Better than Nothing Security	M. Richardson	
Internet-Draft	Williams	
Intended status: Informational	SSW	
Expires: September 25, 2009	M. Komu	
	Tarkoma	
	Helsinki Institute for Information	
	Technology	
	March 24, 2009	

TOC

C-Bindings for IPsec Application Programming Interfaces
draft-ietf-btns-c-api-04

Status of this Memo

Copyright Notice

Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

IPsec based security is usually transparent for applications and they have no standard APIs for gathering information on connection security properties. This document specifies an API that increases the visibility of IPsec to applications. The API allows applications to allow BTNS extensions, control the channel bindings, and control also other security properties related to IPsec. This document presents C-bindings to the abstract BTNS API.

Table of Contents

---

1. Introduction

TOC

The "better than nothing" (BTNS) extensions for IKE [I-D.ietf-btns-core] (Williams, N. and M. Richardson, “Better-Than-Nothing-Security: An Unauthenticated Mode of IPsec,” August 2008.) are intended to protect network traffic on their own (Stand Alone BTNS, or SAB), and may be useful in providing network layer security that can be authenticated by higher layers in the protocol stack, called Channel

Bound BTNS (CBB). The motivation for SAB is to remove the need to deploy authentication information altogether. The motivation for CBB is to remove the need for redundant authentication at multiple layers. This document defines APIs for these purposes. The APIs can also be used by other protocols such as the Host Identity Protocol (HIP) [RFC5201] (Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, "Host Identity Protocol," April 2008.) and Session Initiation Protocol (SIP) [RFC3261] (Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol," June 2002.). For example, a SIP user agent can use the presented APIs for determining whether or not required integrity and confidentiality protection is already in use. For certain networks and configuration this is expected to reduce overhead associated with the security mechanisms.

The network communications of applications are usually secured explicitly with TLS on transport layer [RFC4346] (Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," April 2006.), or using even higher layer interfaces such as GSS [RFC2744] (Wray, J., "Generic Security Service API Version 2 : C-bindings," January 2000.) or SASL [RFC4422] (Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)," June 2006.) APIs. However, such interfaces do not exist for IPsec because it operates on lower layers and is mostly transparent to applications. Using IPsec to protect existing applications is therefore easier than with, for example, TLS because IPsec does not require changes in the application. However, it is difficult for an application to detect when network connections are secured using IPsec. IPsec can be used as an "all or nothing" security measure, which can be problematic especially in deployments where the number of IPsec enabled machines is small. An alternative approach is to use IPsec when peer supports it. However, the application or the user may not have any knowledge that the communications was actually protected by IPsec in this case. In addition, it is more efficient to remove redundant authentications when IPsec and TLS are being used for the same connection.

In this document, we define APIs that increase the visibility of the IPsec layer to the applications. This document fulfills the BTNS requirements presented in [I-D.ietf-btns-ipsec-apireq] (Richardson, M. and B. Sommerfeld, "Requirements for an IPsec API," April 2006.) and present C-bindings to the abstract APIs [I-D.ietf-btns-abstract-api] (Richardson, M., "An abstract interface between applications and IPsec," November 2008.). The APIs defined in this document are based on the sockets API [POSIX] (Institute of Electrical and Electronics Engineers, "IEEE Std. 1003.1-2001 Standard for Information Technology - Portable Operating System Interface (POSIX)," Dec 2001.). For related API work, please refer to [I-D.ietf-hip-native-api] (Komu, M. and T. Henderson, "Basic Socket Interface Extensions for Host Identity Protocol (HIP)," January 2010.), [mcdonald] (Internet Engineering Task Force, "A Simple IP Security API Extension to BSD Sockets," Mar 1997.)

and [atkinson] (USENIX 1996 Annual Technical Conference, “Implementation of IPv6 in 4.4 BSD,” Jan 1996.).
The documents defines an explicit way of enabling IPsec in applications. This API allows the dual use of both IPsec and higher layer security mechanisms (TLS, GSS or SASL) simultaneously. The security and performance related benefits of this are described in more detail in [I-D.ietf-btns-prob-and-applic] (Touch, J., Black, D., and Y. Wang, “Problem and Applicability Statement for Better Than Nothing Security (BTNS),” July 2008.).

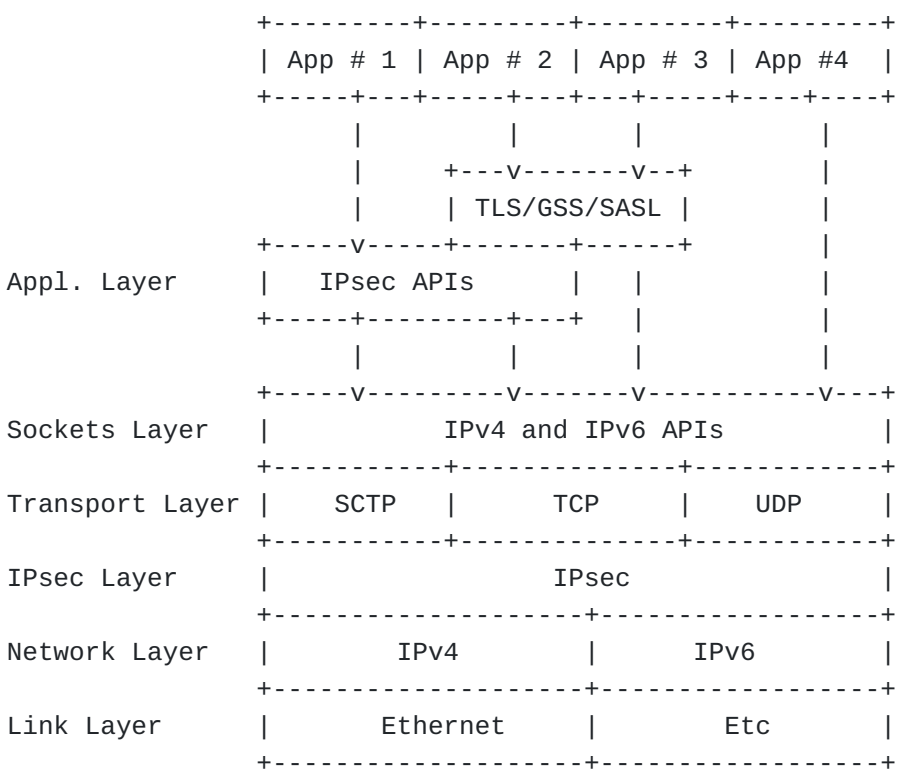

Figure 1: API Layering

Figure 1 (API Layering) illustrates four different applications. The first application is using only the IPsec APIs based on either IKE based authentication or Stand-alone BTNS. The second application is using both TLS (or other similar APIs) and IPsec APIs. In this case, the application can skip IKE authentication because of it is already provided by TLS. On the other hand, the application can avoid the use of TLS altogether when IKE authentication is available. The third application is using only TLS and the fourth one is using neither IPsec or TLS APIs.
In the first three cases, the application is explicitly modified to use either TLS or IPsec. In contrast, the fourth application is not using

either TLS or IPsec explicitly, but it may be using IPsec implicitly. This document covers the use of applications one and two.

2. IPsec APIs

TOC

The IPSec APIs are accessed by using tokens. The pToken has a per-process scope and is used to access the IPSec API. This token can be obtained, for example, from a connected socket, a received datagram, or a file descriptor. This token cannot be serialized. The iToken is a serializable token and represents the identity of a remote system. This section defines constants, data structures and functions for manipulating IPsec related data structures. The definitions are based on C-language. The integer values are always in host byte order.

2.1. Identity Tokens

TOC

Application can use identity tokes for querying the peer identity and for requiring certain channel bindings for a socket to implement ACLs or for logging purposes. Then, the application can communicate with a peer through the socket and the communication succeeds only when channel bindings are acceptable to the application. The application can also communicate with an peer of unknown identity, and to store and require the same peer identity in subsequent communications.

2.1.1. Creation of Identity Tokens

TOC

Identity tokens, iTokens, are machine-readable, opaque data structures. They can present either the local or remote identity, such as a public key. The iToken has a typedef which is illustrated Figure 2.

```
typedef struct ipsec_iToken * ipsec_iToken_t;
```

Figure 2

Operating environments that support the IPSec API will provide appropriate constructor and destructor for the iToken objects. Because applications will often not be aware of the byte-representation of the

iToken object, nor will they know which attributes to initialize upon construction, applications MUST only use the provided constructor to create an iToken object. When an iToken object is no longer needed, applications MUST use the provided destructor to destroy it. Figure 3 illustrates this API.

```
ipsec_iToken_t ipsec_create_iToken();
int ipsec_free_iToken(ipsec_iToken_t p);
```

Figure 3

Function ipsec_create_iToken() allocates memory for a iToken and initializes it. The function returns the created iToken, or NULL upon failure.
Function ipsec_free_iToken() deinitializes and frees the memory allocated to an iToken. It returns zero on success, and non-zero upon failure.

2.1.2. Attributes of Identity Tokens

TOC

This section describes the c-language bindings to section 8 in [I-D.ietf-btns-abstract-api] (Richardson, M., "An abstract interface between applications and IPsec," November 2008.). Identity token attributes are shown in Figure 4. They are accessed using the functions defined in Section 2.2 (Token Attributes).

```
enum {
     IPSEC_API_ATTR_auditString,
     IPSEC_API_ATTR_authenticationMethod,
     IPSEC_API_ATTR_certificateAuthorityDN,
     IPSEC_API_ATTR_certificateDN,
     IPSEC_API_ATTR_pubKeyID,
     IPSEC_API_ATTR_channelBinding
} iToken_attribute;

enum {
     IPSEC_API_ATTR_authMeth_NONE,
     IPSEC_API_ATTR_authMeth_BTNS,
     IPSEC_API_ATTR_authMeth_LEAPOFFAITH,
     IPSEC_API_ATTR_authMeth_PRESHAREDKEY,
     IPSEC_API_ATTR_authMeth_GROUPKEY,
     IPSEC_API_ATTR_authMeth_XAUTH,
     IPSEC_API_ATTR_authMeth_EAP,
     IPSEC_API_ATTR_authMeth_PKIX_TRUSTED,
     IPSEC_API_ATTR_authMeth_PKIX_INLINE,
     IPSEC_API_ATTR_authMeth_PKIX_OFFLINE
} iToken_auth_meth;
```

Figure 4

The group of attributes defined in iToken_attribute enumeration cannot be modified. The auditString attribute is a character array ending with a zero byte. It contains a human-readable description of the peer identity. The authenticationMethod attribute defines the key manager authentication method in an unsigned integer of two octets.
The certificateAuthorityDN attribute is a character array ending with a zero byte and contains a human-readable description of the peer certificate authority. The pubKeyID attribute contains a binary presentation of the peer public key. The channelBinding attribute is a character array ending with a zero byte. It contains a human-readable description of the channel binding. Two channel bindings can be compared with the memcmp() function.
The group of attributes in iToken_auth_meth enumeration contains a list of authentication methods. These attributes are both writable before network communications and readable after network communications. Here the use of the attributes is described only from the point of view of writing.
The attributes in this group are 2-octet unsigned integer values, with values IPSEC_API_ATTR_ENABLE, IPSEC_API_ATTR_DISABLE and IPSEC_API_ATTR_ANY. The first two of the values enable or disable the attribute, and third one refers that the application relies on the system defaults.

The attributes of the iToken_auth_meth are defined in [I-D.ietf-btns-abstract-api] (Richardson, M., “An abstract interface between applications and IPsec,” November 2008.).
The first NONE attribute describes that no authentication should be used. The BTNS attribute enables or disables the extensions defined in [I-D.ietf-btns-core] (Williams, N. and M. Richardson, “Better-Than-Nothing-Security: An Unauthenticated Mode of IPsec,” August 2008.)
The LEAPOFFAITH attribute declares that the peer was authenticated using a key which was previously cached, but was previously received inline, and was not verified in anyway.
The PRESHAREDKEY attribute denotes that a unique preshared key should be used and GROUPKEY correspondingly refers to a non-unique group key.
The XAUTH, EAP, and PKIX attributes refer to the respective authentication methods.

2.2. Token Attributes

TOC

IPsec properties are handled indirectly using objects called tokens. They are are opaque data structures that must not be manipulated directly. Instead, the application uses accessor functions shown in Figure 5.

```
int ipsec_get_token_attr(const void *token,
                         uint32_t attr_type,
                         uint32_t *attr_len,
                         void *attr_val);
int ipsec_set_token_attr(const void *token,
                         uint32_t attr_type,
                         uint32_t attr_len,
                         const void *attr_val);
```

Figure 5

Both of the functions can be applied both to policy and identity tokens to retrieve or change the low-level attributes.
Function ipsec_token_attr_get() searches for the given attribute type (attr_type) from the token and writes it to attr_val. Parameter attr_len defines the size of attr_val structure in bytes.
Function ipsec_set_token_attr() writes the attribute (attr_val) to the token. The type and length of the attribute must be set in attr_type and attr_len. The attr_val must not be NULL and attr_len must have the size of the allocated object.

Both of the functions return zero on success. They return -1 on error and set errno accordingly.

2.3. Protection Tokens

TOC

An application creates a "protection token" and attaches some attributes for it. For example, the application can define in the attributes of protection token that it accepts BTNS extensions for a certain socket.

2.3.1. Creation of Protection Tokens

TOC

Application uses protection tokens, or pTokens, as "handles" to the key management or the IPsec module of the host. The application uses pToken attributes to e.g. enabled the BTNS extensions and to control iTokens. The former allows the use of IPSec without authentication, and the latter allows e.g. querying of channel bindings.
The data structure that represents a pToken is contained in an opaque ipsec_pToken structure. The application must not alter the data structure contents directly, but rather use the accessor functions introduced in the following sections. The application can use ipsec_pToken_t typedef as a short hand for the policy structure. The typedef is shown in Figure 6.

```
typedef struct ipsec_pToken * ipsec_pToken_t;
```

Figure 6

The size of a policy is variable and applications MUST NOT declare them directly. Instead, the application uses the constructor and destructor functions shown in Figure 7.

```
ipsec_pToken_t ipsec_create_pToken(void);
int ipsec_free_pToken(ipsec_pToken_t p);
```

Figure 7

Function ipsec_create_pToken() allocates memory for a pToken and initializes it. The function returns the created pToken, or NULL upon failure.
Function ipsec_free_pToken() deinitializes and frees the memory allocated to a pToken. It returns zero on success, and non-zero upon failure.

2.3.2. Attributes of Protection Tokens

TOC

This section defines c-bindings for section 7 in [I-D.ietf-btns-abstract-api] (Richardson, M., "An abstract interface between applications and IPsec," November 2008.). Protection token attributes are shown in Figure 8. They are get or set using the functions defined in Section 2.2 (Token Attributes).

```
enum {
     IPSEC_API_ATTR_privacyProtected,
     IPSEC_API_ATTR_integrityProtected,
     IPSEC_API_ATTR_compressionAvailable,
     IPSEC_API_ATTR_policyName,
     IPSEC_API_ATTR_iToken,
     IPSEC_API_ATTR_remote_iToken,
     IPSEC_API_ATTR_tunnelMode,
     IPSEC_API_ATTR_ipoptionsProtected,
     IPSEC_API_ATTR_auditString,
     IPSEC_API_ATTR_informationString
} pToken_attribute;
```

Figure 8

The attributes of the pToken_attribute structure are defined in [I-D.ietf-btns-abstract-api] (Richardson, M., "An abstract interface between applications and IPsec," November 2008.).
Here the use of the attributes is described only from writing point of view. Attribute value IPSEC_API_ATTR_DISABLE defines that the attribute should not be used. Value IPSEC_API_ATTR_ENABLE describes that the corresponding attribute should be used.
It is possible to enable an attribute by declaring the "level" of the attribute with IPSEC_API_ATTR_LEVEL_LOW, IPSEC_API_ATTR_LEVEL_MEDIUM or IPSEC_API_ATTR_LEVEL_HIGH.

The privacy, integrity and compression attributes are 2-octet unsigned integer values. These attributes are writable before network communication and readable after network communications. They can be used to enforce and negotiate required attribute values.
privacyProtection - unsigned integer. Set to IPSEC_API_ATTR_DISABLE if the connection has either no privacy configured (AH, ESP-null), or if the privacy configured is known to be untrustworthy by the administrator.
integrityProtection - unsigned integer. Set to IPSEC_API_ATTR_DISABLE if there is no data integrity protection other than the UDP/TCP checksum.
compressionAvailable - unsigned integer. Set to IPSEC_API_ATTR_DISABLE if data count sent/ received from socket maps directly to data sent/ received on wire.
policyName - string. A handle which describes the system policy which was used (or is desired), to establish the connection.
iToken - object. Set to iToken object which represents identity of remote system.
remote_iToken - object. Set to iToken object which was used to represent our identity to the remote system.
tunnelMode - unsigned integer. Set if tunnel mode was used, or if it is desired.
ipoptionsProtected - unsigned integer. Set if ip options (and IPv6 header extensions), are protected.
auditString - string. The auditString is a character array ending in zero byte and contains a human readable description of the protection token.
informationString - string. Readonly. Not part of a template. Valid only after connection establishment. Contains a string which can be displayed to a user, informing them of what kind of security association was established for this connection. This string may be localized. No session keys are disclosed by this string.

2.3.3. Connection Oriented Communications

TOC

Declaring a pToken does not affect the networking communications of an application. For connection oriented communications, the application must first attach the pToken to the socket before the pToken is effective. It is also possible to query for the pToken attached to a socket as shown in Figure 9.

```
int ipsec_set_socket_pToken(int fd, const ipsec_pToken_t pToken);
int ipsec_get_socket_pToken(int fd, ipsec_pToken_t pToken);
```

Figure 9

Both functions input an socket descriptor as the first argument and a pToken as the second argument. Function ipsec_set_socket_pToken() attaches the given pToken to the socket descriptor fd. Function ipsec_get_socket_pToken() assumes that the application has allocated the policy token beforehand with ipsec_create_pToken().
Both functions return zero upon success, and non-zero upon failure.

2.3.4. Datagram Oriented Communications

TOC

The previous section covered the use of connected sockets. Datagram oriented communications based on sendmsg() and recvmsg() functions are supported in the API. Datagram related functions are applicable both to incoming and outgoing packets. The IPsec API functions related sendmsg() and recvmsg() are shown in Figure 10.

```
int ipsec_set_msg_pToken(struct msghdr *msg,
                         const ipsec_pToken_t pToken);
int ipsec_get_msg_pToken(const struct msghdr *msg,
                         ipsec_pToken_t pToken);
```

Figure 10

Function ipsec_set_msg_pToken() attaches the given pToken to the ancillary data of msg. The pToken of a msg can be queried using ipsec_get_msg_pToken() that assumes the application has allocated the policy token beforehand with ipsec_create_pToken.
Both functions return zero on success. The functions return -1 on error and set errno accordingly.
It should be noticed that these functions can be applied only to sendto() and recvmsg() as they support per packet anchillary data. Applications using sendto() and recvfrom() can apply the "stream-based" functions described in the other sections of the document with certain restrictions. TBD: discuss.

TOC

2.3.5. Equivalency of Protection Tokens

An application is not allowed to read or write to pTokens directly. The same restriction applies also to comparison of pTokens. The function for comparing two pTokens is shown in Figure 11.

```
int ipsec_cmp_pToken(ipsec_pToken_t p1,
                     ipsec_pToken_t p2);
```

Figure 11

Function ipsec_cmp_pToken() inputs two policies, p1 and p2, and returns zero if they represent two SAs that cover identical SPD ranges, and have equivalent cryptographic security properties. The function returns a nonzero value if p1 is not equal to p2. The two SAs need not represent SAs that identical --- they might vary in many different ways, including, but not limited to:

- Time: One SA may have been created later, but both are valid.
- Jitter/performance properties: One SA may be on hardware and the other on software, and have different properties about what kind of latency or jitter a packet might experience
- Algorithm: one SA might use AES128-CBC while the other uses AES128-CTR (DISCUSS) for performance reasons.
- IPsec SA endpoints. The two SAs may cover the same inner IP packets, but might connect using differing outer IP addresses, and be used in some kind of multipath IPsec (such as MOBIKE).

2.3.6. Duplication of Protection Tokens

TOC

Byte-wise copying of pTokens is not allowed e.g. with memcpy(). Function ipsec_dup_pToken() duplicates given pToken p and writes it to p_dup. The function allocates the memory for duplicated pToken that the caller is responsible of freeing. Return value is zero on success and non-zero on failure.

```
int ipsec_dup_pToken(const ipsec_pToken_t p,
                     ipsec_pToken_t *p_dup);
```

Figure 12

3. Security Considerations

TOC

The BTNS Stand Alone mode allows applications to omit network layer authentication. In this case, an application is using a higher level security mechanism, such as TLS, and thus the required level of security is maintained. Thus, the application avoids applying duplicate security measures on the network connection.
The channel bindings allow applications to create and manage security channels. Given that applications omit higher layer security techniques based on information in an existing pToken and the corresponding channel binding, there is a possibility for a security channel downgrade attack. In this attack, another application modifies the current application's channel binding in such a way that the application believes that an authenticated IPsec security channel to be active even though there is no such channel. If the application omits TLS or other higher level security mechanism, then there will not be a secured channel and transmitted data is exposed.

4. IANA Considerations

TOC

There are no registries created by this document. The names (and language specific enum) of the pToken and iToken properties are internal to a single system, and therefore do not need standardization.

5. Acknowledgements

TOC

Thanks for Love Hörnquist Åtrand, Julien Laganier and Vijay Gurbani for feedback, ideas and discussion on the topic. The authors wish to thank also Simon Josefsson and Daniel McDonald for comments on the draft.

TOC

6. References

6.1. Normative References

TOC

[I-D.ietf-btns-abstract-api]	Richardson, M., “An abstract interface between applications and IPsec,” draft-ietf-btns-abstract-api-02 (work in progress), November 2008 (TXT).
[I-D.ietf-btns-core]	Williams, N. and M. Richardson, “Better-Than-Nothing-Security: An Unauthenticated Mode of IPsec,” draft-ietf-btns-core-07 (work in progress), August 2008 (TXT).
[I-D.ietf-btns-ipsec-apireq]	Richardson, M. and B. Sommerfeld, “Requirements for an IPsec API,” draft-ietf-btns-ipsec-apireq-00 (work in progress), April 2006 (TXT).
[I-D.ietf-btns-prob-and-applic]	Touch, J., Black, D., and Y. Wang, “Problem and Applicability Statement for Better Than Nothing Security (BTNS),” draft-ietf-btns-prob-and-applic-07 (work in progress), July 2008 (TXT).
[POSIX]	Institute of Electrical and Electronics Engineers, “IEEE Std. 1003.1-2001 Standard for Information Technology - Portable Operating System Interface (POSIX),” Dec 2001.

6.2. Informative References

TOC

[I-D.ietf-hip-native-api]	Komu, M. and T. Henderson, “Basic Socket Interface Extensions for Host Identity Protocol (HIP),” draft-ietf-hip-native-api-12 (work in progress), January 2010 (TXT).
[RFC2744]	Wray, J., “Generic Security Service API Version 2 : C-bindings,” RFC 2744, January 2000 (TXT).
[RFC3261]	Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, “SIP: Session Initiation Protocol,” RFC 3261, June 2002 (TXT).
[RFC4346]	Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.1,” RFC 4346, April 2006 (TXT).
[RFC4422]	Melnikov, A. and K. Zeilenga, “Simple Authentication and Security Layer (SASL),” RFC 4422, June 2006 (TXT).
[RFC5201]	Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, “Host Identity Protocol,” RFC 5201, April 2008 (TXT).

[atkinson]	USENIX 1996 Annual Technical Conference, “Implementation of IPv6 in 4.4 BSD,” Jan 1996.
[mcdonald]	Internet Engineering Task Force, “A Simple IP Security API Extension to BSD Sockets,” Mar 1997.

Authors' Addresses

TOC

	Michael C. Richardson
	Sandelman Software Works
	470 Dawson Avenue
	Ottawa, ON K1Z 5V7
	CA
Email:	mcr@sandelman.ottawa.on.ca
URI:	http://www.sandelman.ottawa.on.ca/
	Nicolas Williams
	SUN Microsystems
	5300 Riata Trace Ct
	Austin, TX TX 78727
	US
Email:	Nicolas.Williams@sun.com
	Miika Komu
	Helsinki Institute for Information Technology
	Metsänneidonkuja 4
	Espoo
	Finland
Phone:	+358503841531
Fax:	+35896949768
Email:	miika@iki.fi
URI:	http://www.iki.fi/miika/
	Sasu Tarkoma
	Helsinki Institute for Information Technology
	Metsänneidonkuja 4
	Espoo
	Finland
Phone:	+358503841517
Fax:	+35896949768
Email:	sasu.tarkoma@hiit.fi
URI:	http://www.cs.helsinki.fi/u/starkoma/