Smart-M3 Security: Authentification and Authorization Mechanisms

Kirill Yudenok
Saint-Petersburg
Electrotechnical University
Saint-Petersburg, Russia
kirill.yudenok@gmail.com

Ilya Nikolaevskiy
Helsinki Institute for
Information Technology
Helsinki, Finland
ilya.nikoalevskiy@hiit.fi

Abstract

Smart spaces are dynamic environments for sharing device information. Key challenges for smart spaces include security and interoperability between heterogeneous devices. Thus, smart spaces and its environments must provide feasible solutions for authentication, access control and privacy. Open source Smart Space platform Smart-M3 is actively developed but does not have a sufficient security mechanism yet. The main focus of this paper is analysis and development of security mechanisms for Smart-M3 platform.

Index Terms: Smart Spaces, Smart-M3, Security, Access control, HIP.

I. Introduction

Smart spaces (SS) are physical spaces where devices cooperate and share information to intelligently provide services for the users. *Cook D. J.* in [1] define a smart environment as one that is able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment. The terms smart space and smart environment are widely used interchangeably — this article uses the term smart space.

Smart spaces are vulnerable for various security and privacy related threats. This technology expands borders of the different kinds of attacks and opens up new possibilities for the penetration to the agents SS data. Therefore, we need solutions for identification and authentication of all smart space agents, confidentiality and access control. Solutions should be applicable for embedded devices with limited communication, processing, memory and battery capabilities. Also, solutions should work in dynamic environments where new devices may join, store and subscribe information and leave at any time [2].

There are few existing smart space software implementations: Smart-M3 [3], [4], ADK [5] and RIBS [6]. Our solutions are based on smart space Smart-M3 platform.

Smart-M3 is an open source software platform that aims to provide Semantic Web information sharing infrastructure between software entities and various types of devices. The platform combines ideas of distributed, networked systems and Semantic Web [7]. The major application area for Smart-M3 is the development of smart spaces solutions, where a number of devices can use a shared view of resources and services. Smart spaces can provide better user experience by allowing users to easily bring-in and take-out various electronic devices and seamlessly access all user information in the multi-device system from any of the devices.

Smart-M3 consist of two kinds of architectural elements: Knowledge Processors (KP) and Semantic Information Brokers (SIB). KPs join to the smart space and publish and consume information in it. Brokers provide smart space access, information storage, retrieval and subscription services. For information exchange between KPs and SIB using Smart Space

Access Protocol (SSAP). In order to enable interoperability, different KPs must know the representation format of data. Applications may utilize different ontologies, which define the concepts, properties, and their relations for different use cases. To enable use of ontologies, information is transmitted in eXtensible Markup Language (XML) format and stored according to Resource Description Framework (RDF) [8] specification [4].

This article will address the implementation of the security mechanisms for the Smart-M3 platform. Section 2 describes related work on the research and development of SS security mechanisms, provides guidance on the selection and development of security mechanisms and provides architecture of security mechanisms for the Smart-M3 platform. Section 3 is devoted to the research and development of basic security mechanisms, authentication mechanism based on the HIP protocol and access control mechanism based on the mapping SS RDF-graph to the virtual file system (VFS). Improvements of proposed security mechanisms are also presented in that section. Section 4 describes details of security mechanisms implementation.

II. Smart Space security

A. Smart Space security recommendations

There are many works devoted to the research and implementation of the SS security mechanisms. Following SS security mechanisms are defined in accordance with each Semantic Web level as presented in Fig. 1 [7]. The Semantic Web Layers [9] (left side) illustrates the hierarchy of languages, where each layer exploits and uses capabilities of the layers below. It shows how technologies that are standardized for Semantic Web are organized to make the Semantic Web possible. The essential security elements (right side) illustrates how some of above described security technologies fit to the layer of Semantic Web.

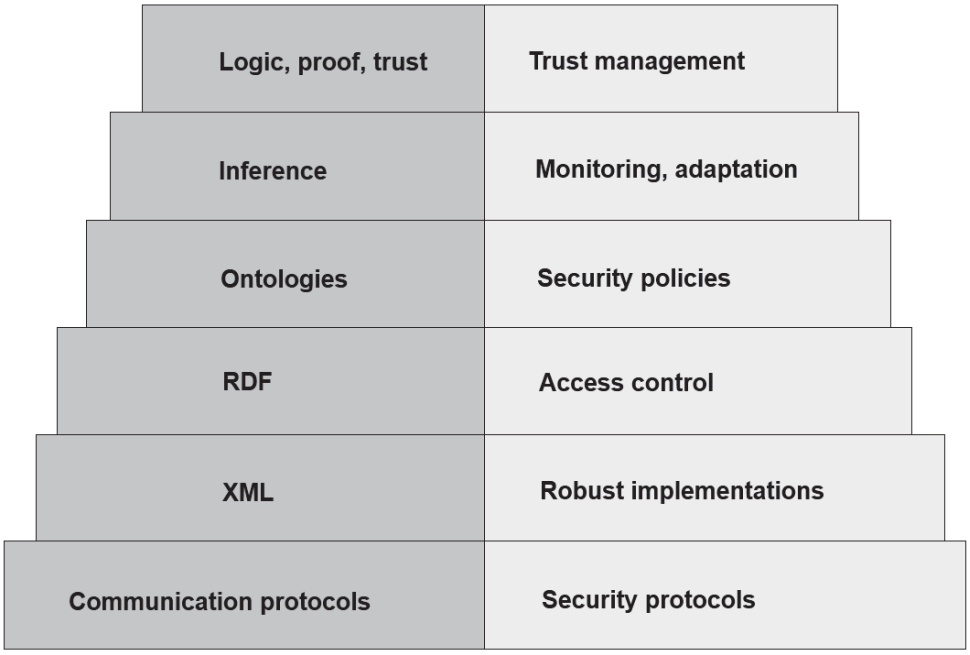

Fig. 1. Smart space security cake — Layers of Semantic Web (left [7]) and essential security elements (right)

Security technologies use variety of solutions, such as cryptography and encryption keys management for identification, authentication and connection privacy for secure information transmission between all parties. Security mechanisms are located at all represented levels.

Each level provides solution for certain security problem. For example, handling protection and input validation of XML documents is done at the XML level, access right description languages, RDF security policies and specialized ontologies are used at RDF Access Control level [5], [6].

Different SS platforms use different communication protocols between its modules (SIB and KP): Smart-M3 uses SSAP/XML protocol for exchanging data, RIBS uses SSAP/WAX protocol. These protocols cannot boast high productivity, which reduces the whole system performance. A new communication protocol — "Knowledge Sharing Protocol" (KSP) [10] solves that problem and can be used in the future to ensure security (access control) based on the context, RDF and ontologies.

The security mechanisms discussed in this article located at "Communication protocols" and "RDF" levels respectively. These mechanisms provide the identification and authentication of agents at the network level and data access control at the application, but access control solution is outside of this chart.

For now Smart-M3 platform has a unique security mechanism — Access Control at Triple Level [11]. Smart Spaces security problems described in the articles [2], [12].

B. Security architecture for Smart-M3

Our proposed security solution protects information sharing between KP and SIB. It provides privacy and information authentication. The Fig. 2 shows proposed architecture. KPs establish secure data channel with SIB. During key exchange all entities are cryptographically authenticated and all communication privacy is further secured with cryptography. System administrator is a Smart-M3 administrator, that configure all objects to trust certain objects and also get all information about the state of the security mechanisms.

System administrators are responsible only for objects they own. Therefore, each device owner will be system administrator for her mobile KP. She can choose to trust some Smart Spaces SIBs depending on her personal preferences or corporate guidelines. The SIBs are configured by responsible authorities. They decide which KPs may access corresponding Smart Spaces and that access rights they have.

III. Security mechanism for Smart-M3 platform

A. Network security

To provide robust authentication we propose to use HIP protocol for key exchange [13]. Unlike other protocols (IKEv2, DTLS) HIP provides additional DoS protection with puzzle mechanism. Normally HIP relies on HIP Base Exchange (BEX) to establish shared key which later is used for encryption with IPSEC. Adopting HIP provides privacy protection and secure identification of the hosts as cryptographic identities are used during key exchange.

To accommodate variety of constrained devices used as smart objects we propose using HIP Diet Exchange (DEX) [18]. DEX is a lightweight modification of BEX. HIP DEX requires rather limited computation capabilities from the devices [14]. It uses Elliptic Curve Cryptography to distribute shared secret between Initiator and Responder, see Fig. 3. In Smart-M3 context the KP will be initiator of a HIP DEX exchange and SIB will be responder. Although HIP DEX is designed to work in the restricted environments it still provides possibility to control performance level by adjusting cryptographic puzzle difficulty.

Since Smart-M3 has modular architecture it may be preferable to add HIP DEX as a new module into SIB and KP. Instead we propose to embed HIP DEX as a library only in modules responsible for network interaction. Such approach produces less overhead and allows tight

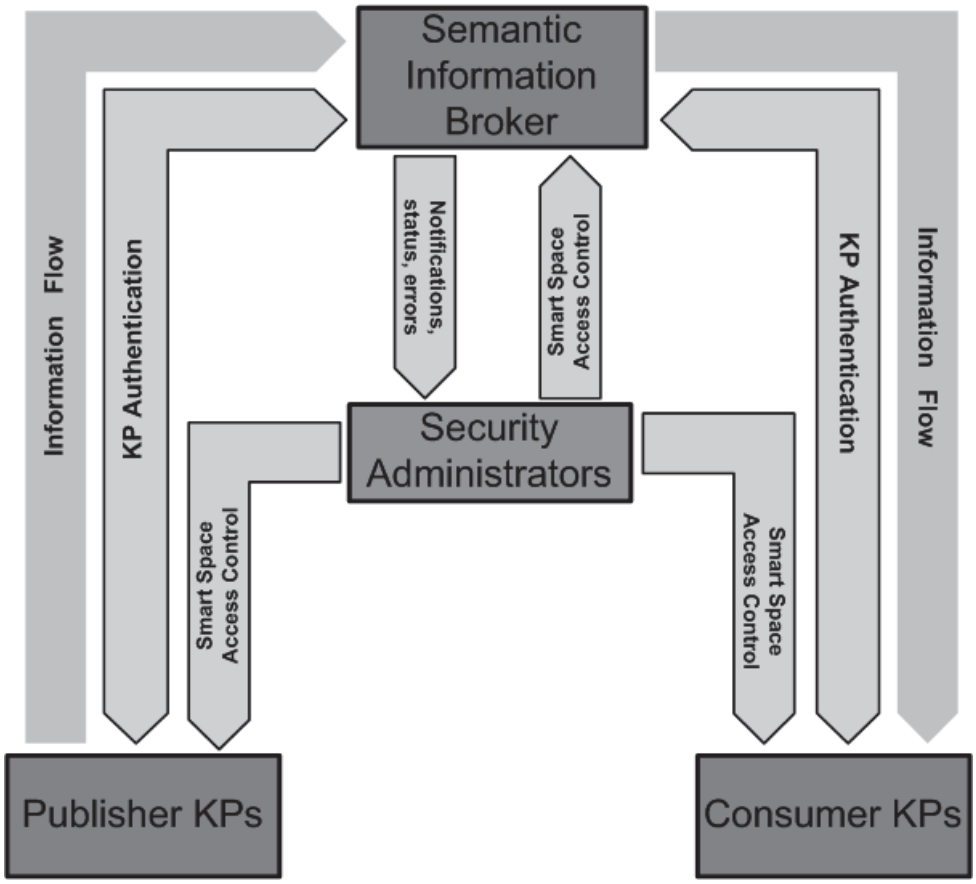

Fig. 2. Smart-M3 security architecture

integration of HIP and Smart-M3. Fig. 4 shows Smart-m3 system general architecture and HIP extension place there.

The HIP exchange also authenticates both devices, providing robust identities for smart objects. The tight integration of HIP and the SIB access module allows using HIP cryptographic identities for access control within the Smart-M3 space. In particular, SIB may restrict access to information that the KPs publish in the shared space.

To prove feasibility of our proposal we have made a prototype implementation. It is based on low-level ANSI C KP interface provided within SmartSlog SDK [15] and Redland SIB [17]. We use HDX++ library for HIP DEX protocol. We implemented a secure sib-tcp module of SIB and ansi-c-kpi module for KPI side. Before each join operation HIP DEX is performed and shared key produced during handshake is used later to secure all transferred data with AES-CTR encryption. Our experimental evaluation showed that proposed mechanism is feasible on constrained devices.

B. Access control mechanism

All Smart-M3 platform information is stored in a database (DB) (SQLite, Berkeley DB [19]), which is a representation of the RDF-graph, because all information is stored in the triple form. The access control mechanism will provide SS triples as a file system tree structure by mapping Smart-M3 RDF-graph to the virtual file system.

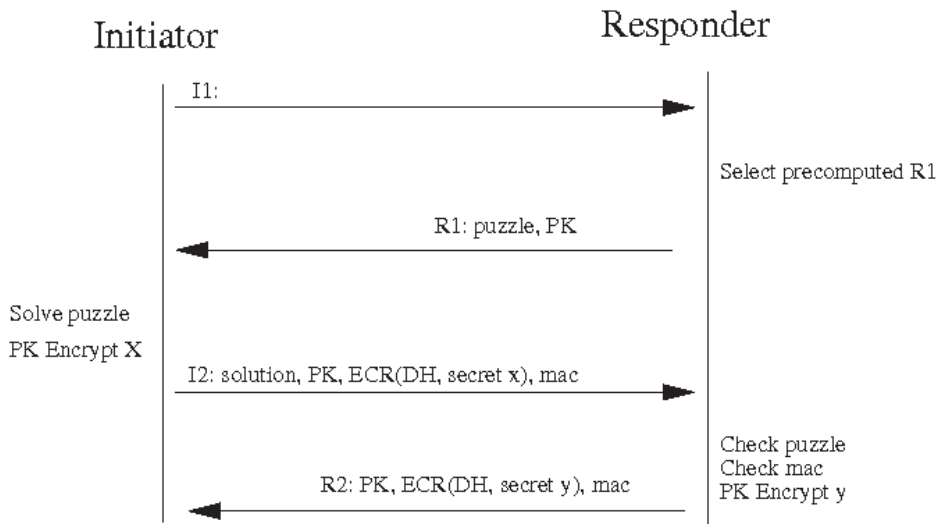

Fig. 3. HIP DEX scheme. Only four packets of total size about 530 bytes are used

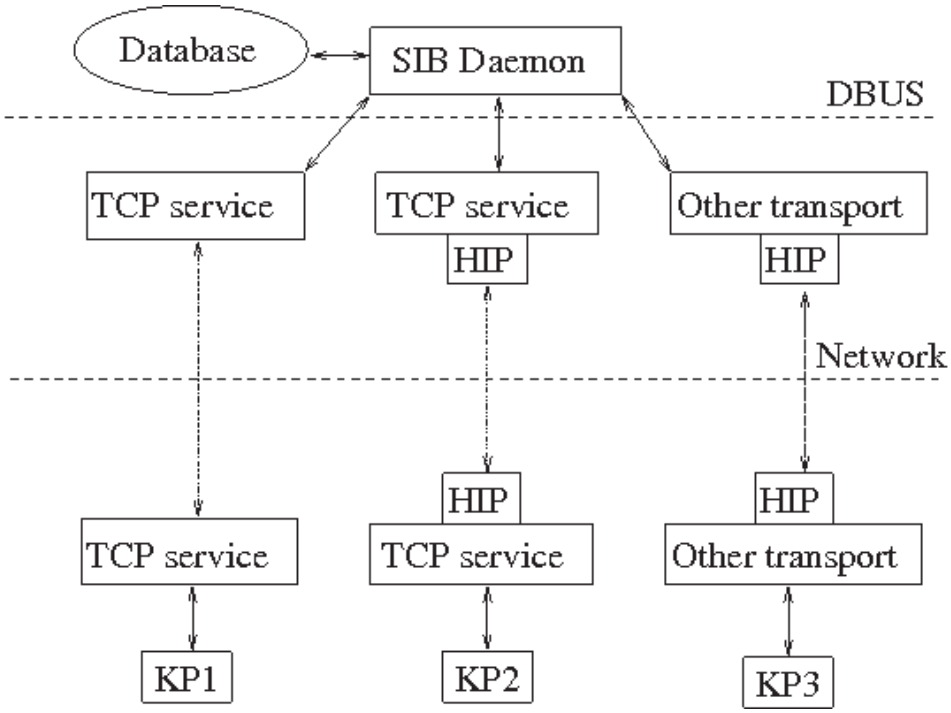

Fig. 4. Smart-M3 architecture with new HIP functionality

This representation data model will apply the standard file system access control methods to the Smart-M3 information. For this we assume that the file system operations are analogous to RDF operations. For example, the ”read” right will be similar to ”extract” (query) operation in RDF [12]. SS data mapping model to the VFS also allows us to use different methods for the file system management and control.

Finally, after the full Smart-M3 mapping to the certain file system structure, it will replace the standard RDF-graph from database to the VFS. Fig. 5 shows the location of the Smart-

M3 RDF-graph mapping mechanism in Smart-M3 platform architecture. VFS *SibFS* stores RDF-graph information in a specific directory structure, where S – RDF-graph subjects directory/file, P – predicates, O – objects directory/files. Details of the VFS structure described in [12].

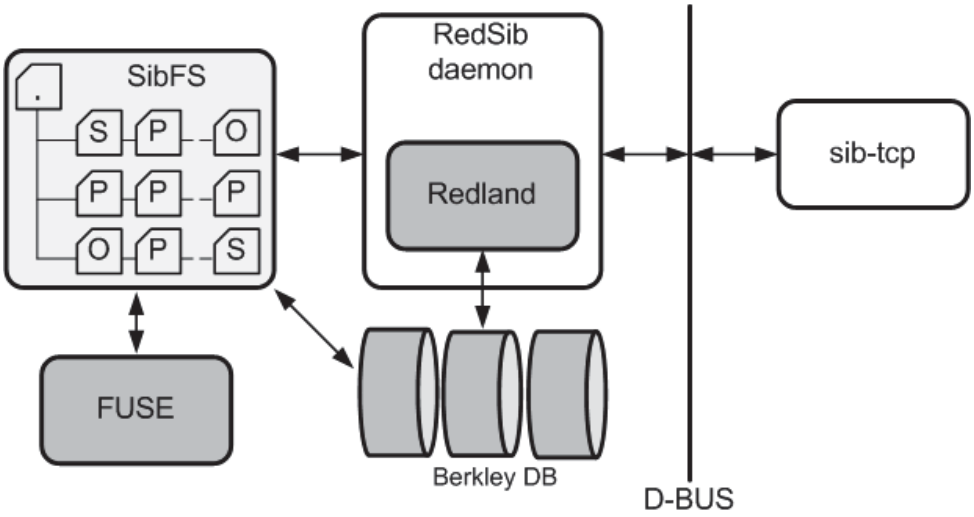

Fig. 5. Smart-M3 RDF-graph mapping overview

Smart-M3 platform is divided into client and server parts. Server module *"redsibd"* is responsible for interacting agents with Smart-M3 RDF-graph, namely, for all DB operations. Since the VFS assumes the duties of a main Smart-M3 storage, it is necessary, firstly, to link the virtual file system with Smart-M3 DB. As a result, the VFS will be filled with DB data; Secondly, develop KP's authorization function to check access to the Smart-M3 content. Thirdly, it is necessary to provide additional functionality to work with the VFS, for example, creating catalog-file chains when client inserting data to the RDF-graph, request selected data from the file system, extract the file system user rights, logging access history to the FS data and other functions.

1) Virtual File System interface: VFS is the main component of the access control mechanism. A VFS development based on DB consists of the following steps:

1) File System representation scheme creation in the Smart-M3 or separate DB. Create data structures for inodes, directories, links and other supporting data for storage FS representation.
2) FS entity creation interface development. Basic operations for FS presentation on the basis of the Berkeley DB.
3) File system interface definition and implementation, its core operations, which users can access in the file system.

Smart-M3 RDF-graph mapping model consists of formation and representation of the overall FS structures in the DB, so that the database is used not only for the representation file system entities (inodes), but also as an actual file system data.

2) SS subjects authorization function: The main purpose of the *Authorization()* function is to check the subjects rights to the file system object where the subject is a Smart-M3 agent (KP), rights is a subject access rights or rules in the file system and the object is a mapped DB entity (file or directory).

redsibd module is responsible for working with a DB and it is logical that the authorization function will be implemented in this module. This module has access to each KP connected to the Smart-M3 and all operated SIB data.

As a *Smart-M3 user* means KP connected to the platform. This is the fair, as well as for authentication mechanism and access control model. For each connected KP to the platform allocated a unique number (ID). And also assign access rights or rules to work with the platfrom regarding VFS. FS has assigned to it users with specific access rights. In fact, FS users will be the Smart-M3 KP's, so the access rights of FS users will apply to KP operations. This mechanism can be simplified by adding access rules (rules groups) to the VFS access mechanism.

Redsibd module stores all connected KP's ID in the *ssap_kp_header→kp_id* structure field. Structure *ssap_message_header* keeps inserted data graph in the *inserted_graph* field and requested data graph in the *query_graph*. User rights are set and retrieved from the access control list (ACL) or from the file system *inodes*.

C. Smart-M3 security improvement solutions

We consider various solutions to improve the security mechanisms of the Smart-M3:

1) *Security monitor*. Each full security system should have its own security monitor to control all system security processes. Security monitor performs the basic management and control functions of the system security mechanisms state. Its main functions are:
   - security systems monitoring;
   - security attributes installation;
   - mechanisms faults notification;
   - reliability and fault tolerance;
2) *Trust (access) levels*. This mechanism is based on assigning to SS agents some kind of data (permissions, roles), for example, based on HIP certificates that will indicate certain trust levels for each agent. Certificates may be issued during the client authentication when connecting to the SS or after changing the security level.
3) *Monitoring functions*. This is a logging function mechanism built in the Linux FS. To monitor state of the security mechanisms it is necessary to provide security mechanisms for logging a history of messages, received by the security monitor or some Linux monitoring daemon (*syslogd*). Because access control is based on the VFS, that mechanism may use system functions or special programs for file system monitoring, for example, *stat, fsstat, inotify* and other, but different attributes must be saved in a log file.
4) *Security levels*. System security levels required in the case of a suspicious client activity. Each security level restrict access to the data. For example, if the SS administrator did not reveal any suspicious activity – established green or yellow level (checking all data rights), in case of the suspicious is established strictest level – red, which checks all SS agents security attributes.

IV. Smart Space RDF-mapping to the VFS

A. Berkeley DB mapping to the VFS

A list of Smart-M3 DB mapping steps to the VFS is presented in the *"Virtual File System interface"* section. Here we will describe it in more details.

1) Each file system object (file or directory) is represented by the *inode* in a Linux. *inode* keeps all the metadata for the management file system objects (including the

possible operations). To store *inode* in the database it is necessary to determine them, for example, using a data structure. All representations, FS nodes will be eventually stored in the database and used for files, directories or links creation.
VFS mounted only one time with the redsibd launch and each file–catalog chain created during the KP Insert operation. KP Update operation will search the file and change its value and so on. Subscription operation will described after realization basic Smart-M3 SIB operations.

2) The database interface is performing several tasks: an execution of a basic database operations such as open/close database, write data; another major task is the inode management to represent file system files and directories. The operations of this task include: inode's {*create, delete, update, free*}, inode's {*list, search, definition*}, inode type definition, as well as operations on the inode basis: {*read, write, delete, search, list*} files, directories, links and other.
It is worth to note one aspect of saving data in the Smart-M3 DB. At first, all SS data located in the random access memory and only after a certain amount of data (approximately about 64 Kbytes), it is written to disk in three hash database. Function DB→sync() is responsible for this operation. This synchronization is necessary to increase system performance but permanent synchronization overloads PC resources and will not be effective.
3) FUSE technology [20] provides an interface to define low-level operations to work with the file system. The main task in creation your own file system, override these operations. For example, the file system initialization — open the database, FS cleaning — closed, reading is performed inode read operation and its data from the database. VFS entities and their data are represented using two separate databases. File system implementation based on Berkeley DB can be found in *dbfs* realization [21].

B. Mapping implementation in the Redland SIB

Authorization function interface is — bool authorization(gchar* kp_id, guint32 mode, gchar* object). It returns a boolean value indicating whether the subject right exist or not. Depending on this value, further execution of the SIB operation may be changed.

As mentioned earlier, all FS rights are stored in an *inodes*, also they can be extracted from the FS POSIX ACL [22]. Thus, it is possible to find agent rights for any file, because inode stores the user ID. For easy access to each user rights it may be stored in a database or extracted from inode each time. Details on how to retrieve information from all file system inodes and use them are described in the logging FS — *bbfs* [23].

In order for a complete Smart-M3 DB replacement to the mapped file system to occur it is necessary to rewrite functions responsible for handling Smart-M3 DB operations, such as *rdf_retractor* (remove), *rdf_writer* (insert), *rdf_reader* (query). When KP inserts data to the SIB, interface functions that handle a file system database will be called. In that case, will be called functions for inserting triplets in the common database and creation triple inode in the metadata database. SS subjects authorization function is called from each of these functions.

RDF-graph mapping mechanism implementation scheme to the Smart-M3 platform redsibd module is presented at Fig. 6.

The main goal while changing redsibd module operations is to achieve previous functionality and stability with the FS.

V. CONCLUSION

The following components are implemented as a basic Smart-M3 security mechanisms:

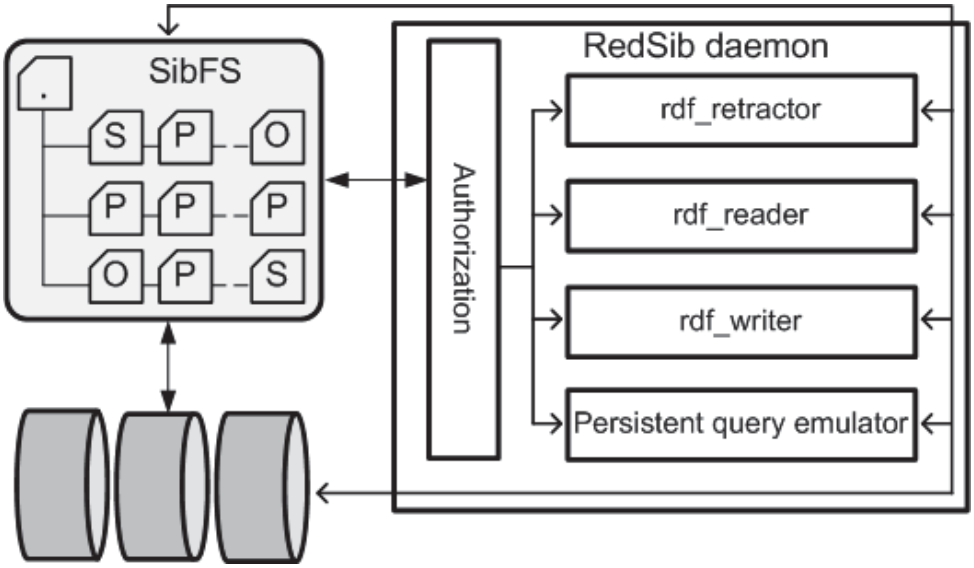

Fig. 6. RDF-graph mapping mechanism scheme

1) HIP DEX based authentication:
   - HIP DEX based modules for secure network communication (sib-tcp and ckpi);
   - HIP DEX cryptography based authentication;
   - AES-128-CTR encryption of all communications between KP and SIB.
   - proposed mechanism has low hardware demands and is applicable even for constrained devices used as KP;
2) Access control mechanism:
   - VFS RDF-graph mapping mechanism prototype;
   - basic interface for handling VFS permissions;
   - KP's authorization function implemented in the redsibd module;
   - started process for reworking Smart-M3 DB operations to VFS.

   Future research directions:
   - VFS rights rules and their groups mechanism;
   - VFS rights management tool;
   - security mechanisms testing, performance evaluation.

VI. Acknowledgement

Authors would like to thank FRUCT Smart Space Working Group (SS WG) for providing feedback and guidance.

References

[1] D. J. Cook, S. K. Das, How smart are our environments? An updated look at the state of the art, *Pervasive Mob, Comput*, 2007, 3, 53–73.
[2] J. Suomalainen, P. Hyttinen, Security Solutions for Smart Spaces, In *Proceedings of the 11th International Symposium on Applications and the Internet*, Munich, Germany, 18-21 July 2011, IEEE, 2011; pp. 297–302.
[3] Smart-M3 Open Source Project, Web: http://sourceforge.net/projects/smart-m3.
[4] J. Honkola, L. Hannu, R. Brown, and O. Tyrkkö, Smart-M3 information sharing platform, *IEEE Symposium on Computers and Communications (ISCC)*, pp. 1041–1046, 2010.

[5] J. F. Gomez-Pimpollo, R. Otaolea, Smart Objects for Intelligent Applications — ADK, *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2010, 267–268, DOI=http://doi.ieeecomputersociety.org/10.1109/VLHCC.2010.52.
[6] J. Suomalainen, P. Hyttinen, P. Tarvainen, Secure information sharing between heterogeneous embedded devices, in *Proceedings of the 4th European Conference on Software Architecture: Doctoral Symposium, Industrial Track and Workshops (ECSA 10)*, pp. 205–212, August 2010.
[7] T. Berners-Lee, J. Hendler, and O. Lassila, The semantic web, *Scientific American*, vol. 284, pp. 34–43, 2001.
[8] RDF Semantics, Web: http://www.w3.org/TR/rdf-mt/.
[9] Semantic Web Layers, Web: http://en.wikipedia.org/wiki/Semantic_Web_Stack.
[10] J. Kiljander, F. Morandi, J. P. Soininen, Knowledge Sharing Protocol for Smart Spaces, *International Journal of Advanced Computer Science and Applications*, Vol. 3, No. 9, 2012
[11] A. D'Elia, J. Honkola, D. Manzaroli, T. Salmon Cinotti, Access Control at Triple Level: Specification and Enforcement of a Simple RDF Model to Support Concurrent Applications in Smart Environments, In *Proceedings of the 11th International Conference, NEW2AN 2011, and 4th Conference on Smart Spaces, ruSMART 2011*, St. Petersburg, Russia, 2225 August 2011, Springer: Berlin-Heidelberg, Germany, 2011; pp. 63–74.
[12] K. Yudenok, K. Krinkin, Distributed Service Environment (Smart Spaces) Security Model Development, *12th FRUCT Conference of Open Innovations Framework Program FRUCT*, Oulu, Finland, 5-9 November 2012
[13] A. Gurtov, M. Komu, R. Moskowitz, Host Identity Protocol (HIP): Identifier/locator split for host mobility and multihoming, *Internet Protocol Journal*, vol. 12, no. 1, pp. 27–32, Mar. 2009.
[14] R. Moskowitz, *HIP Diet EXchange (DEX): draft-moskowitz-hip-rgdex-06*, May 2012, work in progress. Expires in November 2012
[15] P. Nie, J. Vähä-Herttua, T. Aura, A. Gurtov, Performance analysis of HIP diet exchange for WSN security establishment, In *Proceedings of the 7th ACM symposium on QoS and security for wireless and mobile networks*, ser. Q2SWinet'11. New York, NY, SA: ACM, 2011, pp. 51–56. [Online]. Available: http://doi.acm.org/10.1145/2069105.2069114
[16] D. G. Korzun, A. A. Lomov, P. I. Vanag, J. Honkola, S. I. Balandin, Multilingual ontology library generator for Smart-M3 information sharing platform, *International Journal on Advances in Intelligent Systems*, vol. 4, no. 3&4, pp. 68–81, 2011
[17] F. Morandi, L. Roffia, A. D'Elia, F. Vergari, T.Salmon Cinotti, RedSib: a Smart-M3 Semantic Information Broker implementation, *12th FRUCT Conference of Open Innovations Framework Program FRUCT*, Oulu, Finland, 5-9 November 2012
[18] HDX++ library, Web: http://sourceforge.net/projects/hdx/.
[19] Berkeley DB, Web: http://en.wikipedia.org/wiki/Berkeley_DB.
[20] FUSE documentation, Web: http://www.ibm.com/developerworks/ru/library/l-fuse/, http://fuse.sourceforge.net/.
[21] Berkeley DB4 File System (dbfs), Web: http://git.kernel.org/?p=fs/fuse/dbfs.git.
[22] POSIX ACL, Web: http://users.suse.com/~agruen/acl/linux-acls/online.
[23] Writing a FUSE File System: a Tutorial (bbfs), Web: http://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial/.