Annales UMCS Informatica AI XII, 4 (2012) 69–81
DOI: 10.2478/v10065-012-0032-0

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

On the modelling of Kerberos protocol in the Quality of Protection Modelling Language (QoP-ML)

Bogdan Księżopolski[1,2*], Damian Rusinek[1†], Adam Wierzbicki[2‡]

[1] *Institute of Computer Science, Maria Curie-Sklodowska University*
pl. M. Curie-Sklodowskiej 5, 20-031 Lublin, Poland
[2] *Polish-Japanese Institute of Information Technology*
Koszykowa 86, 02-008 Warsaw, Poland

Abstract – The security modelling of IT systems is a very complicated task. One of the issues which must be analysed is the performance of IT systems. In many cases the guaranteed security level is too high in relation to the real threats. The overestimation of security measures can decrease system performance. The paper presents the analysis of Kerberos cryptographic protocol in terms of quality of protection performed by Quality of Protection Modelling Language (QoP-ML). The analysis concerns the availability attribute. In the article the Kerberos protocol was modelled and the QoP analysis of two selected versions was performed.

1 Introduction

During modelling the IT security of the organization one has to consider system performance and financial costs. System security is guaranteed by using different types of security mechanisms [**1**]. Security analysts must decide which security measures should be used for the system protection and whether the selection is sufficient. The usage of strongest security mechanisms can lead to the overestimation of security measures which causes an unreasonable increase in the system load [**2, 3**]. A better solution is to adjust the security measures to the required level of protection. Such an approach can be achieved by means of the Quality of Protection systems where the security measures are evaluated according to their influence on the system security.

*bogdan.ksiezopolski@umcs.lublin.pl
†damian.rusinek@gmail.com
‡adamw@pjwstk.edu.pl

1.1 Related Work

In the literature the Quality of Protection (QoP) models are described in the following articles [**4, 5, 6, 7, 8, 9, 10, 11**]. S. Lindskog and E. Jonsson attempted to extend the security layers in a few Quality of Service (QoS) architectures [**6**]. Unfortunately, the descriptions of the methods are limited to the confidentiality of the data and based on different configurations of the cryptographic modules. C. S. Ong et al. in [**8**] present the QoP mechanisms, which define security levels depending on security parameters. These parameters are as follows: key length, block length and contents of an encrypted block of data. P. Schneck and K. Schwan [**10**] proposed an adaptable protocol concentrating on the authentication. By means of this protocol, one can change the version of the authentication protocol which finally changes the parameters of the asymmetric and symmetric ciphers. Y. Sun and A .Kumar [**11**] created QoP models based on the vulnerability analysis which is represented by the attack trees. The leaves of the trees are described by means of the special metrics of security. These metrics are used for describing individual characteristics of the attack. In the article [**4**] B. Księżopolski and Z. Kotulski introduced mechanisms for adaptable security which can be used for all security services. In this model the quality of protection depends on the risk level of the analysed processes. A. Luo et al [**7**] present the quality of protection analysis for the IP multimedia systems (IMS). This approach includes the IMS performance evaluation using Queuing Networks and Stochastic Petri Nets. E. LeMay et al [**5**] create the adversary-driven, state-based system security evaluation, the method which quantitatively evaluates the strength of system's security. In the article [**9**] D. C. Petriu et al present the performance analysis of security aspects in the UML models. This approach takes as an input the UML model of the system designed by the UMLsec extension [**12**] of the UML modelling language. This UML model is annotated with the standard UML Profile for schedualability, performance and time and then was analysed for performance. In the article [**13**] B. Księżopolski introduced the Quality of Protection Modelling Language (QoP-ML) which provides the modelling language for making abstraction of cryptographic protocols that put emphasis on the details concerning quality of protection.

1.2 QoP-ML

The QoP-ML [**13**] is the Quality of Protection Modelling Language by means of which one can abstract all operations executed during the flow of cryptographic protocol. The QoP-ML introduces the multilevel [**14**] protocol analysis that extends the possibility of describing all possible states of the cryptographic protocol. Every single operation defined by the QoP-ML is described by the security metrics which evaluate the impact of this operation on the overall system security. When the security impact is defined for all actions executed in the protocol, then one can set the protection level for all analysed systems. The assumption of the quality of protection analysis based on the QoP-ML is to be fully automatic, where the analysis time is an important factor. The

analysis engine of the modelled protocol is the part of the core system. In the paper [**13**] the syntax, semantics and algorithms of the QoP-ML are presented.

In the paper we would like to present the case study of the quality of protection modelling by means of the QoP-ML. For illustration of the QoP analysis process we chose one of the most popular cryptographic protocols - Kerberos [**15**].

2 Case Study: Kerberos protocol

In this section we are going to present the case study of QoP modelling of the Kerberos cryptographic protocol [**15**]. We are analysing the two versions of the protocol. In the first one the symmetric key is generated by Trusted Third Party and in the second one the key is generated by both sides: A and B. The flows of both version of the protocol are realized in four steps and the schemes are presented in Figs 1 and 2.

Notation for Figs 1 and 2
TTP - Trusted Third Party;
A - side A;
B - side B;
$ticket_B = (K, A, L)_{K_{BTTP}}$ - ticket for the side B;
$authenticator_1 = (A, T_A)_K$ - authenticator 1;
$authenticator_2 = (A, T_A, K')_K$ - authenticator 2;
K - new generated symmetric key;
L - lifetime of the ticket $ticket_B$;
T_X - timestamp from the local clock of side X;
N_X - the nonce of the X.
K' - subkey of the key K generated by the site A;
K'' - subkey of the key K generated by the site B.

1. $A \rightarrow TTP : A, B, N_A$
2. $A \leftarrow TTP : ticket_B, (K, N_A, L, B)_{K_{ATTP}}$
3. $A \rightarrow B : ticket_B, authenticator_1$
4. $A \leftarrow B : (T_A)_K$

Fig. 1. The protocol flow of the Kerberos (simplified) - version 1

1. $A \rightarrow TTP : A, B, N_A$
2. $A \leftarrow TTP : ticket_B, (K, N_A, L, B)_{K_{ATTP}}$
3. $A \rightarrow B : ticket_B, authenticator_2$
4. $A \leftarrow B : (T_A, K'')_K$

Fig. 2. The protocol flow of the Kerberos (simplified) - version 2

The flows presented in Figs 1 and 2 are the simplified, fifth version of the Kerberos protocol. This protocol is fully analysed and described in [**15**]. In the next section it is briefly described according to the introduced notation used in Figs 1 and 2.

Step 1:
A Client A generates a nonce N_A and sends to the TTP the following information: his identification (A), generated nonce (N_A) and the identification of the side B with which he wants to perform the key exchange process.

Step 2:
The TTP generates a new symmetric key K and defines the lifetime of ticket ($ticket_B$). After this, the TTP generates the ticket ($ticket_B$). The ticket contains: generated symmetric key K, identifier of side A (A) and the lifetime of the ticket L. This data is encrypted with the symmetric key K_{BTTP} shared between TTP and B. Next, the TTP creates the message addressed to side A containing: the symmetric key K, identifier of side B (B), the lifetime of the ticket L and the previously received nonce N_A and encrypts it with the symmetric key K_{ATTP} shared between TTP and A. Finally, the TTP sends these two encrypted data to the A.

Step 3:
The side A decrypts the second encrypted data using the key K_{ATTP}. After this it verifies the integrity of the received nonce N_A with the one sent in step 1. In the next operation, side A generates one of the messages $authenticator_1$ or $authenticator_2$ depending on the selected version of the protocol. Both these messages contain identification of A and the current timestamp T_A. In the second version, A additionally generates its subkey K' and includes it in the message. Next the prepared message is encrypted with the received key K. Finally A sends the received ticket ($ticket_B$) and the authenticator ($authenticator_1$ or $authenticator_2$) to the side B.

Step 4:
The side B decrypts the received ticket ($ticket_B$) using the shared key K_{BTTP} and obtains the key K. After this the side B decrypts the authenticator and verifies the following information: the identifiers A in the ticket and the authenticator, the timestamp in the authenticator and the ticket lifetime L. If the verification of all components is positive then, depending on the selected version, the side B either encrypts only the received timestamp T_A (in version 1) or generates its own subkey K'' and encypts both the received timestamp T_A and generated key K''. In both cases the encryption is performed using the key K. The encrypted data is sent to the side A. Next, the side A receives the data and decrypts it with the key K. The decrypted timestamp T_A is used for authentication of the side B.

The QoP analysis process includes the five steps: protocol modelling, security metrics definition, precess instantiation, QoP-ML processing and QoP evaluation [**13**]. The following section describes these steps during modelling of the Kerberos protocol.

2.1 Protocol modelling

In the first step one has to model all operations required in the Kerberos protocol [**15**]. These operations are generally described in the protocol flow scheme (Figs 1 and 2). In the article we present one level analysis where only the cryptographic operation will be considered. The QoP analysis can refer to different security attributes and each of them must be proceeded according to the special algorithms. In the article which introduces QoP-ML [**13**] the algorithm referring to the availability security attribute [**16**] is presented. Thus the Kerberos protocol will be analysed according to this security attribute.

The protocol modelling step includes the four operations: function defining, equation defining, channels defining and protocol flow description.

Functions

For modelling of the Kerberos protocol we defined the functions which refer to the cryptographic operations required in the protocol. These functions are presented below. In the round bracket the description of these functions is presented.

```
fun id(); (creating id of a side)
fun skey()[Availability:bitlength, algorithm];(compute symmetric key)
fun nonce()[Availability:bitlength, algorithm];(compute new nounce)
fun lifetime();(compute ticket lifetime)
fun enc(data,key)[Availability:bitlength, algorithm, mode];(encrypt the data)
fun dec(data,key)[Availability:bitlength, algorithm, mode];(decrypt the data)
fun time();(timestamp generating)
fun newstate(state);(state of the protocol)
fun finished(); (finished state of the protocol)
```

Equations

After defining the functions one can describe the relations between them.

```
eq dec(enc(data,K),K) = data (symmetric encryption/decryption)
```

Channels

In the presented example we define two synchronous channels.

```
channel ch1,ch2(0);
```

Protocol flow

The last and the most important operation during the modelling process is abstracting the protocol flow. In the presented case study we analyse two versions of the Kerberos protocol. In the first one, the exchanged key K is generated by the Trusted Third Party (TPP) and used by both sides. In the second version, the key generated by

TPP is used to encrypt the messages which include the subkeys K' and K'' of boths sides while the exchanged key will be derived from these subkeys.

To analyse it, one does not have to design these two versions separately, but they can be abstracted in one protocol flow. While defining the protocol instantiation one can specify the parameters typical of specific versions of Kerberos.

The operations inside the processes are numbered owing to which one can easily refer to them during the QoP analysis. These operations are numbered independently (locally) inside every single process.

In the following section the high hierarchy processes will be described in detail.

```
host A (rr)(*)
{
 #K_ATTP=skey()[256,LinuxPRNG];

 process A1(ch1,ch2)
 {
 1.IDA = id();
 2.IDB = id();
 3.NA = nonce()[256,LinuxPRNG];
 4.M1 = (IDA, IDB, NA);
 5.out(ch1:M1);
 6.in(ch1:TicketB,Y);
 7.M4=dec(Y,K_ATTP)[256,AES,CBC];
 8.TA=time();

 subprocess Av1(*)
  {
  1.M5=(IDA,TA);
  2.K=M4[0];
  3.authenticator1=enc(M5,K)[256,AES,CBC];
  4.out(ch2:TicketB, authenticator1);
  }
```

```
 subprocess Av2(*)
  {
  1.KA=skey()[256,LinuxPRNG];
  2.M5=(IDA,TA,KA);
  3.K=M4[0];
  4.authenticator2=enc(M5,K)[256,AES,CBC];
  5.out(ch2:TicketB, authenticator2);
  }

 9.in(ch2:Z);
 10.M7=dec(Z,K)[256,AES,CBC];
 11.TArec=M7[0];
 12.if(TArec==TA){
     13.status=newstate(finished());
   } else {
     14.stop;
   }
 }
}
```

Host A

The processes inside the `Host A` are scheduled according to the round robin (`rr`) algorithm where the quantum of time is defined in QoP-ML as the single operation. All communication channels are accepted by this process `(*)`.

Firstly, the operation, which is executed before the process `Host A` starts, is defined (# operator). This is `K_ATTP` - defining the symmetric key shared by the Hosts A and TTP.

process A1

Process `A1` can communicate with the other processes through the channels `ch1` and `ch2`. In the first operation the id of side A, which starts the protocol, is created and is assigned to the variable `IDA`. Next, Host A creates the id of the side `IDB` which he wants to authorise and with whom he wants to exchange the new key. In the third operation the nonce is generated and assigned to the variable `NA`. Additionally, the

qop parameters are defined (according to the defined function description), there is the bits length - `256` and the algorithm used in the Linux Ubuntu System named - `Linux PRNG`. In the next operation the message `M1` is created containing the following data: `IDA, IDB, NA`. After this, the message `M1` is sent through the channel `ch1` and Host A starts to listen on the channel `ch1`.
When the messages are received on the channel `ch1`, they are assigned to the variables `TicketB` and `Y`. The variable `Y` is decrypted using the key `K_ATTP` and the result is assigned to variable `M4`. Next, timestamp `TA` is created and one of the subprocesses (`Av1` or `Av2`) is run (depending on the selected version in the process instantiation). These subprocesses differ in only one operation - the new symmetric subkey generation (function - `skey`) which distinguishes the analysed Kerberos protocol versions. As the output of these subprocesses the message containing `TicketB` and one type of the `authenticator` are created and sent through the channel `ch2`. In the next operation the `Host A` starts to listen on the channel `ch2`. When a message is received, it is assigned to the variable `Z` and decrypted using the key `K`. The result of this operation is assigned to the variable `M7` and the ingredient containing the timestamp `TArec` is compared with the previously sent timestamp `TA`. When these timestamps are the same, then the authorization process is finished sucessfully and the state of the protocol flow is changed into the finished `newstate`. When the timestamp `TArec` is not equal to `TA`, the protocol is stopped `stop`.

subprocess Av1
The subprocess `Av1` can communicate with other processes through all channels `(*)`. In the first operation the message `M5` is created which contains the created timestamp `TA` and identification of side A `IDA`. After this, the message `M5` is encrypted with the key `K` received in the message `Y`. The result of this encryption is assigned to the variable `authenticator1`. Next, both the `TicketB` (received with the message `Y`) and `authenticator1` are sent through the channel `ch2`.
subprocess Av2
The subprocess `Av2` can communicate with other processes through all channels `(*)`. This subprocess differs from the subprocess `Av1` in one step only. In this process the symmetric key `KA` is created and joined to the message `M5`. As a result, the new authenticator is created `authenticator2` and sent with `TicketB` through the channel `ch2`.

```
host TTP (rr)(*)
{
 #K_BTTP=skey()[256,LinuxPRNG];
 #K_ATTP=skey()[256,LinuxPRNG];

 process TTP1(ch1,ch2)
 {
 1.in(ch1:X);
 2.K=skey()[256,LinuxPRNG];
 3.L=lifetime();
 4.IDA=X[0];
 5.M2=(K,IDA,L);
 6.TicketB=enc(M2,K_BTTP)[256,AES,CBC];
 7.NA=X[2];
 8.IDB=X[1];
 9.M3=(K,NA,L,IDB);
 10.M3E=enc(M3,K_ATTP)[256,AES,CBC];
 11.out(ch1:TicketB,M3E);
 }
}
```

Host TTP

In this Host the processes are scheduled according to the same algorithm as in `Host A` and all communication channels are accepted by this process `(*)`. As in `Host A`, firstly the operations executed before `Host TTP` starts the main process are defined (# operator). There are: `K_BTTP` - the secret key shared between `B` and `TTP` and `K_ATTP` - the secret key shared between `A` and `TTP`.

process TPP1

Process `TPP1` can communicate with other processes through the channels `ch1` and `ch2`. At the beginning the process is waiting for incoming message which is assigned to the variable `X`. When the message is received the process generates a new symmetric key and stores it in the variable `K`. Next the lifetime of ticket is stored in `L` and identification of the side that started protocol is retrived from the message `X` and assigned to `IDA`. The process `TTP1` can now create message `M2` including: the generated key `K`, retrived id `IDA` and lifetime `L` and encrypt it using the key `K_BTTP` to obtain `TicketB` destined for the side `B`. In the next four operations the `process TTP1` preapres the returning message for the side `A`. This message includes: the nonce retrived in message `X`, generated key `K`, lifetime `L` and identification of side `B` obtained from message `X`. Next the message is encryptrd with the key `K_ATTP` shared between `Host A` and `Host TTP` and both `TicketB` and the encrypted message `M3E` are sent through the channel `ch1`.

```
host B (rr)(*)
{
 #K_BTTP=skey()[256,LinuxPRNG];

 process B1(ch2)
 {
 1.in(ch2:TicketB,authenticator);
 2.M1=dec(TicketB,K_BTTP)[256,AES,CBC];
 3.M2=dec(authenticator,K)[256,AES,CBC];

 subprocess Bv1(*)
  {
  1.TA=M2[1];
  2.K=M1[0];
  3.M6=enc(TA,K)[256,AES,CBC];
  4.out(ch2:M6);
  }

 subprocess Bv2(*)
  {
  1.TA=M2[1];
  2.K=M1[0];
  3.KB=skey()[256,LinuxPRNG];
  4.M6b=(TA,KB);
  5.M6=enc(M6b,K)[256,AES,CBC];
  6.out(ch2:M6);
  }
 }
}
```

Host B

In `Host B` processes are sheduled in the same way as in prevoius hosts and all communication channels are accepted by this process `(*)`. It has one previously computed value stored in the variable `K_BTTP`. It is the symmetric key shared between `Host B` and `Host TTP`.

process B1

The process `B1` can communicate with other processes using the channel `ch2`. Firstly,

the process is waiting for two messages (stored in `TicketB` and `authenticator` variables) on the channel `ch1`. Next, both messages are decrypted. `TicketB` is decrypted using the key `K_BTTP` because it comes from `Host TTP` and was forwarded by `Host A`. The result of decryption is stored in the `M1` variable. The second message `authenticator` is decrypted with the key `K` generated by `TTP` and the result is stored in the variable `M2`.
Further, one of the subprocesses (`Bv1` or `Bv2`) is executed depending on the selected version in the process instantion.

subprocess Bv1
The subprocess `Bv1` can communicate with other processes through all channels `(*)`. Firstly, it obtains the timestamp of side `A` (`TA`) and the key generated by `TTP` (`K`) from the retrived messages. Next, the subprocess sends the timestamp `TA` encrypted with the key `K` through the channel `ch2`. Then the protocol is finished for `Host B`.

subprocess Bv2
The second version of the protocol flow is similar at the beginning. It also obtains the timestamp of side `A` (`TA`) and the key generated by `TTP` (`K`), but later the flow is different from the previous subprocess. Instead of sending only timestamp `TA`, the subprocess `Bv2` generates the subkey `K''` of side `B` stored in the variable `KB`. Next, the subprocess sends both: timestamp `TA` and generated key `KB` encrypted with the key `K` through the channel `ch2`. Then the protocol is finished for `Host B`.

2.2 Security metrics definition

When modelling the protocol, the designer needs to define the security metrics for all functions connected with each security attribute which he wants to test. In the presented case study we test the availability of two different flows of Kerberos protocol. Hence, we need metrics for all functions that may affect the availability. We have checked the execution times of operations used in the Kerberos protocol that may be used (ie. nonce/key generation, encryption).

Many security metrics may be obtained from benchmarks present in both, official hardware specifications and literature [**17, 18**]. Some of them can be found in the specialized scientific articles, such as performance characteristics of the S-blocks [**19, 20**] . However, some metrics may depend on the hardware on which the protocol is executed [**21**]. Therefore, in our case study we have used commonly applied software to compute metrics so that everyone can very easily compute them on his host. For encrypting and decrypting we have used openssl program with speed library [**22**]. For the functions which generates the nonce and keys we prepared our own software described in [**13**].

```
metrics
{
  conf(host1)
  {
```

```
    CPU = Intel Core i7-3930K 3.20GHz;
    CryptoLibrary = openssl 0.9.8o-5ubuntu1.2;
    OS = Ubuntu 11.04 64-bit;
  }
   data(host1)
  {
    primhead[nr][bit length][algorithm][mode][function][Av:time(mspb)];
    primitive[1][256][AES][CBC][enc][0.0000000049];
    primitive[2][256][AES][CBC][dec][0.0000000049];
  }
  data+(host1:1)
  {
    primhead[nr][bit length][algorithm][function][Av:time(ms)];
    primitive[1][256][LinuxPRNG][nonce][0.0025];
    primitive[2][256][LinuxPRNG][skey][0.0025];
  }
  set host A(host1:1);
  set host B(host1:1);
  set host TTP(host1:1);
}
```

2.3 Process instantiation

During the process instantiation one can define the versions of the modelled protocol. In the presented example we set two versions of the Kerberos protocol, the first version with one key generation and the second one with subkeys generated by both sides (`A` and `B`). In these versions three high hierarchy processes are executed: `Host A`, `Host B` and `Host TTP`.

In `version 1` inside the process `Host A` the process `A1` is executed (function - `run`) with the subprocess `Av1`. Inside the process `Host B` the process `B1` is executed with the subprocess `Bv1`. The process `Host TTP` is run the same way in both cases. The Kerberos protocol versions can be modelled by defining the subprocess which will be executed in the specific protocol instantiation.

```
version 1
{
  run host TTP
  {
    run TTP1(*)
  }
  run host B
  {
    run B1(Bv1)
  }
  run host A
  {
    run A1(Av1)
  }
}
```

```
version 2
{
  run host TTP
  {
    run TTP1(*)
  }
  run host B
  {
    run B1(Bv2)
  }
  run host A
  {
    run A1(Av2)
  }
}
```

The second version of the Kerberos protocol differs from the first one in the key generation. The key is generated from two subkeys generated by both sides `A` and `B`. This flow was modelled as the subprocesses `Av2` and `Bv2` in the processes `A1` and `B1`, respectively.

2.4 QoP-ML processing and QoP evaluation

The final step in the QoP analysis process is QoP-ML processing and QoP evaluation which can investigate the influences of the security mechanisms for ensuring security attributes. In the presented case study we focus on the availability of the cryptography protocol. The QoP-ML processing of this security attribute should be prepared according to the algorithms presented in the article [**13**]. Unfortunately, realization of this security attribute is a complex task because it refers to configuration of the whole teleinformatic infrastructure in which the protocol is realized. In the article the QoP evaluation is focused on one level of security aspect which refers to the cryptographic algorithm. Based on these algorithms we calculated the total execution time (T_{Total}) of the two analysed versions of the Kerberos protocol. For the first version of the protocol the $T_{Total} = 0.0050004018\ s$. In the second the $T_{Total} = 0.010000441\ s$. The execution time for the first version of the protocol is 50% shorter than in the second version.

3 Conclusions

The aim of this study was to present a new language QoP-ML [**13**] and show how to perform an QoP analysis of cryptographic protocols. A full, multi-level cryptographic protocol analysis is very complex and exceeds the opportunity to be presented in this article. The study contains two selected versions of the Kerberos protocol and only cryptographic algorithms were taken into account. The QoP-ML modelling language allows to analyse protocols in terms of different security attributes, this paper presents an analysis in terms of availability. Based on the algorithms presented in [**13**] we calculate the total protocol runtime for two versions of the Kerberos protocol.

The main feature of QoP-ML is that the cryptographic protocol can be analysed on different levels of security analysis. Owing to that the QoP analysis can take into consideration any factors which influence the overall system security. Another main feature of QoP-ML is that one can define the security metrics of the used operations in the analysed protocol.

Acknowledgements

Research partially supported by the grant "Reconcile: Robust Online Credibility Evaluation of Web Content" from Switzerland through the Swiss Contribution to the enlarged European Union

References

[1] ISO/IEC 27002:2005; Information technology - Security techniques - Code of practice for information security management (2005).

[2] Ksiezopolski B., Kotulski Z., Szalachowski P., Adaptive approach to network security, Communications in Computer and Information Science 158 (2009): 233.

[3] Ksiezopolski B., Kotulski Z., Szalachowski P., On QoP method for ensuring availability of the goal of cryptographic protocols in the real-time systems, European Teletraffic Seminar (2011): 195.

[4] Ksiezopolski B, Kotulski Z., Adaptable security mechanism for the dynamic environments, Computers & Security 26 (2007): 246.

[5] LeMay E., Unkenholz W., Parks D., Adversary-Driven State-Based System Security Evaluation, In Workshop on Security Metrics - MetriSec (2010) .

[6] Lindskog S., Modeling and Tuning Security from a Quality of Service Perspective. PhD dissertation, Department of Computer Science and Engineering, Chalmers University of Technology, Goteborg, Sweden (2005).

[7] Luo A., Lin Ch., Wang K., Lei L., Liu Ch., Quality of protection analysis and performance modeling in IP multimedia subsystem. Computers Communications 32 (2009): 1336.

[8] Ong C.S., Nahrstedt K., Yuan W., Quality of protection for mobile applications, In IEEE International Conference on Multimedia & Expo (2003): 137.

[9] Petriu D. C., Woodside C. M., Petriu D. B., Xu J., Israr T., Georg G., France R., Bieman J. M., Houmb S. H., Jürjens J., Performance Analysis of Security Aspects in UML Models, In Sixth International Workshop on Software and Performance (2007).

[10] Schneck P., Schwan K., Authenticast: An Adaptive Protocol for High-Performance, Secure Network Applications, Technical Report GIT-CC-97-22 (1997).

[11] Sun Y., Kumar A., Quality od Protection(QoP): A quantitative methodology to grade security services, In 28th confrence on Distributed Computing Systems Workshop (2008): 394.

[12] Jürjens J., Secure System Development with UML, Springer (2007).

[13] Ksiezopolski B., QoP-ML: Quality of Protection modelling language for cryptographic protocols, Computers & Security 31(4) (2012): 569.

[14] Theoharidou M., Kotzanikolaou P., Gritzalis S., A multi-layer Criticality Assessment methodology based on interdependencies, Computers & Security 29 (2010): 643.

[15] Neuman C., Ts'o T., Kerberos: An Authentication Service for Computer Networks, IEEE Communications 32 (9) (1994): 33.

[16] ISO/IEC 27001:2005. Information technology – Security techniques – Information security management systems – Requirements (2005).

[17] Rusinek D., Ksiezopolski B., Influence of CCM, CBC-MAC, CTR and stand-alone encryption on the quality of transmitted data in the high-performance WSN based on Imote2 Annales UMCS Informatica AI XI (3) (2011): 117.

[18] Szalachowski P., Ksiezopolski B., Kotulski Z., CMAC, CCM and GCM/GMAC: advanced modes of operation of symmetric block ciphers in the Wireless Sensor Networks, Information Processing Letters 110 (2010): 247.

[19] Grocholewska-Czurylo A., Cryptographic properties of modified AES-like S-boxes, Annales UMCS Informatica AI XI (2) (2011): 37.

[20] Grocholewska-Czurylo A., Chmiel K., Stoklosa J., Involutional block cipher for limited resources, IEEE Globecom (2008).

[21] Jaquith A., Security Metrics: Replacing Fear, Uncertainty, and Doubt, Addison-Wesley (2007).

[22] Openssl Project: http://www.openssl.org/
Jürjens J., Tools for Secure Systems Development with UML. International Journal on Software Tools for Technology Transfer 2007; 9:527-544.

Lambrinoudakis C., Gritzalis S., Dridi F., Pernul G., Security requirements for e-government services: a methodological approach for developing a common PKI-based security policy 2003. Computers & Security 2003; 26:1873-1883.