IoT Security Approaches in Oil & Gas Solution Industry 4.0

Cristian TOMA, Marius POPA
Bucharest University of Economic Studies
cristian.toma@ie.ase.ro, marius.popa@ie.ase.ro

Oil and Gas Industry is a very complex one where very specialized equipment, tools and assets are used. The last years, the trend within that industry is to integrate digital technologies in the oil and gas extraction processes as ICT performance has increased and the price has declined. As effect, the productivity of the industry has increased by using digital technologies as IoT, cloud computing, industrial internet, artificial intelligence, block-chain etc. This paper highlights IoT approaches and solutions that could be applied in the oil and gas industry in creating new value in information generated by IoT infrastructures by integration the sensor data, communication channels and data analytics. Large variety of IoT deployments and protocols raises the IoT security assurance way. In this sense, the paper provides security solutions and examples.
Keywords: IoT, IoT Cloud Service, HTTP-REST API, Cyber Security, Cryptographic Security

1 Introduction

Industry 4.0 and Internet of Things – IoT are new terms on hype these days and in the market more solutions appear in the field of these buzzing words. According with Wikipedia [9], "Industry 4.0 is a name for the current trend of automation and data exchange in manufacturing technologies. It includes cyber-physical systems, the Internet of things, cloud computing and cognitive computing. Industry 4.0 is commonly referred to as the fourth industrial revolution."

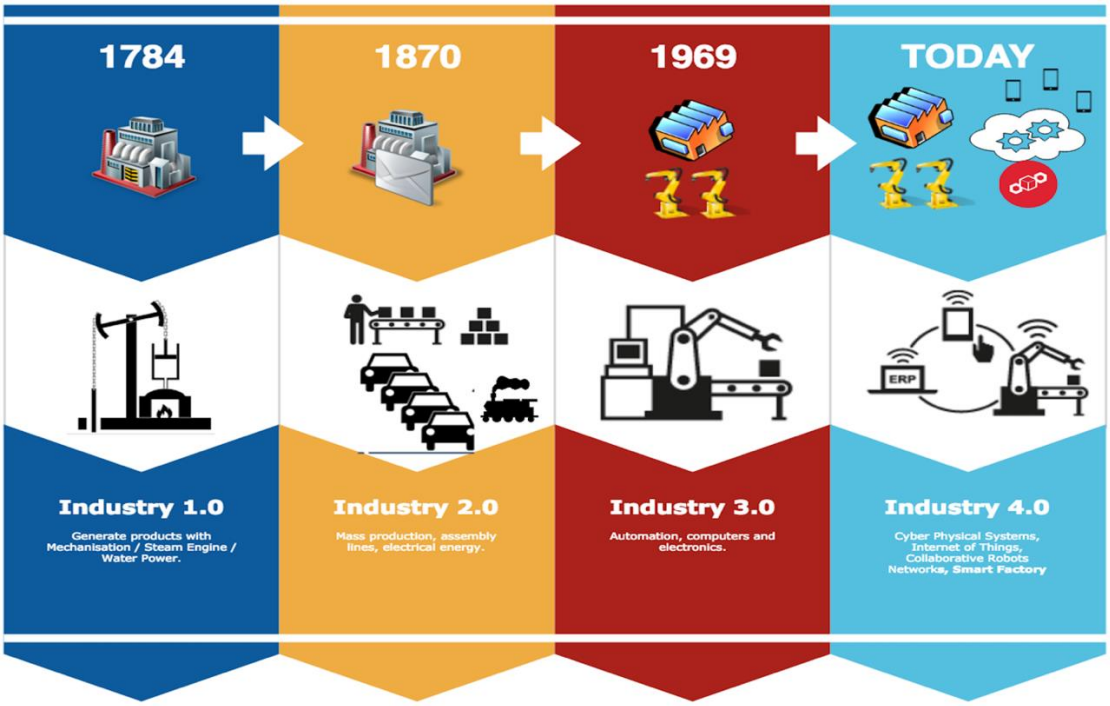

Fig. 1. Industry 4.0 Phases [9]

The Industry 4.0 revolution applied into Oil and Gas field, involves the improvement of the existing SCADA systems and specific filed bus communications protocols (e.g. OPC) with Internet of Things and Cloud computing technologies, in order to provide

DOI: 10.12948/issn14531305/22.3.2018.05

predictive analytics. The predictive analytics help the IoT solution to detect potential downtimes and to operate fixes within the productions systems with zero-down-time approach. In Oil and Gas filed specific equipment and protocols are deployed and the cyber security level should be high in order to use the produced data in cloud computing context. Therefore, for the Oil & Gas Industry 4.0 solution, the following diagram shows the potential Sensors/Actuators Applications [10]:

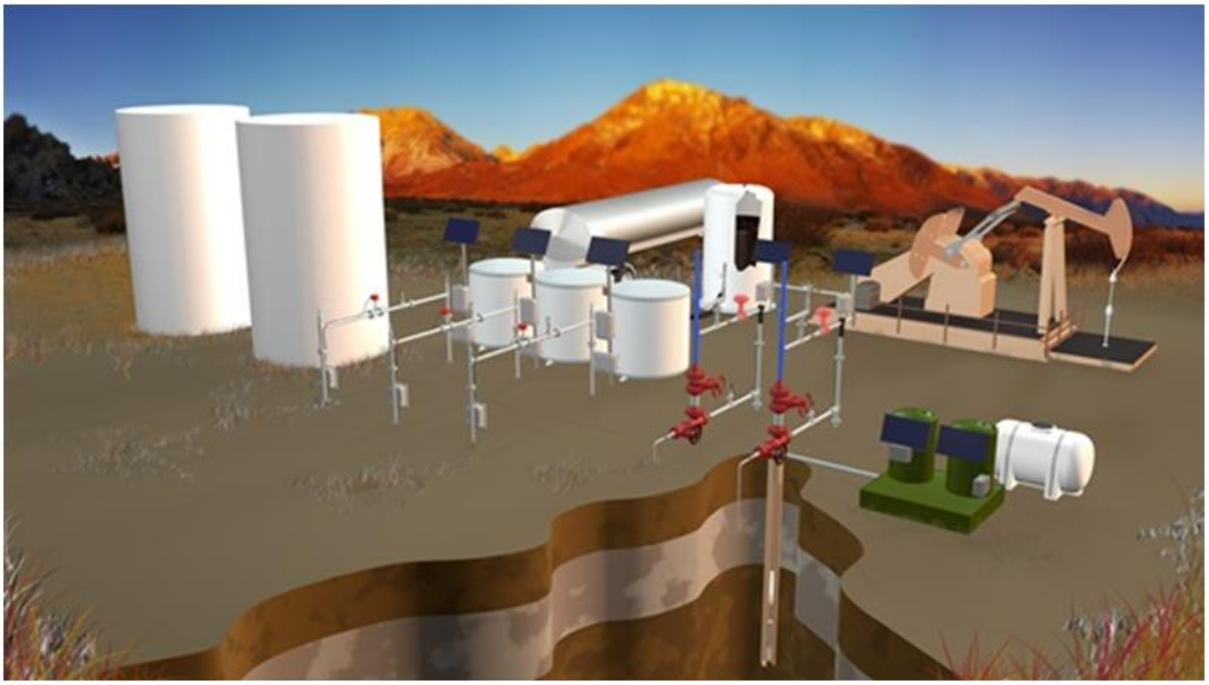

Fig. 2. – Oil and Gas Field Extraction diagram – Copyright [10]

Wide array of sensors (e.g. gas pressure sensors, gas pressure switches and level sensors) to detect pressure, level, and flow in a variety of Oil & Gas sensor applications such as:

- Wellhead Automation (Plunger Lift, Main Line Valve Control, etc.)
- Storage Tank Level Monitoring
- Temperature, Humidity, etc. of the equipment and the environment
- Chemical Injection
- Hydraulic and Lubricating Oil Reservoirs
- Fracturing Truck Monitoring
- Drilling Fluid ("Mud") Tank Monitoring
- Tote Level Monitoring
- Spark Protection in Diesel Fuel Distribution
- CNG (Compressed Natural Gas) Vehicle Conversion

Next section presents the proposed IoT Solution for Oil & Gas Industry use case.

2. IoT Solution for Oil & Gas Industry 4.0

Figure 3 present the system architecture and the overview of the components interaction of the Oracle IoT Cloud with the Matrikon/Honeywell Data-Logger Gateway and the Enterprise Dashboard:

DOI: 10.12948/issn14531305/22.3.2018.05

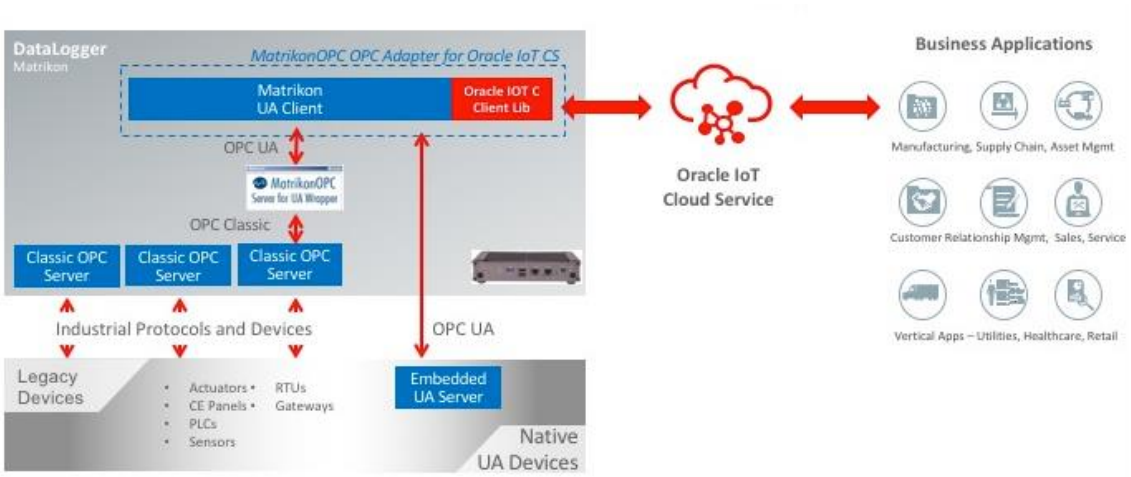

Fig. 3. System Architecture of the prototype for Matrikon/Honeywell Data Logger Gateway integration with Oracle IoT CS

According to the Figure 3, there are the following components:

C1) Distillation Tower Simulator (DTSIM) simulates the behavior of a crude distillation unit, including Temperature and output (Barrels per Hour - BPH) data as it would be captured by sensors on the distillation site – simulator 1.

C2) Oracle IoT Cloud Simulator for generating data into the IoT CS Cloud, in order to have different and various patterns into the tower pressure – simulator 2.

C3) IoT Gateway device which have the following characteristics: OPC, OPC UA, architecture integration. The simulator sends data to an OPC server running on Data-Logger. That data is then sent to Oracle IoT CS via the OPC UA client, also running on the Data-Logger gateway – ot could be a Matrikon/Honeywell Data Logger Gateway.

C4) Oracle Internet of Things Cloud Service (version 16.4) is the key component used for storing the data from the industrial sensors into NoSQL data structures and to interact with different Enterprise applications and/or Mobile/Web dashboards. Also, it is able to process and run SAX and analytics algorithms in a scalable distributed manner within PaaS – Platform as a Service Cloud, but it is able to highlight also the sensors/actuators status within Asset Monitoring Application in terms of SaaS – Software as a Service Cloud, figures 4 (IoT CS PaaS), 5 (IoT CS SaaS AM – Asset Monitoring Application) and 6 (IoT CS PaaS Analytics SPARK Processors).

C5) Mobile Technician Application is displaying to the Service Technicians the emergency notification and the service data. Through this application they can also notify when a repair is complete as figure 7 highlights.

C6) Analytics Web Dashboard is used for having predictive maintenance.

Components Overview and Data Flow

The first two components from the architecture (C1 and C3) are property of Matrikon/Honeywell and the company allowed Oracle to use them into this prototype. The other components (from C2, C4, C5 and C6 inclusive) are property of the Oracle company and the components have been developed within Oracle IoT Cloud organization.

The data collected from the sensors is sent via OPC to an OPC Sever, which is running into a Virtual Machine from the IaaS – Infrastructure as a Service Cloud. The OPC UA Server is running into the Matrikon/Honeywell Data-Logger Gateway as well as the OPC UA Client which has been developed by Oracle with Matrikon OPC UA SDK. The OPC UA client gather the sensors data from the OPC UA Server within the gateway and the gateway, also is running the Oracle IoT CS Client Software – C Windows

DOI: 10.12948/issn14531305/22.3.2018.05

Library [11] for pushing sensors data into the Oracle IoT CS.
Both Oracle IoT Simulator (the virtual devices) and Matrikon/Honeywell Data-Logger are sending data into Oracle IoT CS PaaS Cloud for storing the sensors messages, with a sample rate of 1 message per second, Figure 4.

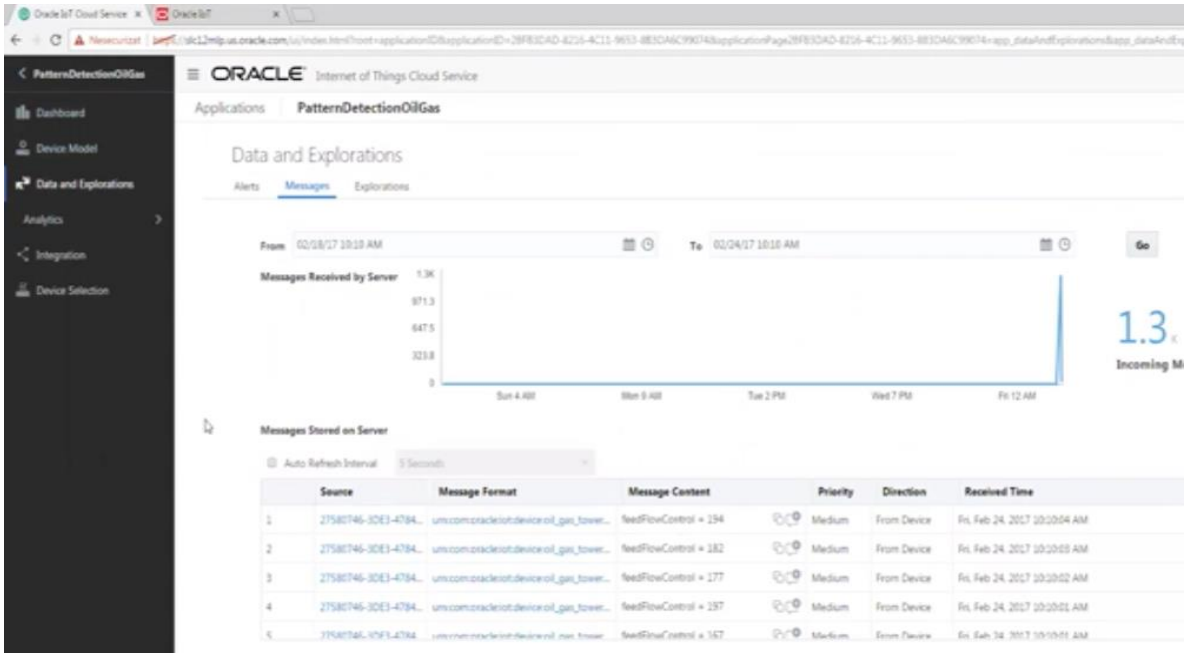

Fig. 4. Oracle IoT CS Platform – SaaS & PaaS Cloud

Oracle IoT CS PaaS si sharing the incoming messages with the Asset Monitoring application from the IoT CS SaaS Cloud in order to have the ability of real time sensors assets supervision and incidents rules configuration:

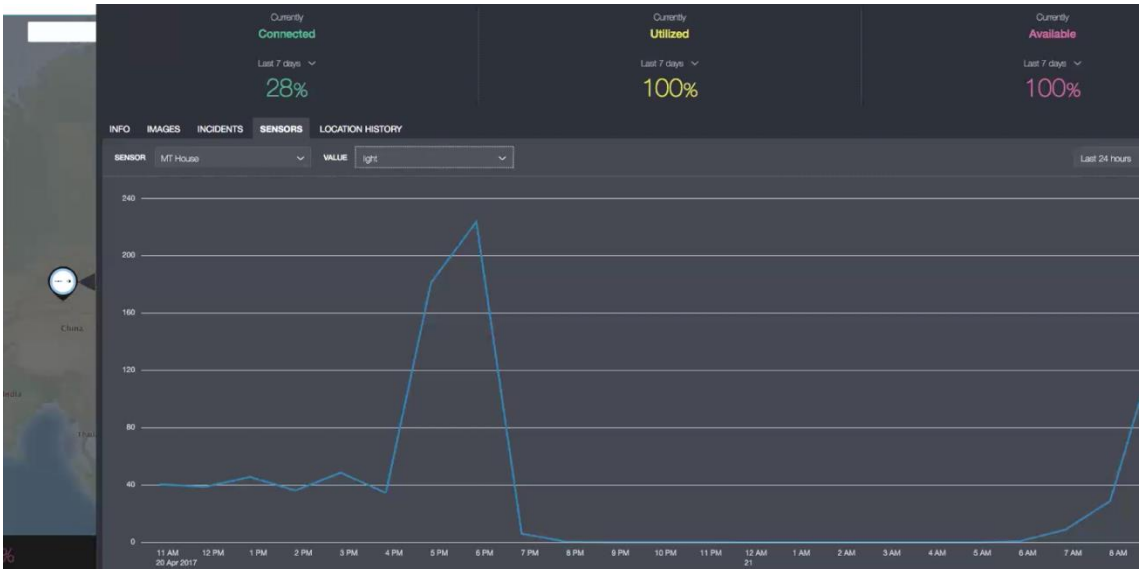

Fig. 5. Oracle IoT CS SaaS AM – Asset Monitoring Application

Once the sensors data are in the cloud, the analytics Apache SPARK processors from Oracle IoT CS PaaS should be developed, configured and parametrized:

DOI: 10.12948/issn14531305/22.3.2018.05

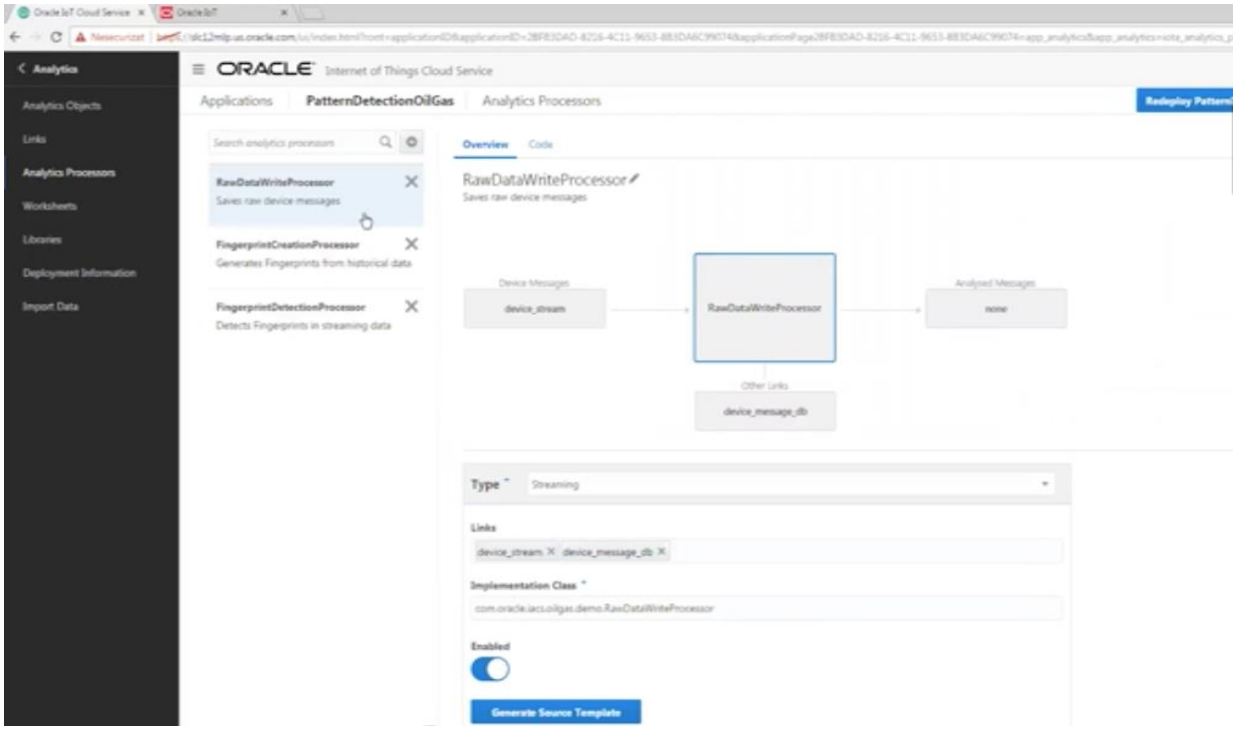

Fig. 6. Oracle IoT CS PaaS Analytics SPARK Processors

This IoT CS cooperates with the Enterprise applications such as C5 and C6, via IoT Enterprise REST API and it can be integrated with different other Clouds such as Oracle MCS – Mobile Cloud Service and BICS – Business Intelligence Cloud Service. The incidents and the key metrics for each distillation tower unit are presented by the Mobile Technician Application:

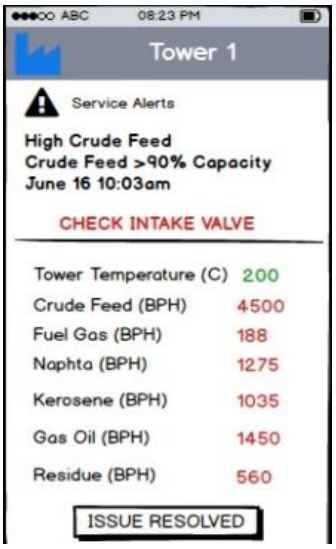

Fig. 7. Oracle Android Mobile Technician App – Monitoring Screen

Once the analytics processors are running within IoT CS, the sixth component Analytics Web Dashboard can bidirectional communicate with the IoT Cloud and display the similarity search for the seal rupture.

3. Security Improvements for the IoT REST APIs

REpresentational State Transfer (REST) facilitates communication between computer systems on the web. Implementation of REST is made by using the next elements [12]:

DOI: 10.12948/issn14531305/22.3.2018.05

- Resources provided by directory structure URIs.
- Structured files (e.g. JSON, XML) as representation of the objects and attributes.
- HTTP methods to send messages across web.
- Session state hold only by the clients.

According to TechTarget, "*A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data*". RESTful API provides high flexibility to software developers to design, implement and maintain applications thanks to stateless and modularity principles of the REST. RESTful APIs are appropriate for web applications, but they are also successfully used in cloud computing and microservice implementations.

Because REST services are used over the web, security must be the main concern and challenge for RESTful API implementers and integrators. According to [14], [16], following technologies and security measures can be used when RESTful APIs are implemented for IoT cloud computing solutions:

- **HTTPS** – it is mandatory because RESTful API transmits over the web sensitive information related to passwords, API keys, JSON Web Tokens (JWT) etc. in order to make the authentication of the edge devices or IoT gateways to the cloud infrastructure. This information must be protected by encryption on transport layer of the computer network infrastructure. HTTPS must be implemented both by client IoT devices and cloud servers.
- **Access control** – it is implemented for each REST endpoint and it is related to authentication and authorization. For effectiveness reasons the access control decisions are taken locally by the REST endpoint and the access tokens are issued by a centralized Identity Provider server. There are different protocols to be used in order to manage the access control to the cloud infrastructure.
- **JSON Web Tokens (JWT)** – represents JSON data structures used by RESTful API for access control. JWT must be protected by encryption or message authentication code (MAC) to avoid lack of integrity. JWT is RFC document describing the requirements, constraints and security considerations, and providing examples on those [15]. JWT must be validated against its integrity and contained claims.
- **API Keys** – they are used by the endpoint to create HTTP requests to server. API keys are unique byte streams and usually they are included into the HTTP request header or in the URI itself. Still, the second approach will expose the key within the browser history and API logs at server level. API keys represent a security REST implementation for the public cloud infrastructure where there is no a strict access control to it. Hence, endpoint accesses are limited for those having API keys. Also, some access filters are applied depending on client endpoint category.
- **Restrict HTTP methods** – not all endpoints have access to all RESTful services provided by the cloud infrastructure. This is implemented by restricting some HTTP REST methods or creating endpoint blacklists.
- **Input validation** – it should be always implemented by a RESTful service and a validation response is sent back to the client endpoint. Validation is implemented by checking different characteristics of the request data: length, range, format and type, using of strong data types, strings created by regular expressions, unexpected or illegal content, HTTP request size, number of failed input validations per unit time, considering security issues of the parser used for incoming requests.
- **Validate content types** – the REST message types are provided by the HTTP request header. The message content must match the type provided by the request header. Hence, following security measures must be implemented by the RESTful API for both request and

DOI: 10.12948/issn14531305/22.3.2018.05

response messages: rejecting the request where the content type is missing or unexpected, not exposing unintended content type, not included into the response the content type of the request, matching the right content type according to the response body.

- **Management endpoints** – they must use stronger authentication mechanisms to avoid their exposure to uncontrolled access via Internet. Also, other security measures to do that include use of firewalls, appropriate computer network setup, access control lists. It is preferable to avoid management endpoints via Internet.
- **Error handling** – it is implemented inappropriately it could be a source of relevant information about RESTful service. Therefore, the RESTful API has not to reveal details by responding with generic message errors. Also, technical details have not to be passed back to the client.
- **Audit logs** – they could be a security measure to avoid possible attacks by logging the token validation errors. Also, audit logs must be sanitized by audit log injection attacks.
- **Security headers** – they contain the right information about the content type to be correctly interpreted by the browser. Such kind of headers are `X-Content-Type-Options` and `X-Frame-Options`.
- **Sensitive information in the HTTP headers** – RESTful API operates with sensitive information over the Internet (passwords, JWTs, API keys). Therefore, it has to prevent credential leakage because the sensitive information becomes available over web (browser history, web server logs etc.) where the RESTful API operates. Sensitive data are placed within the request body or request header.
- **HTTP return code** – the RESTful API must return the correct code for the available HTTP response code list.

A RESTful API must consider all security issues related to computer network communication via web and it has to implement the appropriate security measures to protect sensitive data and to avoid attacks over the clients and cloud infrastructure by using information exchanged between them via REST requests and response.

Examples of IoT cloud infrastructures within the current IoT industry to be considered are Amazon Web Services (AWS) IoT and Oracle Internet of Things Cloud Service.

AWS IoT is an infrastructure that provides bi-directional communication between IoT devices and AWS Cloud. Collected data are used to monitor and control various IoT devices by implementing services, applications and interfaces between IoT data repository and different tools running on different components within the AWS cloud infrastructure [18].

From the security point of view, AWS IoT shares responsibility between the IoT clients and IoT Cloud by provisioning the device with secure items like certificates to authenticate it and secure the data sent to the AWS Cloud. By other hand, the AWS Cloud implements security policies to send data to the devices or other AWS services.

Interactions with the IoT clients is customized by using some AWS IoT interfaces as follows [18].

- AWS Command Line Interface (AWS CLI) – set of commands for different platforms (Windows, macOS, Linux) allowing creation and management of different items within AWS IoT infrastructure.
- AWS IoT API – allows building of IoT applications using HTTP/HTTPS requests. These IoT applications are built programmatically.
- AWS SDKs – wraps AWS IoT API to allow cross-platform IoT application development.
- AWS IoT Device SDKs – allow IoT application development running on IoT devices.

In order to enable the bi-directional communication and AWS IoT cloud services,

DOI: 10.12948/issn14531305/22.3.2018.05

the AWS IoT clients must be authenticated within the cloud and they have to encrypt the all traffic to AWS IoT infrastructure over the Transport Layer Security (TLS). TLS ensures confidentiality of the application protocols (MQTT, HTTP) supported by the AWS IoT [18].

Depending of what kind of AWS IoT interface is used to interact with the AWS IoT cloud services, the available IoT authentication schemes are [18]:

- X.509 certificates – authentication scheme used by IoT devices.
- IAM users, groups, and roles – used by AWS CLI and web and desktop IoT applications.
- Amazon Cognito identities – used by the IoT mobile applications.
- Federated identities – used by web and desktop IoT applications.

AWS IoT devices use X.509 certificates issued by a Certification Authority (CA) to be trusted. The certificates allow usage of asymmetric cryptography with benefits regarding the storage of the private key within the IoT device. The private key does not leave the IoT device therefore that is a stronger authentication scheme over client credentials or bearer token usages. For an IoT device, its certificate could be revoked and replaced by other new certificate. That implies a management of the certificates within entire IoT device infrastructure. AWS IoT infrastructure provides the services and tools to do the following operations [18]:

- Create and register an AWS IoT certificate.
- Register a CA certificate.
- Register a device certificate.
- Activate or deactivate a device certificate.
- Revoke a device certificate.
- Transfer a device certificate to another AWS account.
- List all CA certificates registered to your AWS account.
- List all device certificates registered to your AWS account.

Two kinds of certificates are creating within AWS IoT [18]:

- AWS IoT device certificate – created by the AWS IoT services and tools as AWS IoT console (options available in graphical user interface) and AWS IoT CLI (commands to create X.509 certificate from issued public key or certificate signing request).
- AWS IoT user certificate – a CA certificate must be registered with AWS IoT in order to sign device certificates. The CA certificate could be created by using OpenSSL tool. In order to use users' certificates, the following stages must be followed [18]
  1. Creating CA certificate – two steps to be accomplished [18]:
     1.1 Generate a key pair by using OpenSSL tool

```
openssl genrsa -out rootCA.key 2048
```

1.2 Generate CA certificate using the private key from the keypair:

```
openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out
rootCA.pem
```

  2. Registering the CA certificate – accomplished by following the steps [18]:
     2.1 Get a registration code from AWS IoT to be used as `Common Name` of the private key verification certificate:

```
aws iot get-registration-code
```

2.2 Generate a key pair for the private key verification certificate:

DOI: 10.12948/issn14531305/22.3.2018.05

```
openssl genrsa -out verificationCert.key 2048
```

2.3 Create a Certificate Signing Request (CSR) for the private key verification certificate. The registration code is used for the `Common Name` field:

```
openssl req -new -key verificationCert.key –out
verificationCert.csr
```

As result, some additional information is asked from the user side in the command prompt:

```
Country Name (2 letter code) [AU]:
State or Province Name (full name) []:
Locality Name (for example, city) []:
Organization Name (for example, company) []:
Organizational Unit Name (for example, section) []:
Common Name (e.g. server FQDN or YOUR name)
[]:XXXXXXXXXXXXMYREGISTRATIONCODEXXXXXX
Email Address []:
```

2.4 Use the CSR to create a private key verification certificate:

```
openssl x509 -req -in verificationCert.csr -CA rootCA.pem -CAkey
rootCA.key -CAcreateserial -out verificationCert.pem -days 500 -sha256
```

2.5 Register the CA certificate with AWS IoT:

```
aws iot register-ca-certificate --ca-certificate
file://rootCA.pem --verification-cert
file://verificationCert.pem
```

2.6 Activate the CA certificate:

```
aws iot update-ca-certificate --certificate-id xxxxxxxxxx
--new-status ACTIVE
```

3. Creating IoT device certificate – according to [18], the following steps must be accomplished:
   3.1 Generate a key pair:

```
openssl genrsa -out deviceCert.key 2048
```

3.2 Create a CSR for the device certificate:

```
openssl req -new -key deviceCert.key -out deviceCert.csr
```

As result, the following information is asked at command prompt:

```
Country Name (2 letter code) [AU]:
State or Province Name (full name) []:
Locality Name (for example, city) []:
```

DOI: 10.12948/issn14531305/22.3.2018.05

```
Organization Name (for example, company) []:
Organizational Unit Name (for example, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

3.3 Create a device certificate from the CSR:

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key -
CAcreateserial -out deviceCert.pem -days 500
-sha256
```

3.4 Register an IoT device certificate:

```
aws iot register-certificate --certificate-pem
file://deviceCert.pem --ca-certificate-pem
file://rootCA.pem
```

3.5 Activate the IoT device certificate:

```
aws iot update-certificate --certificate-id xxxxxxxxxx
--new-status ACTIVE
```

4. Registering IoT device certificate – a CA certificate must be used to sign IoT device certificate. If there are more CA certificate, the AWS IoT user has to specify the right CA certificate to be used in order to sign the IoT device certificate. There are two approaches of registering ioT device certificates [18]:

4.1 Registering the IoT device certificate manually – following AWS CLI statement is used to do that:

```
aws iot register-certificate --certificate-pem
file://deviceCert.crt --ca-certificate-pem
file://caCert.crt
```

4.2 Registering the IoT device certificate when a first connect attempt occurs – following operations must be performed for automatic registrations:

4.2.1 Set automatic registration status flag of the CA certificate when `update-ca-certificate` API is used:

```
aws iot update-ca-certificate --cert-id caCertificateId
--new-auto-registration-status ENABLE
```

4.2.2 Alternatively, set automatic registration for `register-ca-certificate` API use:

```
aws iot register-ca-certificate --ca-certificate
file://rootCA.pem --verification-cert
file://privateKeyVerificationCert.crt
--allow-auto-registration
```

DOI: 10.12948/issn14531305/22.3.2018.05

The MQTT topic is used for a message publishing when an automatic registration of an IoT device certificate is done or an IoT device certificate status is `PENDING_ACTIVATION`. That MQTT topic is:

```
aws/events/certificates/registered/caCertificateID
```

`caCertificateID` is the ID of the CA certificate used to issue the IoT device certificate.

5. Deactivating CA certificate – useful operation to prevent issuing of IoT device certificates by using a compromised CA certificate. Deactivation is made by the following command [18]:

```
aws iot update-ca-certificate --cert-id certificateId
--new-status INACTIVE
```

6. Revoking IoT device certificate – useful operation for a compromised IoT device certificate. `update-certificate` API is used to do that [18]:

```
aws iot update-certificate --cert-id certificateId
--new-status REVOKED
```

Having an active IoT device certificate within the AWS IoT infrastructure, the AWS IoT user can attach IoT devices (called things in AWS) to that certificate. IoT devices are registered using the AWS IoT console where the graphical user interface of that tool provide management and monitoring options for IoT devices, certificates and policies. Also, all above operations regarding the CA and IoT device certificates can be implemented by using the same AWS IoT console tool.

The second example aiming IoT REST API implementation is Oracle Internet of Things (IoT) Cloud Service REST APIs by enabling execution of functions and services by sending REST calls to Oracle IoT Cloud Service [17]. In order to enable execution of functions and services provided by Oracle IoT Cloud Service, following operations must be accomplished [17]:

- Device model definition in Oracle IoT Cloud Service.
- Device registration.
- Device activation.
- Message sending from device to Oracle IoT Cloud Service.
- Message processing from Oracle IoT Cloud Service.

Connections between devices (Oracle IoT Cloud Service clients) and Oracle IoT Cloud Service are securely established by using Verisign certificates.

Oracle IoT Cloud Service users must be registered into accounts and their credentials are used to establish the privilege layer of the user. Users use the Certificate Authority certificate to make the authentication. Once authentication being done, OAuth2 access tokens are used to authorize the user to execute operations over IoT resources according to its privilege layer. Oracle IoT Cloud Service users obtain OAuth2 access token by using Oracle IoT Cloud Service REST API after their authentication. The REST scheme to get the access token is [17]:

Method: **`POST`**
Server Path: **`/iot/api/v2/oauth2/token`**

DOI: 10.12948/issn14531305/22.3.2018.05

OAuth2 is an authorization framework through a limited access is provided to the user account over HTTP. Requests for OAuth2 access token use the scope attributes with following values [17]:

- oracle/iot/activation – it is used when the activation IoT REST API is called.
- Empty scope – it is used for non-activation IoT REST API calls.

Oracle IoT Cloud Service REST API uses JWT as authentication mechanism, containing the signature generated by specific cryptographic operations to prove the private key holding. OAuth2 tokens are included into JWT to be securely exchanged between IoT clients and IoT Cloud Service infrastructure.

The JWT structure consists of [17]:

- Header – it is composed by two items: type of token (JWT) and hashing algorithm (HS256, RS256).
- Payload – it contains claims for authorization:
  - **iss** – the issuer (Device Activation ID, Device Endpoint ID).
  - **exp** – expirations time (in seconds).
  - **aud** – audience (`oracle/iot/oauth2/token`).
- Signature – it is generated by considering the items: the encoded header, the encoded payload, a key, the algorithm specified in the header and the signature generation pattern as:

```
Algorithm(base64UrlEncode(header) + "." + base64UrlEncode(payload), key)
```

In [17], some examples to get the activation token and message token are provided to guide the Oracle IoT Cloud Service REST API user how to create requests for such kind of tokens. The requests are sent by using cURL utility tool in the command line and the server response is provided also in the command line.

The cURL command for getting the activation token looks like [17]:

```
curl -X POST
     -H 'Accept:application/json'
     -H 'Content-Type: application/x-www-form-urlencoded'
     --data "grant_type=client_credentials&
               client_assertion_type=urn:ietf:params:oauth:client-
                  assertion-type:jwt-bearer&
               client_assertion=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc
                  3MiOiIwLVlRRlEiLCJleHAiOjE0Njc3NjYyNjMsImF1ZCI6Im9yYWNsZ
                  S9pb3Qvb2F1dGgyL3Rva2VuIn0.AU9eYChN9EOupzyLoWf0AMZ0QiK4d
                  nmpA1n5tb_kSSw&
               scope=oracle/iot/activation"
      http://instance-identitydomain.iot.us.oraclecloud.com
             /iot/api/v2/oauth2/token
```

Considerations regarding the previous cURL example:

- X – It indicates the REST method used (POST).
- H – It considers the next parameter as part of the header request.
- data – Content to be sent as part of the request.
- Last parameter considers the Oracle IoT Cloud Service instance together with the REST path (`iot/api/v2/oauth2/token`).

Therefore, the following structures are considered to send an activation token request [17]:

- Request Header:

```
Content-Type: application/x-www-form-urlencoded
Accept: application/json
```

DOI: 10.12948/issn14531305/22.3.2018.05

- Request Form Parameters:

```
grant_type=client_credentials
client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-
    bearer
client_assertion=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiIwLVlRRlE
    iLCJleHAiOjE0Njc3NjYyNjMsImF1ZCI6Im9yYWNsZS9pb3Qvb2F1dGgyL3Rva2VuIn0.A
    U9eYChN9EOupzyLoWf0AMZ0QiK4dnmpA1n5tb_kSSw
scope=oracle/iot/activation
```

The `client_assertion` parameter is built on the following JWT information [17]:

```
Header: {
  "typ": "JWT",
  "alg": "HS256"
}
Payload: {
 "iss": "6e5e3e593bcf-1c79",
 "exp": 1467751899,
 "aud": "oracle/iot/oauth2/token"
}
```

The server response for the above request looks like [17]:

Response Header

```
HTTP/1.1 200 OK
Content-Type: application/json
```

Response Body

```
{
   "expires_in":3600000,
   "token_type":"Bearer",
   "access_token":"677b4afa55236110825ae0a3d38275ad"
}
```

The cURL command for getting the message token looks like [17]:

```
curl -X POST
     -H 'Accept:application/json'
     -H 'Content-Type: application/x-www-form-urlencoded'
     --data "grant_type=client_credentials&
             client_assertion_type=urn:ietf:params:oauth:client-
                  assertion-type:jwt-bearer&
             client_assertion=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3
                  MiOiJBQUFBQUFRMEZSMEEtQ0UiLCJhdWQiOiJvcmFjbGUvaW90L29hd
                  XRoMi90b2tlbiIsImV4cCI6MTQ2OTQ3NDE5M30.oDiGi13Xw1ZJJ3iM
                  DOSpZLN1NNfTVF8wJuWy2sMzcKKVIUbULeFCUoyd8SZiJBXMQDd4n_d
                  RtF5vq7or3kGGiKOiWkS3V-r1qqqL0-
                  o02Hh2tQ4AOHvwMRbEQOGE3lA7doNRd9sr7BR5zQGu4yahCh5581Q_4
                  B5gV-ZN65gbdt4&
             scope="
     http://instance-identitydomain.iot.us.oraclecloud.com
            /iot/api/v2/oauth2/token
```

The previous cURL components are [17]:

- Request Header:

```
Content-Type: application/x-www-form-urlencoded
```

DOI: 10.12948/issn14531305/22.3.2018.05

```
Accept: application/json
```

- Request Form Parameters:

```
grant_type=client_credentials
client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-
    bearer
client_assertion=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJBQUFBQUF
    RMEZSMEEtQ0UiLCJhdWQiOiJvcmFjbGUvaW90L29hdXRoMi90b2tlbiIsImV4cCI6MTQ2O
    TQ3NDE5M30.oDiGi13Xw1ZJJ3iMDOSpZLN1NNfTVF8wJuWy2sMzcKKVIUbULeFCUoyd8SZ
    iJBXMQDd4n_dRtF5vq7or3kGGiKOiWkS3V-r1qqqL0-
    o02Hh2tQ4AOHvwMRbEQOGE3lA7doNRd9sr7BR5zQGu4yahCh5581Q_4B5gV-ZN65gbdt4
scope=
```

The `client_assertion` parameter is built on the following JWT information [17]:

```
Header: {
  "typ": "JWT",
  "alg": "RS256"
}
Payload: {
 "iss": "0-AECA",
 "exp": 1469474193,
 "aud": "oracle/iot/oauth2/token"
}
```

The private key of the device is obtained during the activation procedure and it has the following content [17]:

```
MIICWwIBAAKBgQDdlatRjRjogo3WojgGHFHYLugdUWAY9iR3fy4arWNA1KoS8kVw33cJibXr8
bvwUAUparCwlvdbH6dvEOfou0/gCFQsHUfQrSDv+MuSUMAe8jzKE4qW+jK+xQU9a03GUnKHkk
le+Q0pX/g6jXZ7r1/xAK5Do2kQ+X5xK9cipRgEKwIDAQABAoGAD+onAtVye4ic7VR7V50DF9b
OnwRwNXrARcDhq9LWNRrRGElESYYTQ6EbatXS3MCyjjX2eMhu/aF5YhXBwkppwxg+EOmXeh+M
zL7Zh284OuPbkglAaGhV9bb6/5CpuGb1esyPbYW+Ty2PC0GSZfIXkXs76jXAu9TOBvD0ybc2Y
lkCQQDywg2R/7t3Q2OE2+yo382CLJdrlSLVROWKwb4tb2PjhY4XAwV8d1vy0RenxTB+K5Mu57
uVSTHtrMK0GAtFr833AkEA6avx20OHo61Yela/4k5kQDtjEf1N0LfI+BcWZtxsS3jDM3i1Hp0
KSu5rsCPb8acJo5RO26gGVrfAsDcIXKC+bQJAZZ2XIpsitLyPpuiMOvBbzPavd4gY6Z8KWrfY
zJoI/Q9FuBo6rKwl4BFoToD7WIUS+hpkagwWiz+6zLoX1dbOZwJACmH5fSSjAkLRi54PKJ8TF
UeOP15h9sQzydI8zJU+upvDEKZsZc/UhT/SySDOxQ4G/523Y0sz/OZtSWcol/UMgQJALesy++
GdvoIDLfJX5GBQpuFgFenRiRDabxrE9MNUZ2aPFaFp+DyAe+b4nDwuJaW2LURbr8AEZga7oQj
0uYxcYw==
```

The signature is generated by applying the algorithm SHA256withRSA for the payload of the header.

The server response for the message token request looks like [17]:

Response Header

```
HTTP/1.1 200 OK
Content-Type: application/json
```

Response Body

```
{
   "expires_in":3600000,
   "token_type":"Bearer",
   "access_token":" 6074b571c671fe3cd548a8e668042187"
}
```

DOI: 10.12948/issn14531305/22.3.2018.05

Such kind of previous REST APIs for IoT have been developed for various technological frameworks and platforms and the process is ongoing to spread the applicability and usability of the IoT.

4. Conclusions

IoT covers a huge range of industries and use cases by implementing a huge number of devices and network communication protocols. Hence, applying security best practices during IoT deployments has become a critical requirement of IoT environments and infrastructures. The challenge is to synchronize the multitude of protocols used by IoT deployments related to infrastructure, device discovery and management, data protocols and semantic representations, communication / transport layer, infrastructure and data security.

Possible solutions to achieve secure IoT deployments include JavaCard technology and investigations on those are part of future researches. One reason to choose JavaCard is its maturity to support security requirements within IoT and Industry 4.0 field.

Acknowledgement

This paper presents results obtained within the PN-III-P1-1.2-PCCDI-2017-0272 ATLAS project ("Hub inovativ pentru tehnologii avansate de securitate cibernetică / Innovative Hub for Advanced Cyber Security Technologies "), financed by UEFISCDI through the PN III – "Dezvoltarea sistemului national de cercetare-dezvoltare", PN-III-P1-1.2-PCCDI-2017-1 program.

Parts of this paper were communicated and presented within "The 17th International Conference on Informatics in Economy" IE 2018, Section "Mobile-Embedded and Multimedia Solutions", May 17 – 20, 2018, "Alexandru Ioan Cuza" University, Iaşi, Romania.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies protocols and applications", Proc. IEEE Commun. Surveys Tuts., vol. 17, no. 4, pp. 2347 – 2376, 4th Quart. 2015, https://ieeexplore.ieee.org/document/7123563/

[2] I. Andrea, C. Chrysostomou, and G. Hadjichristofi, "Internet of Things: Security vulnerabilities and challenges", Proc. IEEE Symp. Comput. Commun. (ISCC), Jul. 2015, pp. 180-187

[3] M. Doinea, C. Boja, L. Batagan, C. Toma, and M. Popa, "Internet of Things Based Systems for Food Safety Management", Informatica Economică, vol. 19, no. 1, 2015, pp. 87-97

[4] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, J. Henry, *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things*, Cisco Press, 2017, http://www.ciscopress.com

[5] A. Minteer, *Analytics for the Internet of Things (IoT): Intelligent analytics for your intelligent devices*, Packt Publishing, 2017, http://www.packtpub.com

[6] M. Popa, C. Toma, C. Boja, and A. Zamfiroiu, "Privacy and Security in Connected Vehicles Ecosystems", Informatica Economică, vol. 21, no. 4, 2017, pp. 29-40

[7] B. Russell, and D. van Duren, *Practical Internet of Things Security*, Packt Publishing, 2016, http://www.packtpub.com

[8] D. Slama, F. Puhlmann, J. Morrish and R. M. Bhatnagar. *Enterprise IoT*, O'Reilly Inc. Publishing House, 2016

[9] Wikipedia Industry 4.0: https://en.wikipedia.org/wiki/Industry_4.0

[10] Gems Sensors & Controls – Oil and Gas Applications: http://www.gemssensors.com/Markets/Oil-and-Gas

[11] Oracle IoT CS libraries: http://www.oracle.com/technetwork/indexes/downloads/iot-client-libraries-2705514.html

[12] Understanding REST from Spring: https://spring.io/understanding/REST

[13] RESTful API and Taxonomy: http://searchmicroservices.techtarget.com /definition /RESTful-API
[14] OWASP Security Cheat Sheet: https://www.owasp.org/index.php/ REST_Security_Cheat_Sheet
[15] JSON Web Token (JWT) RFC: https://tools.ietf.org/html/rfc7519
[16] RESTful API Security: https://dzone.com/articles/restful-api-security
[17] Oracle IoT Cloud: https://docs.oracle.com/en/cloud/paas/iot-cloud/iotrq
[18] Amazon AWS IoT Developer Guide: https://docs.aws.amazon.com/iot/latest/ developerguide
[19] MatrikonOPC Data Connectivity Devices: https://www.matrikonopc.com/data-connectivity-devices/

Cristian TOMA has graduated from the Faculty of Cybernetics, Statistics and Economic Informatics, Economic Informatics specialization, within Bucharest University of Economic Studies in 2003. He has graduated from the BRIE master program in 2005 and PhD stage in 2008. In present, he is associate professor at Economic Informatics and Cybernetics Department and he is member in research structures such as ECO-INFOSOC. Since the beginning - 2005 - he is scientific secretary of IT&C Security Master Program from Bucharest University of Economic Studies and since 2006, he is in the editorial board of the SECITC – The Inter-national Conference on Security for Information Technology and Communications and JMEDS – Journal of Mobile, Embedded and Distributed Systems. His research areas are in: distributed and parallel computing, mobile applications, smart card programming, e-business and e-payment systems, network security, computer anti-viruses and viruses, secure web technologies and computational cryptography. He is teaching in Department of Economic Informatics and Cybernetics, and IT&C Security Master program. He has published 3 books and over 50 papers in indexed reviews and conferences proceedings.

Marius POPA has graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 2002. He holds a PhD diploma in Economic Cybernetics and Statistics. He joined the staff of Bucharest University of Economic Studies, teaching assistant in 2002. Currently, he is Associate Professor in Economic Informatics field and branches within Department of Economic Informatics and Cybernetics at Faculty of Economic Cybernetics, Statistics and Informatics from Bucharest University of Economic Studies. He is the author and co-author of 9 books and over 140 articles in journals and proceedings of national and international conferences, symposiums, workshops in the fields of data quality, software quality, informatics security, collaborative information systems, IT project management, software engineering.

DOI: 10.12948/issn14531305/22.3.2018.05