On the Efficiency Modelling of Cryptographic Protocols by Means of the Quality of Protection Modelling Language (QoP-ML)

Bogdan Ksiezopolski, Damian Rusinek, Adam Wierzbicki

On the efficiency modelling of cryptographic protocols by means of the Quality of Protection Modelling Language (QoP-ML)

Bogdan Ksiezopolski[1,2], Damian Rusinek[2], and Adam Wierzbicki[1]

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

Abstract. The problem of efficiency in the IT systems is now widely discussed. One of the factors affecting the performance of IT systems is implementation and maintaining a high level of security. In many cases the guaranteed security level is too high in relation to the real threats. The implementation and maintenance of this protection level is expensive in terms of both productivity and financial costs.
The paper presents the analysis of TLS Handshake protocol in terms of quality of protection performed by the Quality of Protection Modelling Language (QoP-ML). The analysis concerns efficiency.

1 Introduction

In the design of teleinformatic systems, the analyst must consider the system performance. One of the aspects which influence system performance is its security. System security is represented by means of the security attributes [7] which precisely define the system security requirements. Finally, the security requirements are guaranteed by using different types of security measures [5] which are realized by means of cryptographic protocols. Cryptographic protocols can be run with different parameters which affect system performance [6]. Security and efficiency analysts must decide which security measures and efficiency parameters should be used for the protocol realization and whether the selection is sufficient. Such an approach can be achieved by means of the Quality of Protection systems where the security measures and efficiency factors are evaluated according to their influence on the system security and performance.

1.1 Related Work

In the literature the security adaptable models are introduced as the Quality of Protection (QoP) models [4, 5, 8–10, 12, 13]. S.Lindskog and E.Jonsson attempted to extend the security layers in a few Quality of Service (QoS) architectures [9]. Unfortunately, the descriptions of the methods are limited to the

confidentiality of the data and based on different configurations of the cryptographic modules. Y.Sun and A.Kumar [13] 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 [5] B.Ksiezopolski 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.Lua et al [10] presents the quality of protection analysis for the IP multimedia systems (IMS). This approach presents the IMS performance evaluation using Queuing Networks and Stochastic Petri Nets. E. LeMay et al [8] created the adversary-driven, state-based system security evaluation, the method which quantitatively evaluates the strength of systems security. In the article [12] D.C. Petriu et al present the performance analysis of security aspects in the UML models. This approach takes as an input a UML model of a system designed by the UMLsec extension [2] of the UML modelling language. This UML model is annotated with the standard UML Profile for schedualability, performance and time and is then analysed for performance. In the article [4] B.Ksiezopolski 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. The intended use of QoP-ML is to represent the series of steps which are described as a cryptographic protocol. The QoP-ML introduced the multilevel [15] protocol analysis that extends the possibility of describing the state of the cryptographic protocol.

In the QoP-ML the time of analysis is an important factor. The analysis engine is the part of the core system so the analysis can be performed in real time systems. In the article we present the new method and construction for modelling the efficiency of cryptographic protocols by means of the QoP-ML. The advantage of this approach is that the efficiency can be modelled simultaneously with security attributes and can be done for real time systems. For illustration of the QoP analysis process we choose one of the most popular cryptographic protocols - TLS [16]. In the article [4] the syntax, semantics and algorithms of the QoP-ML are presented.

2 Case Study: TLS Handshake protocol

In this section we are going to present the case study of QoP modelling of TLS cryptographic protocol. We are analysing the two versions of the protocol, the first one with compression of the transmitted data and the second one without compression. The flow of the TLS Handshake protocol is realized in five steps and the scheme is presented in Fig. 1

Notation for Fig. 1:
PK_X - the public key of the X;
ID_{SX} - the id of the session X;
SK_X - the secret key of the X;
V_{TLS} - the version of TLS protocol;
$V_{TLS}(SET)$ - the established version of TLS protocol;
Cip_X - the available cipher suite for the session X;
Com_X - the compression method for the session X;
CA - the certificate authority;
$PK_X(cert) = (PK_X, ID_X, T)_{SK_{CA}}$ - the certificate of the X;
T - the timestamp;
ID_X - the id of the site X;
N_X - the nonce of the X.

1. $C \rightarrow S : ID_{S1}, V_{TLS}, Cip_1, Com_1, N_1$
2. $S \rightarrow C : V_{TLS}(SET), Cip_1(SET), Com_1(SET), N_2, PK_S(cert), Done(S)$
3. $C \rightarrow S : (K_1)_{PK_S(cert)}, ReadyEnc(C), Fin(C)$
4. $S \rightarrow C : ReadyEnc(S), Fin(S)$
5. $C \rightarrow S : (Data)_{K_1}$

Fig. 1. The protocol flow of the TLS Handshake protocol

The version of the TLS protocol presented in the Fig. 1 is the standard one. This protocol is fully analysed and described in [16].

The QoP analysis process includes the five steps: protocol modelling, security metrics definition, precess instantiation, QoP-ML processing and QoP evaluation. The following subsections describe these steps during modelling of the TLS protocol.

2.1 Protocol modelling

In the first step one has to model all operations required in the TLS Handshake protocol. These operations are generally described in the protocol flow scheme (Fig.1). The complete QoP analysis of cryptographic protocols should contain many aspects like: the use of any security mechanism (not only cryptographic operation), key management operations, security policy management, legal compliance, implementation of the protocol and cryptographic algorithms, communication process, data storage and other factors which influence the system security. These aspects can be modelled by means of QoP-ML but this process is very complex and its presentation must be described on many pages. Therefore in the article we present one level analysis where only the efficiency factors which refers to cryptographic operation will be considered. The QoP analysis can refer to different security attributes and each of them must be proceeded according to the dedicated algorithms. In case of efficiency analysis we are focused on protocol time analysis which can be performed by means of availability algorithm which is introduced in [4].

The protocol modelling step includes the four operations [4]: function defining, equation defining, channels defining and protocol flow description.

Functions

For modelling of the TLS protocol we define the functions which refer to the cryptographic operations and affecting protocol efficiency. These functions are presented below. In the round bracket the description of these functions is presented.

```
fun id() (creating id of a session);
fun date() (create timestamp);
fun Vlist() (TLS versions list);
fun Clist() (creating ciphers list);
fun Comlist() (creating compression method list);
fun data() (prepare data);
fun set(X) (setup the X parameter);
fun info(X) (creating information message about X);
fun ReadyEncClient();
fun FinClient();
fun ReadyEncServer();
fun FinServer();
fun Done();
fun sk(id)[Av.:bitlength, algorithm] (compute secret key for id);
fun pk(sk)[Av.:bitlength, algorithm] (get public key from secret key);
fun cert(pk,id,t,ca)[Av.:bitlength, algorithm](compute certificate);
fun nonce() [Availability:bitlength, algorithm] (compute new nounce);
fun skey()[Availability:bitlength, algorithm] (compute symmetric key);
fun enc(data,key)[Availability:bitlength, algorithm, opt] (encrypt the data);
fun dec(data,key)[Availability:bitlength, algorithm, opt] (decrypt the data);
fun hmac(data)[Avail.: algorithm, block_size_in_MB] (hmac generation);
fun ver(X1,X2) (comparing X1 to X2);
fun com(data)[Avail.:data_type, blocksize_in_GB, algorithm](compression);
fun decom(data)[Av.:data_type, blocksize_in_GB, algorithm](decompression);
fun newstate(state) (state of the protocol);
fun st_active() (active state);
fun st_closed() (closed state);
```

Equations

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

```
eq dec(enc(data,pk(SKid)),SKid) = data (asymmetric enc/dec)
eq dec(enc(data,K),K) = data (symmetric encryption/decryption)
eq ver(hmac(data),data) = true (verification of hmac digests)
eq decom(com(data)) = data (data compression/decompression)
```

Channels

In the presented example we define five synchronous channels.

```
channel ch1,ch2,ch3,ch4,ch5(100)[10 mbits];
```

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 TLS protocol. In the Listing 1 the TLS client is modelled and in the Listing 2 the TLS server is modelled.

Listing 1: The client of TLS protocol modelled in the QoP-ML

```
host Client (rr)(*)
{
 #D1 = data();

 process C(ch1,ch2,ch3,ch4,ch5)
 {
  ID1 = id();
  V1 = Vlist();
  C1 = Clist();
  Com1 = Comlist();
  N1 = nonce()[256,Linux PRNG];
  M1 = (ID1, V1, C1, Com1, N1);
  out(ch1:M1);

  in(ch2:Y);
  PKScert=Y[4];
  K1=skey()[256,Linux PRNG];
  K1E=enc(K1,PKScert)[2048,RSA,pk];
  ReadyEC=info(ReadyEncClient());
  FinC=info(FinClient());
  M3=(K1E, ReadyEC,FinC);
  out(ch3:M3);

  in(ch4:Q);
  Status=newstate(st_active());

  subprocess Cv1(*)
  {
   D1Com=com(D1)[bin,1.14,Deflate];
   D1ComE=enc(D1Com,K1)[256,AES,CBC];
   D1all=hmac(D1ComE)[SHA1,1];
   M5=(D1ComE,D1all);
  }

  subprocess Cv2(*)
  {
   D1E=enc(D1,K1)[256,AES,CBC];
   D1all=hmac(D1E)[SHA1,1];
   M5=(D1E,D1all);
  }

  out(ch5:M5);
  Status=newstate(st_closed());
 }
}
```

Listing 2: The server of TLS protocol modelled in the QoP-ML

```
host Server (rr)(*)
{
 # S = id();
 # CA = id();
 # SKS=sk(S)[2048,RSA];
 # PKS=pk(SKS)[2048,RSA];
 # T1=date();
 # PKScert=cert(PKS,S,T1,CA)[2048,RSA];

 process S(ch1,ch2,ch3,ch4,ch5)
 {
   in(ch1:X);
   V1ok=set(X[1]);
   C1ok=set(X[2]);
   Com1ok=set(X[3]);
   N2=nonce()[256,Linux PRNG];
   DoneS=info(Done());
   M2=(V1ok,C1ok,Com1ok,N2,PKScert,DoneS);
   out(ch2:M2);

   in(ch3:Y);
   ReadyES=info(ReadyEncServer());
   FinS=info(FinServer());
   M4=(ReadyES, FinS);
   out(ch4:M4);

   Status=newstate(st_active());

   in(ch5:Z);

   subprocess Sv1(*)
   {
    K1E=Y[0];
    D1ComE=Z[0];
    D1all=Z[1];
    K1=dec(K1E,SKS)[2048,RSA,sk];
    D1ComEbis=hmac(D1ComE)[SHA1,1];
    Vres=ver(D1all,D1ComEbis);
    D1Com=dec(D1ComE,K1)[256,AES,CBC];
    D1=decom(D1Com)[bin,1.14,Deflate];
   }

   subprocess Sv2(*)
   {
    K1E=Y[0];
    D1E=Z[0];
    D1all=Z[1];
    K1=dec(K1E,SKS)[2048,RSA,sk];
    D1Ebis=hmac(D1E)[SHA1,1];
    Vres=ver(D1all,D1Ebis);
    D1=dec(D1E,K1)[256,AES,CBC];
   }

   Status=newstate(st_closed());
 }
}
```

To analyse them, one does not have to design these two versions separately, these two versions can be abstracted in one protocol flow. During defining the protocol instantiation, one can specify the parameters characteristic of specific versions of TLS Handshake protocol.

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 configurations of TLS Handshake protocol. Hence, we need metrics for all functions that may affect the availability. We have checked the execution times of operations used in the TLS protocol that may be configured (ie. compression, encryption).

Many of security metrics may be obtained from the benchmarks present in both official hardware specifications and literature [14]. However, some metrics may depend on the hardware on which protocol is executed [1]. Therefore, designers should be able to compute those metrics on hosts on which the protocol will be executed. In our case study we have applied commonly used software to compute metrics, so that everyone can compute them on their host.

For encrypting, decrypting (both symmetric and asymmetric) we have used the openssl program with speed library [11]. It executes the checked operation over and over for a period of time (ie. 10s) and returns the results which were converted to ms per byte.

In the case of compression and decompression we used *gzip/gunzip* to compute metrics. It contains the *zlib* library that has implementation of compression algorithm based on *deflate*. It is called the *standard reference implementation used in a huge amount of software*.

For the functions which generate the nonce and asymmetric keys we prepared the software which architecture is described in the article [4].

In order to compare the results from the QoP estimation with the real implementation, we have performed a test in which we copied an Linux MEPHIS iso file (1.14GB) using the scp program (secured copy). We configured the ssh connection (used by scp program) to use the same algorithms as we modelled in the presented case study. The ssh uses the TLS Handshake protocol for data transition.

In the QoP-ML the security metrics are defined by the operator `metrics` and the body of the metrics is closed in the curly brackets. Details about other operators used for defining security metrics can be found in [4]. The metrics for analysed versions of TLS protocol are presented on Listing 3.

Listing 3: The metrics for TLS protocol

```
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[function][bitlength][algorithm][opt][Av:time(ms)];
 primitive[enc][2048][RSA][pk][0.049];
 primitive[dec][2048][RSA][sk][1.611];
 #
 primhead[function][bitlength][algorithm][opt][Av:time(mspB)];
 primitive[enc][256][AES][CBC][0.0000000049];
 primitive[dec][256][AES][CBC][0.0000000049];
 #
 primhead[function][algorithm][block_size_in_MB][Av:time(ms)];
 primitive[hmac][SHA1][1][2.475];
 #
 primhead[function][output_size:exact(B)];
 primitive[id][8];
 primitive[data][1224065679];
}

data+(host1.1)
{
 primhead[function][bitlength][algorithm][Av:time(ms)]\\
 [output_size:exact(B)];
 primitive[nonce][256][Linux PRNG][0.0025][8];
 primitive[skey][256][Linux PRNG][0.0025][8];
 #
 primhead[function][data_type][blocksize_in_GB][algorithm]\\
 [Av:time(ms)][output_size:ratio];
 primitive[com][bin][1.14][Deflate][31150][1:0.1];
 primitive[decom][bin][1.14][Deflate][6506][1:10];
}

set host Client(host1.1);
set host Server(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 TLS protocol (Listing 3), the first version with data compression and the second one without compression. In these versions two high hierarchy processes are executed: `host Client` and `host Server`.

In `version 1` inside the process `host Client`, the process `C` is executed (function - `run`) with the subprocess `Cv1`. Inside the process `host Server` the process `S` is executed with the subprocess `Sv1`. The TLS protocol versions can be modelled by defining the subprocess which will be executed in the specific protocol instantiation.

Listing 3: The metrics for TLS protocol

```
version 1
{
 run host Client(*)
 {
  run C(Cv1)
 }
 run host Server(*)
 {
  run S(Sv1)
 }
}
```

```
version 2
{
 run host Client(*)
 {
  run C(Cv2)
 }
 run host Server(*)
 {
  run S(Sv2)
 }
}
```

The second version of the TLS protocol is similar to the first one with one exception, the data is not compressed. The data processing without compression is modelled as the subprocesses `Cv2` and `Sv2` so the process `C` will execute the subprocess `Cv2` and the process `S` the subprocess `Sv2`.

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 the system efficiency. The total execution time (T_{Total}) of the two analysed versions of the TLS protocol is calculated. For the first version of the protocol the $T_{Total} = 42.86\ s$. In the second, without data compression, the $T_{Total} = 5.79\ s$. The execution time for the second version of the protocol is 86.49% shorter than in the first version.

During the analysis one can notice that the first version of the TLS protocol is very inefficient in the case of transmitting the big binary file. The compression ratio for the binary file is only 10% and the time of compression has the largest contribution to the total execution time. The reason for using compression is to reduce the size of data thanks to which the execution time for the creating message authentication code and the transmission will decrease. Reducing the data is justified when the transmitted data is a text, then the compression ratio for the algorithm *Deflate* is about 70%.

3 The runtime of TLS protocol implementation

For validation of the presented case study we compare the results of the protocol runtime estimated in the QoP-ML to runtime of an actual TLS protocol implementation. During the analysis we omit the computational overhead for packet transmission time, bit string comparison time, any hard drive operation time. The test was performed by means of the *scp* program which transmits the data using the TLS protocol. During the test we transmit the MEPHIS Linux iso file analysed in QoP-ML. The first and the second versions of the TLS protocol were executed 50 times and the average value was calculated with the standard deviation. In Tab. 1 we present the test results and the runtime estimated in QoP-ML. Comparing the results of the protocol runtime estimated in the QoP-ML to the runtime of the actual TLS protocol implementation, one can conclude that the protocol runtime estimated in the QoP-ML is in the range specified by the standard deviation. These results confirm the correctness of the efficiency modelling based on the QoP-ML approach.

Table 1. The TLS protocol runtime

	T_{Total} [s] QoP-ML estimation	T_{Total} [s] scp tests	standard deviation
Version 1	42.86	41.44	2.22
Version 2	5.79	6.47	0.80

4 Conclusions

The aim of this study was to present a new method and construction in new language QoP-ML [4] and show how to perform an efficiency 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 performed study includes two selected versions of the TLS protocol and only cryptographic algorithms were taken into account. The QoP 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 the article [4] we calculate the total protocol runtime for two versions of the TLS protocol. The protocol runtime estimated in the QoP-ML was validated by the real usage of the TLS protocol and confirms the correctness of modelling based on the QoP-ML approach.

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 many factors which influence the overall system security and efficiency.

Acknowledgements

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

References

1. Jaquith A. Security Metrics: Replacing Fear, Uncertainty, and Doubt. Addison-Wesley; 2007.
2. Jürjens J. Secure System Development with UML. Springer 2007.
3. Jürjens J. Tools for Secure Systems Development with UML. International Journal on Software Tools for Technology Transfer 2007; 9:527-544.
4. Ksiezopolski B, QoP-ML: Quality of Protection modelling language for cryptographic protocols . Computers & Security 2012; 31(4):569-596.
5. Ksiezopolski B, Kotulski Z. Adaptable security mechanism for the dynamic environments. Computers & Security 2007; 26:246-255.
6. Ksiezopolski B, Kotulski Z, Szalachowski P. Adaptive approach to network security. Communications in Computer and Information Science 2009; 158:233-241.
7. 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.
8. LeMay E, Unkenholz W, Parks D. Adversary-Driven State-Based System Security Evaluation. In Workshop on Security Metrics - MetriSec 2010;
9. 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.
10. Luo A, Lin Ch, Wang K, Lei L, Liu Ch. Quality of protection analysis and performance modeling in IP multimedia subsystem. Computers Communications 2009; 32:1336-1345.
11. Openssl Project: http://www.openssl.org/.
12. 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, Buenos Aires, Argentina, ACM.
13. Sun Y, Kumar A. Quality od Protection(QoP): A quantitative methodology to grade security services. In 28th confrence on Distributed Computing Systems Workshop 2008, p.394-399.
14. 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 2010; 110:247-151.
15. Theoharidou M, Kotzanikolaou P, Gritzalis S. A multi-layer Criticality Assessment methodology based on interdependencies. Computers & Security 2010; 29:643-658.
16. RFC 5246: The Transport Layer Security (TLS) Protocol v.1.2, 2008.