

Advanced Security Considerations in the Arrowhead Framework

Sándor Plósz¹, Csaba Hegedűs², and Pál Varga³

¹ Austrian Institute of Technology,
Dept. of Safety and Security, Vienna A-1220, Austria,
`sandor.plosz.fl@ait.ac.at`

² AITIA International Inc.,
Telecommunications Division, Budapest H-1039, Hungary,
`hegeduscs@aitia.ai`

³ Budapest University of Technology and Economics,
Dept. of Telecommunications and Media Informatics,
Budapest H-1117, Hungary
`pvarga@tmit.bme.hu`

Abstract. The Arrowhead Framework aims to create collaborative automation using networked embedded devices by establishing a service oriented approach to govern them. Various cyber-physical Systems can provide and consume Services from one another in closed automation clouds. These System-of-Systems has been introduced by the Arrowhead framework as Local Clouds. These clouds – being high value targets – can then be subject to an extensive amount of threats. This paper is dedicated towards revising the Arrowhead framework to further enhance its security solutions. A certificate-based architecture is presented to solve authentication and authorization tasks not just within, but in-between Local Clouds by using a token concept applied for services. This schema also allows the integration of resource constrained devices in coexistence with different levels of security.

Keywords: Internet of Things, Collaborating System of Systems, Authorization, Authentication, Certificates, Ticketing

1 Introduction

The Arrowhead project defines a framework for creating distributed industrial systems by collaborative networked embedded devices [12], [2]. This framework intends to integrate popular application layer protocols using a unique Service Oriented Architecture (SOA).

In an Arrowhead automation cloud environment, communications can be subject to an extensive amount of threats. These include i.e. spoofing, tampering or Denial of Service attacks, and can compromise the security and integrity of the whole infrastructure. These System-of-Systems still possess a general infrastructural vulnerability, despite its decentralized architecture [11].

In order to prevent such threats, the Arrowhead Framework has to provide strict authentication and authorization capabilities (AA). These functionalities are indispensable, considering the involved cyber-physical Systems and business processes. Furthermore, the framework targets the collaboration and interoperability of embedded, and often resource-constrained devices using proprietary or industrial protocols. In many cases the devices have limited capability of performing security tasks such as advanced encryption and decryption utilized by modern secure transmission protocols such as Transport Layer Security (TLS) or the Internet Protocol Security (IPSec) [10]. For such cases, a ticketing-based approach has been developed in [7], which works with CoAP. For MQTT and REST – which use the Transmission Control Protocol (TCP) –, security based on TLS would be desirable.

In this work we present the certificate-based security concept for the Arrowhead multi-cloud environment. We investigate how this can co-exist for the other ticket-based security concept within Arrowhead, and how translation between the protocols can work, while keeping security transparent.

2 The elements of the Arrowhead framework

2.1 Local Automation Clouds

It is a great challenge to get heterogeneous systems (or architectures) work together – especially in the automation domain, which has very specific real-time, and security requirements. Legacy protocols, commercial off-the-shelf products and monolithic architectures often cripple interoperability and dynamic reconfigurability. The Arrowhead Framework uses the approach of Service Oriented Architectures (SOA) [4] to tackle this problem: it aims at providing interoperability by facilitating the service interactions within closed or at least separated automation environments.

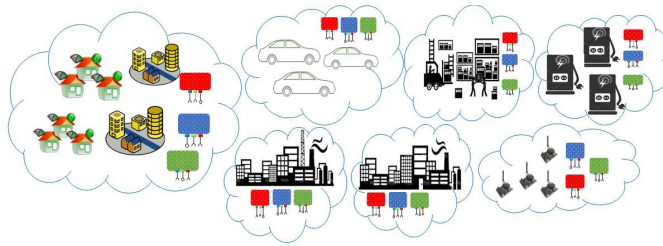


Fig. 1. Arrowhead Local Clouds deployed in different domains

These *Arrowhead Local Clouds* might fulfill various tasks and can have their own sets of appointed stakeholders (e.g. their operators or developers). These Local Clouds all have their operational boundaries, let those be functional, geographical or network-segmented. Nevertheless, they must be governed through

their own instances of the *Arrowhead Core Systems*, as Fig. 1. suggests. These are clouds in the sense that they use common resources: the Core Systems of that domain. These common Core System resources (e.g. related to Service Registry, Authorization or Orchestration) are used by all kinds of other entities – applications – in the network, and can also be implemented in a distributed way. This means that the scalability of the Local Cloud mostly depends on the scalability of the Core Systems (e.g. by implementing them using virtualization and Web Services [1] technologies).

2.2 Systems and Services

In the Arrowhead Local Clouds there can be an arbitrary number of *Systems* that can provide and consume *Services* from one another: they create and finish *servicing instances* dynamically in run-time. Services are defined so that loose-coupling, late binding, and service discoverability can be realized. Here, Arrowhead-compatible Systems must use the mandatory *Core Systems and their Core Services* provided by the Framework, to realize their operational targets (as shown in Fig. 2.). These Core Services also support to set up the initial field for information-exchange. There are two main groups of the Core Systems: the mandatory ones that need to be present in each Local Cloud, and the automation supporting ones that further enhance the capabilities of a Local Cloud.

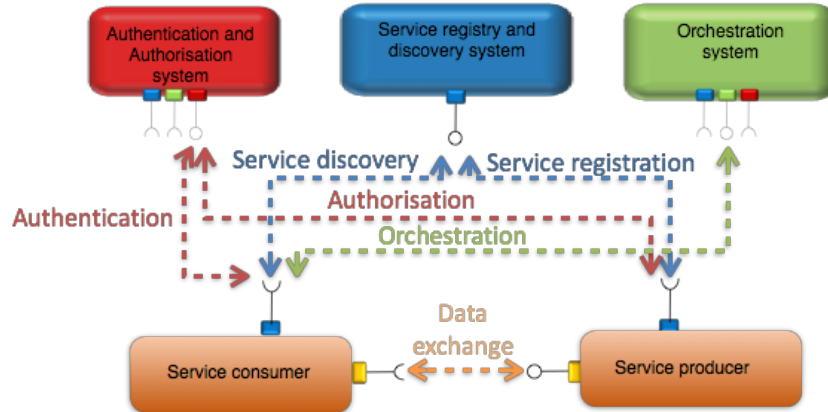


Fig. 2. Arrowhead Core Systems and their use

2.3 Core Systems and Services

In accordance with [12], this work attributes the same functionality to each of the Core Systems. There are three mandatory Core Systems:

- The Service Registry stores all the Systems (that are currently available in the network) and their service offerings. Systems have to announce their presence, and the services they can offer. The registry takes note of this information when systems come online, and might have to revoke them when they go offline.
- The Authorization System – as its name suggests – manages authentication and authorization (AA) tasks.
- The Orchestrator is responsible for instrumenting each System in the cloud: where to connect. It instructs Systems so by pointing towards specific Service Providers to consume specific Service(s) from. This has to be done by a simple request-response sequence, which ends with the requester System receiving a Service endpoint. After these, the System is obligated to consume from that Service instance.

There are further automation supporting Core Systems available, in order to provide additional core services such as event handling, system configuration, factory description, or inter-cloud orchestration, which should otherwise be implemented within many of the systems. This way these services are available for any of the systems, without consuming their local resources.

2.4 Inter-Cloud servicing

Arrowhead Local Clouds are generally autonomous and independent from each other, since they are deployed separately for various reasons. However, there are specific cases, where Systems from different clouds need to consume Services from one another. To this end, an inter-Cloud servicing architecture was introduced in [13]. This comes up with two additional Core Systems to the framework: the Gatekeeper, and the Network Manager.

The Gatekeeper provides essentially two services for the mandatory Core Systems:

- the Global Service Discovery (GSD) process, that aims at locating adequate service offerings in neighboring Clouds;
- the Inter-Cloud Negotiations (ICN) process, in which mutual trust is established between two Clouds and the actual connection between endpoints is then built up.

The concept is depicted by Fig.3. After these processes, the Network Manager is responsible for creating the data path between the Service Provider and the Consumer.

This setup includes some security considerations taking place, e.g. declaring inter-Cloud access rights (from and to other Local Clouds) in the Authorization Systems, or setting up the trusted neighborhood domains.

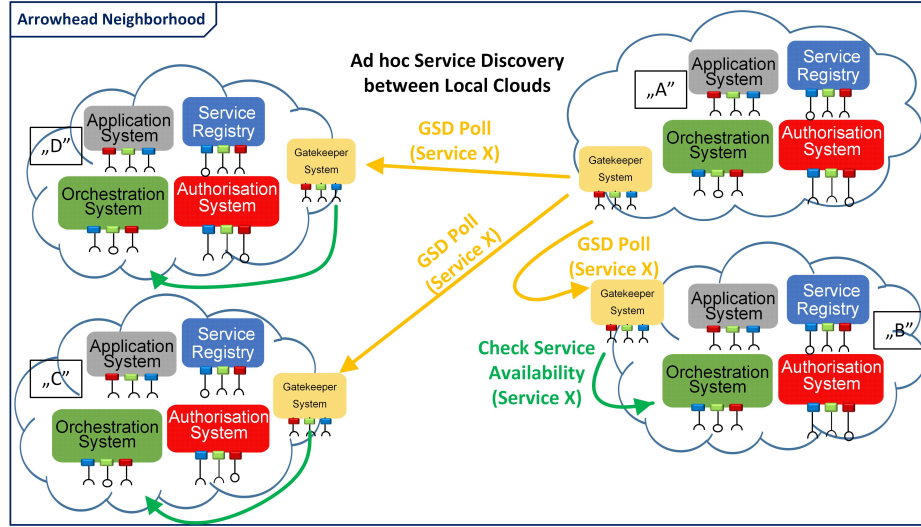


Fig. 3. Inter-Cloud service discovery between Gatekeepers

3 Security in the Arrowhead Framework

3.1 Issues to tackle

Assuring security in distributed architectures is not as trivial as in centralized systems: the roles for authentication and authorization are not bounded to one single entity but are to be taken up by everyone. We need to define a novel security solution in order to fulfill the strict security requirements of the automation world. A threat analysis methodology elaborated in [8, 9] suggests that the following possible vulnerabilities have to be tackled here:

1. Confidentiality: Data is only to be available for those who were intended to access it. This objective needs to be assured by proper authorization paired with authentication. Threats like spoofing and information disclosure can violate this objective.
2. Integrity: Data and system resources are only to be changed in appropriate ways by the appropriate people. This can be assured by proper encryption (or digest) and can be exploited by tampering type of attacks.
3. Availability: Systems should be ready when needed and perform acceptably. Denial of Service (DoS) attacks can endanger this objective.
4. Authenticity: The identity of users is to be verified (unless it is acceptable to service anonymous users). Spoofing type of threats aim to violate this.
5. Accountability: Users should not be able to deny performing their earlier actions. Proper and verifiable logging needs to be implemented to prevent this.

It is worth noting that while authentication should be mandatory for all communications, at least three further levels of application security should be acceptable for Application Systems within Arrowhead (even in co-existence). Besides having no additional security, there can be cases where only integrity is required and cases where both integrity and confidentiality is essential.

3.2 A ticketing approach

Resource-constrained (but Arrowhead-compatible) Systems might not be capable of implementing advanced security measures. A lightweight authentication solution has been developed for such Systems, based on the Constrained Application Protocol (CoAP), and the Kerberos protocol [7]. It uses a centrally issued ticket as a token of identity but does not contain any identity information directly. To authenticate Application Systems within a Local Cloud one must contact the AA server to verify its identity and privileges.

Besides its protocol-restricted implementation (supporting merely CoAP), the weak points of this authentication method make it vulnerable to a number of attacks. Firstly, the challenge-response nature on an insecure channel implies that the ticket itself does not verify the authenticity of the sender – and can be exploited with a man in the middle approach. Secondly, since the service provider has to request the local ticketing AA server to validate the ticket (and therefore authenticate the consumer), it is also easy to bypass this control loop. Thirdly, this validation process is requested at every inbound connection and therefore the AA server is vulnerable to a DoS attack.

On the other hand, this approach is advantageous for a number of reasons. The service providers are relieved from the processing burden of identifying and verifying the consumers: it is done by the central AA entity. It also bears the future capability of integrating it with admission control functions: service providers can assert whether the inbound connection is ratified by the Core Systems – and can be serviced – or not.

3.3 The Public Key Infrastructure

Public Key Infrastructure (PKI) [5] defines how to create certificates that can be used to achieve authentication, message integrity and confidentiality. PKI builds on Public-key cryptography (PKC), which employs an asymmetric encryption scheme. This means that it uses different keys for the encryption and the decryption processes, compared to symmetric key encryption methods, where the same key is used for the encryption and decryption.

There are two type of keys in a PKI architecture: these are called the public and the private keys. As their name suggests, they differ in their confidentiality level. While private keys belong to its owner and must be kept as a secret, whereas the public key can be freely distributed. Information encrypted by the public key can only be decrypted using the private key – and vice versa. Therefore, the validity of the information can be assured both ways (inbound and outbound) when the recipient/sender possesses one of the keys. However,

pre-designated trust is still required in certificate-based authentication and encryption methods that build on the features of the PKI. In order to achieve the mutual trust between entities, a Certificate Authority (CA) must be appointed in the system. The role of a CA is basically to sign certificates (and thereby to validate their identity and content). All entities in the system trust (and know) their CA and can only validate the received certificates that contain information about its counterpart (its public key) and is signed by the local CA. Certificates can be used to both verify the authenticity of the parties, as well as to check the integrity of messages in the communication. Therefore, parties that have signed certificates from the same CA can communicate securely.

As a consequence, if an entity A wishes to communicate with another entity (B), it will need to get the signed certificate of B containing the public key of B.

In the new Arrowhead Service, the functionality of a CA is realized and combined with an authorization methodology embedded in the orchestration process.

4 Enhancements in the Arrowhead Framework

4.1 Creating a certificate hierarchy

To fulfill the security requirements set in Sect. 3.1., we propose an application level AA architecture based on the X.509 [5] PKI infrastructure. Each capable Arrowhead System should be provided with a certificate and should therefore have its identity binded to its public key. Every Local Cloud has a Certificate Authority that issues and signs these System certificates. This CA is the root of the trust chain within its Local Cloud and has its own certificate signed by a parent CA. This entity also possesses the Certificate Revocation List defined in the standard: certificates that have been invalidated and therefore not to be accepted.

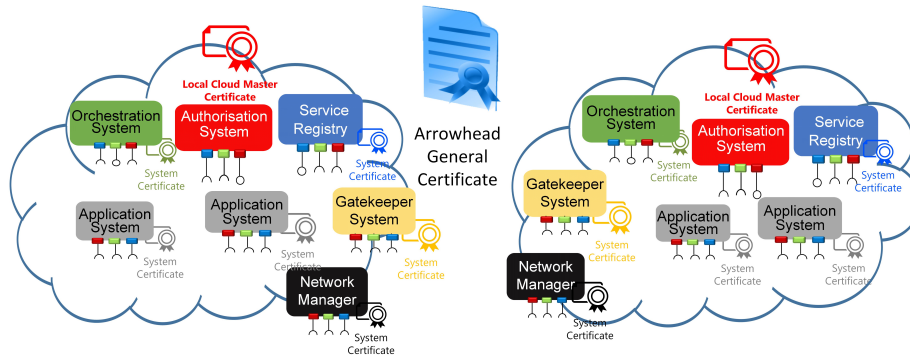


Fig. 4. Arrowhead certificate hierarchy

This chain of trust model fits well into the System hierarchy concept of the Arrowhead framework, as depicted in Fig. 4. A general, master Arrowhead certificate can be signed by a well-known trusted CA (such as Comodo or GlobalSign) and issued to the Arrowhead domain owner (e.g. the project consortia). This administrator entity then can issue and sign Local Cloud certificates for operators in its own application process for establishing new Local Clouds. Within these new Clouds then the Authorization System realizes the CA tasks and owns the cloud certificate. The benefits of this approach:

- The Arrowhead community has oversight over new Local Cloud establishments
- Inter-Cloud interactions are secured by the certificate hierarchy
- Local Cloud owners can issue and revoke certificates for Systems in a deployment procedure
- System bootstrapping and authentication is certificate-secured within its Cloud (can also be bounded to tamper-proof hardware)
- All communications can be signed and/or encrypted with SSL/TLS.

If an Application System in a Local Cloud requires a certificate (e.g. during its deployment procedure), it will have to generate a private-public key pair and submit a Certificate Signing Request (CSR) containing the pair and its identity. There are other fields in a CSR used to verify the identity of the requester. This way the Arrowhead bootstrapping procedure can be automated [14], and augmented with certificate generation.

```
Subject: C=HU, L=Budapest, O=Manufacturer1,  
OU=Fleetcloud1,  
CN=TempSensor1.Car1.FleetCloud1.Manufacturer1.arrowhead.eu
```

Fig. 5. Identity and hierarchical information stored in certificates

We propose a certificate structure to implement the above discussed hierarchy, as depicted on Fig. 5. This format is a customization of the general X.509 certificate and it bounds the identity of the system specifically to a Local Cloud (which makes spoofing attacks more difficult). This approach might seem *l'art pour l'art*, but is a very convenient way of providing authorization, as discussed in the following section.

4.2 Managing access rights

Building on the PKI infrastructure and the certificate distribution capabilities described in sect. 4.1, a new level of security can be added at the servicing level. Since message decryptability only lies with the party owning the appropriate keys, the Core Systems can supervise and provide authorization for Application Systems in a Local Cloud.

There are a number of fundamental questions that describe an automation scenario regarding authentication, authorization or data ownership. The Arrowhead Framework defined Local Clouds that are governed through their own instances of the Core Systems, as described in sect.2. However, full centralization is not desired for multi-stakeholder scenarios.

When Application Systems are introduced in a Local Cloud, there are further bootstrapping tasks besides certificate distribution and the establishment of general trust, a process has been examined in [6]. Such automated bootstrapping procedures require that the operators of the Cloud have properly configured the Core Systems about the new Application System's to-be deployment. These include the followings:

1. Orchestration rules: how it can be orchestrated (e.g. whether it has static servicing connections).
2. Authorization status: where it might be allowed to consume Services from.
3. Further deployment descriptors also have to be configured: e.g. in the QoS Manager or Plant Description Systems.

In Arrowhead, Application Systems generally give up certain autonomy and are to be instructed by the Core Systems on how to operate [14]: which Services they have to consume from which Service Providers. This centralization of control helps managing the System-of-Systems via the Arrowhead Core Systems. However, there are certain cases where Application Systems should retain parts of their autonomy and have the possibility to reject connections even though they have been properly orchestrated. In such exception handling cases re-orchestration will be required.

Moreover, Systems acting as Service Providers should also be able to verify that inbound servicing requests are properly authorized and verified by the Core Systems. To this end, an orchestration process was introduced in [3] and described in Sect. 2 to include process and resource-allocation based restrictions when composing Services.

To this end, we introduce an authorization token building on the ticketing schema that will provide application-level security. A such token is only valid for *one servicing instance*: one Service Consumer is authorized one-time to consume a specific Service from the Service Provider at hand. This information (Consumer-Service-Provider) can be stored in a string and should be decryptable and parseable by Application Systems that require such advanced admission control functionalities. These tokens have the following characteristics:

- It builds on the certificate hierarchy introduced in sect. 4.1;
- Generated by the Authorization System based on access rights during the orchestration process, see Fig. 6;
- This token is then passed on to the Consumer by the Orchestrator;
- It is only decryptable by the Service Provider and its private certificate key;
- It assures that proper orchestration took place and the Consumer is verified to access the Service.
- Based on this, the Service Provider can either accept or reject the connection.

This methodology reduces Provider-side authorization and admission control to a string-based validation of the the Consumer's identity declared in its certificate based on the contents of the encrypted token. Although this requires that Application Systems implement and evaluate this process at every single inbound Service request.

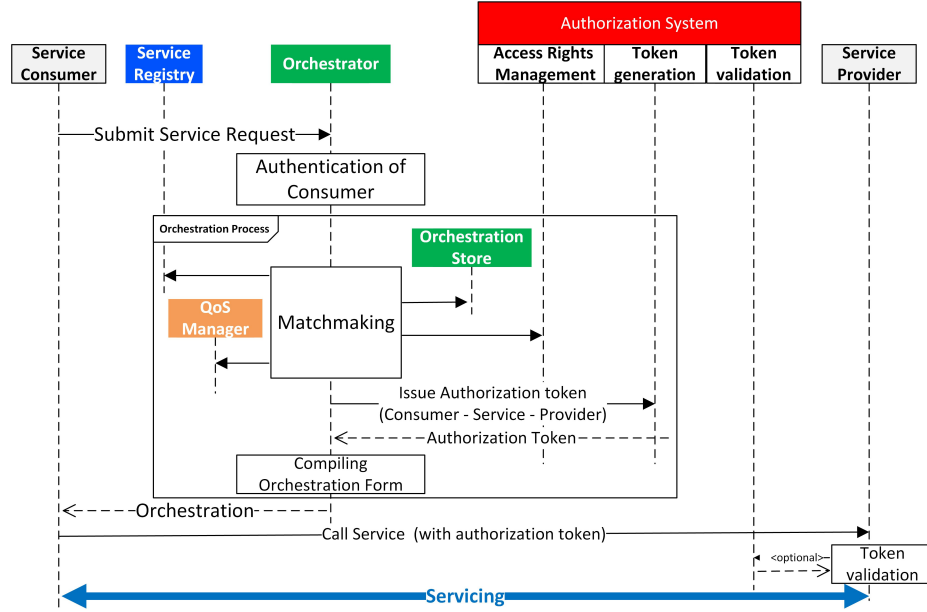


Fig. 6. Authorization tokens generated in the orchestration process

4.3 Integration with the ticketing schema

Extending the works of [7], we propose to integrate the authorization token approach with the ticketing based authentication service in order to provide admission control for all Systems in a Local Cloud: let those be certificate or ticket based.

This requires that the ticket issued for a System should not only authenticate but authorize it as a Service Consumer. In this case, the orchestration response message does not contain a token. Instead, we propose the following process for cases where resource-constrained Consumers only possess the authentication ticket:

1. The orchestration process instructs the ticket-owner Consumer to connect to a specific Service Provider.
2. The Consumer tries connecting with its authentication ticket on an insecure channel instead of using certificates and the token.

3. The Service Provider will try to verify the ticket with the Authorization System.
4. The Authorization System will only respond positively if the Consumer was orchestrated properly before.

This methodology does not rely on the Service Provider’s authentication method. It only requires that the Authorization System stores the result of the last orchestration process for the Consumer (with a validity period) and its currently active authentication ticket. However, when mutual identity verification is required (the Consumer validating a ticket-based Provider), it will require an other round of verification with the Authorization System, but the roles reversed.

4.4 Integration with the inter-Cloud architecture

The authorization token approach is also an adequate solution to the security issues arisen with inter-Cloud servicing requirements, as discussed in Sect. 4.1. This token is generated by the Service Provider’s Cloud in cooperation with the Consumer’s Cloud Core Systems. Therefore, this token is valid even if it was issued for a ”foreigner” Consumer in an inter-Cloud orchestration process: it is completely transparent from the Provider’s point of view (it is still issued by its own Authorization System).

For scenarios where the inter-Cloud data path is realized via tunneling (using the the services of the Network Manager [13]), this token can also help to establish it. This is achieved by orchestrating every System on the data path properly with its own authorization token that validates the partners along the tunneling chain.

5 Summary

This paper presented an enhancement for the security-related capabilities of the Arrowhead Framework. Since any Local Cloud of this automation-specific Internet of Things domain can be subject to extensive threats, providing proper security mechanisms for them is of major interest. After a brief recap on the Arrowhead Framework, we described some security requirements and related issues. The currently used ticketing approach for authorization and authentication has some drawbacks . We can overcome them by using a certificate-based approach.

In order to solve authentication and authorization issues not just within, but in-between Local Clouds, we introduced a few enhancements. First, we applied the concept of certificate hierarchies. In this, every Local Cloud has a Certificate Authority that issues and signs its System certificates. This CA is the root of the trust chain within its Local Cloud, and has its own certificate signed by a parent CA. Second, we described how the certificate and the ticketing approaches can co-exist, and even be translated to one another (within some limitations).

Moreover, we described how to manage access right within Local Clouds by using this approach. For this, we introduced an authorization token, building on

the ticketing schema – but being valid only for one servicing instance. Another advantage of this token approach is that it naturally fits in the inter-Cloud servicing scenarios: the token is valid even if it was issued for a “foreign” cloud.

Future work includes creating a reference implementation of the proposed concept which can be used by any distributed system of systems to manage inter-cloud authentication and authorization.

Acknowledgment

This work is supported by the EU ARTEMIS JU funding, within project ARTEMIS/0001/2012, JU grant nr. 332987 (ARROWHEAD). The authors would like to thank all the Arrowhead partners for the discussions, and would also like to acknowledge the efforts of the Hungarian team who participated in this work: from evopro Hungary and from Budapest University of Technology.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web services. Springer (2004)
2. Blomstedt, F., Ferreira, L.L., Klisics, M., Chrysoulas, C., de Soria, I.M., Morin, B., Zabasta, A., Eliasson, J., Johansson, M., Varga, P.: The arrowhead approach for soa application development and documentation. In: IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society. pp. 2631–2637 (Oct 2014)
3. Csaba Hegedus, Daniel Kozma, G.S.P.V.: Revising the arrowhead framework and its core services for advanced functionalities. IEEE IECON 2016 (2016), to appear
4. Erl, T.: SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl). Prentice Hall PTR, Upper Saddle River, NJ, USA (2007)
5. International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), X.509: Information technology Open Systems Interconnection The Directory: Public-key and attribute certificate frameworks. Recommendation 509 (October 2012), <http://www.itu.int/rec/T-REC-X.509-201210-I/en>
6. Oscar Carlsson, Csaba Hegedus, J.D.P.V.: Organizing iot systems-of-systems from standardized engineering data. IEEE IECON 2016 (2016), to appear
7. Pereira, P.P., Eliasson, J., Delsing, J.: An authentication and access control framework for coap-based internet of things. In: IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society. pp. 5293–5299 (Oct 2014)
8. Plosz, S., Farshad, A., Tauber, M., Lesjak, C., Ruprecht, T., Pereira, N.: Security vulnerabilities and risks in industrial usage of wireless communication. In: Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA). pp. 1–8 (Sept 2014)
9. Plósz, S., Tauber, M., Varga, P.: Information assurance system in the arrowhead project. ERCIM News 2014(97) (2014), <http://ercim-news.ercim.eu/en97/special/information-assurance-system-in-the-arrowhead-project>
10. Ravi, S., Raghunathan, A., Kocher, P., Hattangady, S.: Security in embedded systems: Design challenges. ACM Trans. Embed. Comput. Syst. 3(3), 461–491 (Aug 2004), <http://doi.acm.org/10.1145/1015047.1015049>

11. Roman, R., Zhou, J., Lopez, J.: On the features and challenges of security and privacy in distributed internet of things. *Computer Networks* 57, 2266–2279 (July 2013 2013), <http://www.sciencedirect.com/science/article/pii/S1389128613000054>
12. Varga, P., Blomstedt, F., Lino Ferreira, L., Eliasson, J., Johansson, M., Delsing, J., Martinez de Soria, I.: Making system of systems interoperable - the core components of the arrowhead technology framework. *Journal of Network and Computer Applications* (2016)
13. Varga, P., Hegedus, C.: Service interaction through gateways for inter-cloud collaboration within the arrowhead framework. In: 5th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace and Electronic Systems (Wireless VITAE) (2015)
14. Wiki, T.A.F.: <https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/>