, random numbers, hash values, etc.

4.3. The guessing methods and the tools used

Two different general methods have been used. The first method was used on products where the source

code can be retrieved or reconstructed by attackers. It consists in a the reverse engineering of the `activate_object()` method, commonly used to construct a valid object key.

The second method is based on the analysis of valid IORs generated by a certain ORB instance: It analyzes a large number, e.g., 1,000,000 of valid but different object keys in order to recognize their structure. In the simple case, an attacker may generate these object keys using an instance of the ORB software he/she has at his/her discretion; in other cases, an attacker may retrieve these IORs in other ways. The analysis of the object keys in the retrieved IORs aims at the understanding of the generation algorithm in order to consequently fabricate object keys that enable the access to other Corba objects hosted by the ORB. The method had to consider not only ORB products but also different live instances of the same products, including different POA hierarchies and different POA policies. Also this method for the recognition of object key structures involves some educated guessing based on the recognition of patterns.

The practical tests for the second method were performed using a specifically created software tool that automatically creates POAs, activates objects, and reports the respective IORs. With the help of an IOR-decoder, such as JacORB's `dior`, we extracted the object keys from the IORs. In order to automatically handle thousands of object keys in a comfortable way, `dior` has been modified so as to allow reading and decoding IORs from multiple files.

In a first step, this tool was first creating one POA and was then activating a million of objects at this POA. As an example, Table 1 shows the generated object keys with number 0, 1, 2, and 255 of one ORB. Obviously, the object number appears in hexadecimal form at the end of the key.

<i>Object #</i>	<i>Object Key</i>
0	44 55 07 6E 00 00 00 00
1	44 55 07 6E 00 00 00 01
2	44 55 07 6E 00 00 00 02
255	44 55 07 6E 00 00 00 FF

Table 1. Example sequence of object keys

In the second step, this tool varied the POA hierarchy, the names of POAs, and their creation time. Time variation in this case refers to the time difference between the creation of the object and the creation of the POAs. This step revealed some basic workings of the object key allocation algorithms used. One result of this second guessing method was that valid object keys are not always limited to information about the POA-

hierarchy, the POAs names, and an object-counter (mostly as a cardinal number); some ORB implementations also include the ORB's name, called ORB-Id or ORB-name, and/or the time of creation.

The same approach was consequently applied to all ORBs under consideration. It revealed most of the internal structure of the object keys. Those parts that appeared unchanged during the entire testing procedure were assumed to be constant.

After we had found the object key structures of all ORBs analyzed using the tool already mentioned, a second tool, an IOR-guessing software tool specifically developed for the analysis, was used to generate arbitrary valid object keys. Since the structure of object keys is vendor specific, so is the generation of IORs. Thus a specific version of the guessing tool had to be built for each ORB analyzed. For each generated object key, this tool tried to access those objects that exist in the known ORB domain, but for which the tool has not received a valid IOR. To test if an object is accessible, running `object._non_existent()` on every object is the easiest way. This method returns an exception, if the object does not exist or false, if the object exists. In this way it was possible to validate for any fabricated IOR whether it indeed refers to an existing Corba object.

4.4. Results

The procedure described above was applied to the seven ORBs under consideration. For six of them, it is possible to fabricate IORs. Reflecting the knowledge an attacker might have obtained before the guessing attack, this section considers varying degrees of assumptions about the environment, each of which leaves the possibility of one or more of the following three attacks:

1. Find a valid object without any information about the ORB, except host and port,
2. Find an object on the same POA with a given IOR,
3. Find one on a different POA with a given IOR.

A general result of these tests is that none of the ORBs is vulnerable to attack 1; an attack would take time in the order of at least one year, which is too long in most practically relevant scenarios. A second result is that all tested ORBs are vulnerable to attack 2; it takes only between a few milliseconds and a few minutes (depending on the ORB) to obtain all active objects of a particular POA. A third general result is that with respect to attack 3, the seven ORB implementations yield different results, which is due to

the differing implementations of those parts of the object keys that refer to the managing POA.

Another finding is that once an attacker has obtained a valid IOR (according to attack 2), all other objects of that POA are prone to illegal access; two object keys that refer to objects at the same POA only differ in the value of the object counter, as has already been indicated by Table 1. That effectively means that after attaining a valid IOR, accesses to other objects of the same POA is possible within seconds; just incrementing the object-counter would lead to the next valid object key.

A general finding is concerned with attack 3, which turns out to require a medium effort as compared to the other attacks. Since an IOR issued by another POA is known, it is comparatively easy to find other valid POAs and to guess the object keys issued by this other POA. The POA that is generally easily testable is the RootPOA, because every ORB instance inevitably has a RootPOA and its name is fixed.

The remainder of this section discusses some selected results and findings as well as their explanations on an ORB-specific basis.

ORB1 uses a simple cardinal plus a UNIX-timestamp (time of the ORB's creation). If the point in time of the ORB's creation is narrowed to a day, there will be only 86,400 possible timestamps and the cardinal of the POA. ORB1 does not use a stringified ORB-name, instead it uses a pseudorandom number, such as the ORB's process id, for this. Unfortunately, the process id is an integer with less than 2^{16} different values. The complexity of POA and ORB names of ORB1 is larger than 10^9 . The practical tests have shown that one object key could be tested within milliseconds. That means at least 65 days are needed to test all of these keys. This does not include the cardinals for POA and object – it is assumed that they are 1. If these cardinals are also included, the time needed will be a multiple. This result is only applicable to attack 1. For attack 2 only the cardinal of the object has to be changed. Testing 10^6 objects takes about 17 minutes. After that short time period, the attacker knows every activated object at this POA and could have tested the type of the object. Attack 3 does not appear more complex than attack 2, because only the POA cardinal has to be added. Testing 10^3 POAs with 10^3 objects each takes also about 17 minutes. In summary, ORB1 is a potential victim.

ORB2 has a much more complex object key structure. It contains the creation times of both the ORB and the POA, the full POA structure, and the object counter. The guessing of both timestamps, if narrowed to a day, requires about 86 days. Attack 2 is

as fast as simple and brings the same results as with ORB1. Testing 10^6 objects took about 17 minutes. Attack 3 is hard, since the creation times of the POAs as well as the POAs' names must be guessed. In case the day of creation is known and the POA name is one among a thousand candidates, the object counter is still subject to be guessed. Testing 10^3 objects takes 97 days. Detailed information about the POA names and structure are needed for a successful attack.

ORB3 is immune to attacks 1 and 3 due to the fact that it uses a 12-byte hash as a POA name. The algorithm for generating these hash values could not be found out within the time budget. With respect to attack 2, ORB3 is as vulnerable as the other ORBs.

The object key of **ORB4** consists of an ORB timestamp, the POA-structure, the object-counter and two magic numbers. These numbers encode the ORB's version and Java version, respectively. For attack 1, the timestamp and the POA structure make it complicated to guess a valid object key. Referring to assumptions already discussed above, this attack seems impossible. Attack 3 has to guess a valid POA name and structure. This is feasible only for the RootPOA. If a possible string is one out of a thousand candidates, testing 10^3 objects will also take about 17 minutes. The difficulty consists in guessing the POA's name and its structure. Currently, no further assumptions can be made.

The object key of **ORB5** consists of the POA structure, a POA timestamp, and an object counter. Surprisingly, persistent objects at ORB5 do not have the POA timestamp in their object key. For attack 1, this means that it is easier to guess the object keys for long-living objects. Only the POA structure and the object counter have to be guessed. But usually, long-living objects own a user-defined name. If it can be assumed that a dictionary attack leads to a valid object key for both the POA and the object, only 10^3 strings have to be tested, and an attack could be successful within 17 minutes. But this result is not representative. For other objects the POA's creation time has also to be guessed. Bound to a day, three years are still required to test all object keys. For attacks 2 and 3, the results are the same as for ORB2, except for long-living objects. These could be guessed faster, due to the missing POA time.

ORB6 does not use a POA name or structure inside the object key exactly like ORB1 and ORB3. The object key contains an ORB name, which is a hash value consisting of 4 bytes. This hash value seems to be time depended, but the algorithm behind it could not be resolved. Due to the fact that only the cardinals for the POA and the object have to be changed, attacks 2 and 3 would exhibit the same time requirements as for ORB1.

ORB7, although coming from an unrelated vendor, has almost the same object structure as ORB2, and thus, the results and findings are basically the same.

5. Options for Counter Measures

First, it should be restated that the security problems discussed in Sections III and IV only exist in environments where the application designers solely rely on implicit authorization using object references as authorization tokens. Corba-based application systems that employ any form of explicit authorization and access control, either at the application level or at the Corba level (e.g., using an implementation of the Corba Security Service specified by the OMG [6] [7] [8]), do not face these problems, because in these cases the ability to present a valid object reference alone is not sufficient to access a Corba object.

In scenarios, where it is guaranteed by the characteristics of the environment that legitimately created object references given to authorized entities cannot be eavesdropped on the transport connection or in any other way be learned by potential attackers, it is sufficient that each object key contained in object references given to the outside contains a unique bit sequence (used once) that is practically unpredictable. Examples of such bit sequences are pseudo-random numbers as they are commonly used in cryptographic protocols (as described in standard text books on cryptography, e.g., [9]). The length of the pseudo-random part of the object key must be sufficient to rule out that an attacker can probe with a significant share of possible values. In cases where the unique used-once bit sequence is generated through cryptographic hash algorithms, it has to be ensured that no potential attacker can learn or guess the data from which the hash value is generated.

A special case of using cryptographic hash algorithms is to generate the pseudo-random part attached to the object key as the cryptographic hash value of the concatenation of the essential parts of the object key (e.g., portable adapter identifier and object ID) and a long-living secret bit sequence (secret key) known only to the object server (which in fact is the generation of a message authentication code, MAC). Such a MAC attached to the object key and always sent along enables the receiver of a Corba request (i.e., the Corba server) to verify that it had created the object key contained itself.

The object references suitable for implicit authorization over secrecy-preserving transport connections (i.e., Corba object references containing an object key that contains a unique, used-once

unpredictable bit sequence) can be generated either (i) by the server-side ORB software itself (as the standard way of the ORB to generate Corba object references), or (ii) by a server-side Corba gateway that intercepts all GIOP messages going out and replaces the original Corba object references with proxy references that fulfill the requirement of non-predictability. An example of such a Corba gateway is Ingham, Rees, and Norman's Object Gateway [10], which implements the MAC based approach described above.

An approach to guarantee the security of the use of Corba object references as authorization tokens also in environments where legitimately created object references given to authorized entities could have been leaked to attackers that may attempt to use them to illegitimately access objects, is the session based concept realized in the I-DBC product [11], which is another Corba gateway/firewall replacing references to be handed out to the outside world with proxy references. With the session-based concept, proxy references are bound to an access session of the authenticated client entity and are only valid within this session. The attacker, who cannot inject a request message containing an object key he/she may have got hold of into the (usually cryptographically protected) session for which this object key has been issued, can thus not make use of the illegitimately object key for any type of attack.

It should be noted that all options discussed above assume that the Corba server software, which issues the Corba object references, is itself not corrupted and hands out object references as an act of implicit authorization only if in line with the organizational security policy. A possible approach to support implicit authorization even for Corba servers with a low level of software assurance would be to have the Corba gateways checking the handing out of all references against a formalized IOR export policy, which is defined and administered by the server operator (not the software developer, thus increasing the assurance level of the system in operation) before proxifying it and handing it out.

6. Conclusion

The security of implicit authorization utilizing Corba object references as authorization tokens relies critically on the impossibility for outside users (clients, servers, potential attackers) to successfully use any Corba object references they may have got hold of illegitimately (i.e., obtained in any other way than legitimately receiving it from the server side for further use in the course of the interaction). However, this

paper has demonstrated that some ways to illegitimately obtain Corba object references, such as eavesdropping of object references on the transport or intelligent guessing based on knowledge of the internal structure and the interaction context, are indeed practically feasible with most ORB products available on the market. Especially the guessing and fabrication of such references is a realistic option for attackers, e.g., performed by corrupt software modules planted into distributed Corba based embedded systems, which enables practically unrestricted access to the other Corba objects in the system, thus undermining the security assurance of the whole system. In fact, the practical experiments have indicated that especially ORB products for embedded and real-time environments tend to have simple structures of object keys with no random parts involved, which makes guessing particularly easy.

References

- [1] Object Management Group. The Common Object Request Broker: Architecture and Specification, Version 3.0.1, November 2002.
- [2] Object Management Group. Real-time CORBA Specification, Version 1.2, January 2005.
- [3] US DoD Joint Tactical Radio System (JTRS) Joint Program Office. Software Communications Architecture Specification. MSRC-5000 SCA V2.2, November 2001.
- [4] US DoD Joint Tactical Radio System (JTRS) Joint Program Office. Security Supplement to the Software Communications Architecture Specification. JTRS-5000 SEC V2.2.1, April 2004.
- [5] M. de Vivo, E. Carrasco, G. Isern, and G.O. de Vivo. A Review of Port Scanning Techniques. In *ACM SIGCOMM Computer Communication Review*, Vol. 29, Issue 2 (April 1999), ISSN 0146-4833.
- [6] Object Management Group. CORBA Security, December 1995.
- [7] B. Blakley. *CORBA Security: An Introduction to Safe Computing with Objects*. Addison Wesley Longman, October 1999. ISBN 0201325659.
- [8] B. Hartman, D.J. Flinn, and K. Beznosov. *Enterprise Security with EJB and CORBA*. John Wiley & Sons, April 2001. ISBN 0471401315.
- [9] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC, October 1996. ISBN 0849385237.
- [10] D. Ingham, O. Rees, and A. Norman. CORBA Transactions Through Firewalls. International Symposium on Distributed Objects and Applications, 5-7 September, 1999, Edinburgh, United Kingdom.
- [11] Xtradyne I-DBC product Web page:
<http://www.xtradyne.com/products/i-dbc/i-dbc.htm>
<http://www.xtradyne.com/documents/whitepapers/Xtradyne-I-DBC-WhitePaper.pdf>