Optimization methods for system track data securing using digital signatures

G. Icriverzi
Data Processing Systems Department, ROMATSA, Romania

Abstract

Information security is critical in Air Traffic Management systems. Those systems use, among other sources, information received from the Radar Heads Processors. This information is used in the processed form of ASTERIX System Tracks. The accuracy and protection of this data is essential for the Air Traffic Management process. System Track radar data securing by adding a digital signature to each system track. A record requires high processing resources, especially for 2048 bits RSA keys. This approach does not modify the ASTERIX code structure thus keeping the compatibility with other communication partners. However, the use of this method in a sensitive environment like Air Traffic Management may obstruct real time radar data processing. The aim of this paper is to indicate new securing methods that will overcome this problem. Two alternatives are proposed and analyzed that can be combined in order to secure the data in real time. One alternative is using more powerful processing units and the other is applying an optimized method for adding a digital signature to system track data. Unlike the initial approach, where every System Track Record is added a digital signature, the new method relies on adding a digital signature to an entire block of records. The initial method and the optimized method were tested using several processing units. The results showed that the new method adds an insignificant overhead to the existing processing resources allowing the planning for real time system track data securing.
Keywords: ASTERIX, authentication, digital signature, radar data, system track.

1 Introduction

Air traffic administration services use, among other data sources and equipment, radar information from the radar heads (RHP) (Kelly and McIntyre [1]).

WIT Transactions on The Built Environment, Vol 134, © 2013 WIT Press
www.witpress.com, ISSN 1743-3509 (on-line)
doi:10.2495/SAFE130431

System track radar data (ASTERIX cat.062) authentication is a very important security measure in the air traffic management process. Any successful attempt to modify, eliminate, or add part or all this data will result in erroneous information displayed on the air traffic controller's screen. This can lead to aircraft collisions in air. Authentication can be achieved by the use of digital signature. Generating the digital signature has the disadvantage of high processing resources consumption.

The method described by Icriverzi and Cristea [2] uses the digital signature to authenticate in the code (application and presentation levels) every Record from a System Track Data Block, without modifying the code structure, thus allowing to keep the compatibility with data exchange partners. While the new generated data volume overhead is negligible, the accomplished tests showed very long digital signatures generation times for 2048 bits RSA keys. The air traffic domain deals with highly sensitive data that need to be transmitted in real time. In order to process authenticated real time radar data it is necessary to reduce the digital signature generation time.

To solve these problem two approaches were considered: the elaboration of an optimized authentication method using digital signatures and the alternative use of a faster processor to generate the signatures. An application was created to test the two methods and the results were compared.

The paper structure is as follows: in Section 2 the actual system track data securing method is presented, Section 3 contains the solutions to the first method problems, in Section 4 the experimental part and the results are described and analysed and Section 5 presents the conclusions.

2 Related work

The first approach of the system track code data authentication used the ASTERIX standard coding rules for adding the authentication information to a certain field of every Record (Icriverzi and Cristea [2]). The solution used to authenticate the cat.062 ASTERIX code was the use of the Record's digital signature added in the Record's SPF (Special Purpose Field). ASTERIX code standard is expandable with a number of different categories, each treating a specific type of radar information (Doukas *et al.* [3]). The data category this paper tries to secure is cat.062 (Doukas *et al.* [4]) and contains System Track data that is directly used in the graphical representation of the air traffic situation on the radar console (Still and Bunjevac [5]).

The generated digital signature was applied over the first part of the Record (all the Record data except SPF) and added in the SPF, this being the last field in the cat.062 Record's specification. SPF contains a *length octet* (that gives the total SPF length, including this octet) and the useful information, the digital signature.

The digital signature was generated inside a PKI framework that contains: CA (Certification Authority), RA (Registration Authority), KGS (Key Generation System), CSP (Crypto Service Provider) and KC (Key Ceremony) (Europepki project [6]).

By adding the digital signature to a Record its structure looks like this:

FSPEC (SPF bit = 1)	Field 1	Field 2	...	Field SPF - contains the digital signature

Figure 1: Data record with the digital signature in SPF.

The addition of the digital signatures to a generated sample of radar data was accomplished by a Java application using RSA keys of 512, 1024 and 2048 bits and two digital signature generation algorithms : *SHA256withRSA* and *SHA512withRSA*. The test system was an Intel P4/2.4GHz with 1.5GB RAM and the signature generation routines were executed inside a Java Virtual Machine JVM 1.5. The radar data sample was generated for 15 minutes of continuous air traffic using a custom UAP that allowed the adding of 2048 bits signatures in one field. The data used was custom generated because the access to real was not possible at that moment. The times achieved were as follows:

Table 1: The necessary time (in seconds) to generate the digital signature.

Provider/algorithm-key	SHA256 with RSA/512BITS	SHA512 with RSA/1024BITS	SHA512 with RSA/2048BITS
SunRsaSign version 1.5	60.980	351.282	2666.110
BouncyCastle version 1.46	79.485	502.500	2662.125

The maximum determined time for *verifying* the digital signature was around 84 seconds..

While the digital signature verifying times were acceptable (less than 15 minutes), digital signature generating times were acceptable only for 512 bits and 1024 bits keys. For 2048 bits keys the necessary time is far over 900 seconds (15 minutes).

The aim of this paper is to indicate new authentication methods that can sign the system track data in real time. The first approach of the problem was applied on customized data with a customized UAP (field sequence). As we will see further adding a 2048 bits signature to a Record leads to an incompatible ASTERIX standard encoding, so the new method presented here will address this problem too.

The securing power of this method is given by the RSA private key length and the used hash algorithm (Schneier [7]). Present estimations recommend that the minimum private key length must be 1024 bits. For medium time RSA protection however it is recommended the use of minimum 2048 bits length private keys (Kleinjung *et al.* [8]).

3 The optimization of the securing method

As we saw earlier generating digital signatures using 2048 keys is a very costly operation concerning the processing power.

It is compulsory that in a sensitive system like ATM the data processing to be made in real time. The use of this security method must not affect the resources used for current data processing. The above model consumes important processing resources generating a large number of long keys signatures.

In order to diminish the load on the processing unit it was elaborated an optimized system track data authentication model by the use of a single digital signature for an entire Data Block. In this case the signature is applied to the bit array formed by all the concatenated block Records and is added to the SPF of the first Record from the Block. Thus the processing unit overhead is significantly reduced by generating a smaller number of signatures. The Field Specification (FSPEC) bit array of the implicated Records will have the SPF bit set to 1 when the signature is applied. This convention will serve as a decoding rule to avoid confusions when applying or verifying the signature. This rule was used in the experimental part.

Figure 2: A Data Block's structure after adding the signature using the optimized method.

The ASTERIX specification for the SPF field states that the field's first octet indicates the SPF length in octets including the length field itself. Therefore the maximum bit length of SPF is 256*8 = 2048 bits. If we count 8 bits for the length field it remains maximum 2040 bits for the digital signature. This means that applying the ASTERIX coding rules a RSA 2048 bits signature cannot be added to a single Record. The solution proposed in this case is to split the signature in half and to add it to the SPFs of the first two Records of a Block. Both Records will have SPF bit set to 1 before applying the signature.

Figure 3: A Data Block's structure after adding a long (> 2040 bit) signature using the optimized method.

A special type of Data Blocks is when the block has only one Record. The 2048 bits signature cannot be added to only one Record. In order to overcome this a new Record that contains only the SPF is added. Thus the resulting block will have two Records with only the first one containing the useful information. The SPF bit rule was used here too.

Block length	Record 1	FSPEC (SPF=1)	...	digital signature part1/1	Record 2 dummy	FSPEC (SPF=1)	digital signature part2/2

Figure 4: A Data Block's with only one Record after adding a long (> 2040 bit) signature using the optimized method.

The new method's validation was made in a few steps. This time real data from an operational system was captured in a file and then used. The file was parsed in order to add the digital signature information to each data bock and the resulting data was stored in a new file.

In creating this application Java SDK 1.5 was used and Eclipse 3.2 as the developing environment. For generating the digital signature the default security provider was used (SunRsaSign version 1.5) as well as the BouncyCastle 1.46 distribution to compare. The used algorithms were *SHA256withRSA* and *SHA512withRSA* and the RSA private key lengths were 512, 1024 and 2048. The application was launched on several combination of signature/key. The pair of keys used were previously generated and stored on the disk. 512, 1025 and 2048 bits length keys were generated.

The program execution was carried using the JUnit test framework (Beck and Gamma [9]), and for the performance analysis by simulating several consecutive executions the JUnitPerf framework (Clark [10]).

The old authentication process was applied over a generated ASTERIX data, with a custom UAP, for a period of 15 minutes of traffic, with the traffic values taken from FAA estimations [11]. In this paper the experiment is carried for real traffic data, captured from an operational ATM system for 41 minutes and 24 seconds. To compare the two authentication methods this data is added the authentication information using both the old method from the first paper (Icriverzi and Cristea [2]) and the optimized method presented here. Being captured from a real system the used cat.062 UAP was not customized this time but it was exactly the one stated in the ASTERIX standard (Doukas *et al.* [4]). The experiment showed a more realistic result, given that it relies on real environment data. The experimental part compares the two methods applied over the new captured data sample on the original system and then on faster processors to highlight the time differences.

The chosen recorded traffic period was in the day time when the traffic values are higher than during the night.

4 Experimental part

4.1 The new method

Generating 2048 bits RSA keys signatures using the first method took 0.13 seconds for one signature on the presented system (Icriverzi and Cristea [2]). For a total of 2296 blocks, reducing the signatures number to one per Block, as in the new optimized method, we would have in theory 299.53 seconds as the total time

for generating the signed code, which is an acceptable time compared to the analyzed traffic interval of 900 seconds (15 minutes).

For the new method applied on the new data sample the recorded traffic contained 12432 data blocks and 78612 data records. The file's size was approximately 11 MB for the analyzed period (2484 seconds).

The following values were obtained for the generation and verification of the digital signatures using the new authentication method on the first system (Intel P4/2.8Ghz, 1.5GB RAM).

Table 2: The time in seconds needed for generating the digital signature using the optimized method.

Provider/algorithm-key	SHA256 with RSA/512biţi	SHA512 with RSA/1024biţi	SHA512 with RSA/2048biţi
SunRsaSign version 1.5	63.609	255.219	1472.891
BouncyCastle version 1.46	66.391	260.437	1551.500

Table 3: The time in seconds needed to verify the digital signature using the optimized method.

Provider/algorithm-key	SHA256 with RSA/512biţi	SHA512 with RSA/1024biţi	SHA512 with RSA/2048biţi
SunRsaSign version 1.5	30.610	39.672	71.641
BouncyCastle version 1.46	27.578	37.000	70.235

The obtained values demonstrate that even using this ancient system the optimized securing method can perform the digital signature operations in real time because the values are less than the analysed period – 2484 seconds.

4.2 Performance analysis

Besides running the tests using the JUnit framework for the performance analysis JUnitPerf was used. The applications were executed 10 times in a row for every type of test and a comparative analysis was made.

The tests were conducted on several processors in order to highlight the big time variations especially for long keys. The tested processors were Intel P4/2.8GHz, Intel Core2DuoP8400/2.26GHz, Intel Core2DuoE8400/3.00GHz. The tested application was developed using JVM 1.5.0.

The average times obtained from executing in a row with JUnitPerf the signatures generation on the three processors using the first method and a comparative graphic with data from all the executions are presented next.

Following are the testing of the new data sample from the real environment using both the old and the new methods of data signing. First it was tested on the same system and then on faster systems.

To maintain the full compatibility with ASTERIX encoding, the first authentication method was tested for 512 and 1024 RSA keys only because a 2048 bit signature does not fit into a SPF field of a Record. Following are the

results for the first authentication method on the new data sample from a real environment.

Table 4: The average necessary times (in seconds) for generating the digital signatures using the first method on different processors.

Processor/ algorithm-key	SHA256 with RSA.512/SUN	SHA512 with RSA.1024/SUN	SHA256 with RSA.512/BC	SHA512 with RSA.1024/BC
Pentium 4	240.617	1350.4281	289.6829	1489.1935
Intel P8400	133.9337	656.8596	159.9701	726.8197
Intel E8400	100.2438	522.6687	120.4624	586.0437

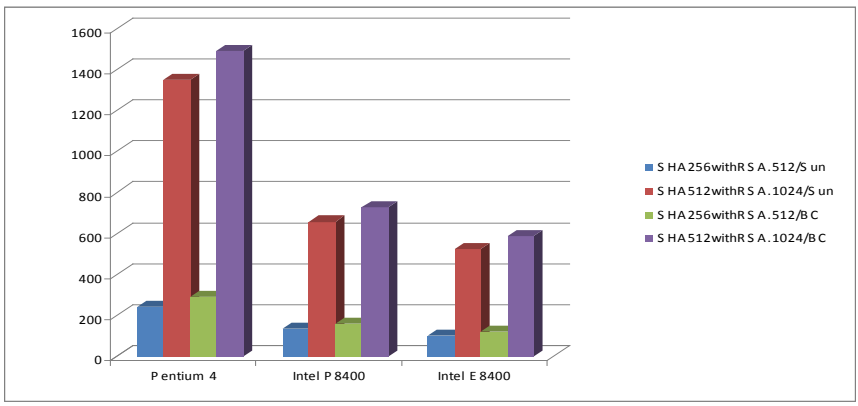

Figure 5: Number of seconds of all JUnitPerf executions using the first method.

It is clear that all the processors have no problem dealing with 512 and 1024 RSA keys (the processing times are smaller than the analysed period). Unfortunately this method cannot be applied for longer keys, otherwise would break the ASTERIX encoding rules. The resulted times show that the Java distribution's default security provider is faster than the BouncyCastle pack.

Next is presented the performance analysis for the optimized method for all the processors using JUnitPerf with ten consecutive executions.

Table 5: The average necessary times (in seconds) for generating the digital signatures using the optimized method on several processors.

Processor/ algorithm-key	SHA256 with RSA.512/ Sun	SHA512 with RSA.1024 /Sun	SHA512 with RSA.2048 /Sun	SHA256 with RSA.512/ BC	SHA512 with RSA.1024 /BC	SHA512 with RSA.2048 /BC
Pentium 4	60.3234	241.114	1473.508	66.636	260.564	1582.0501
Intel P8400	47.2726	128.6204	683.0727	51.3429	140.4515	725.8618
Intel E8400	27.2438	93.0546	542.5689	29.3873	102.2796	577.4153

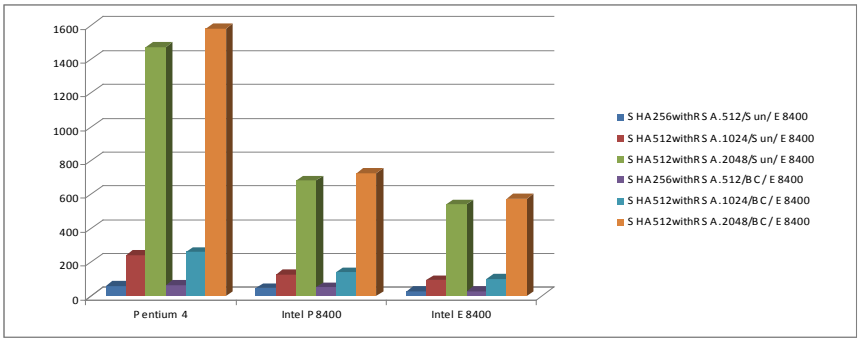

Figure 6: The necessary times (in seconds) for all JUnitPerf executions - signatures generation using the optimized method.

Using the optimized method gives real time performance on all the tested processors. In the same manner the times for verifying the digital signatures were analyzed.

The slowest system verifies one 2048 bit signature in 5.76 ms, compared to the fastest 2048 bit signature generation in 54.9 ms. This means that the processing resources needed for the signature checking are more than enough for signatures verifying even with the weakest system using any of the authentication methods

The total signature verification times vary between 30.498 seconds for P8400 and 512 bit keys using the optimized method and 112.313 seconds for P4 and 1024 bit keys using the first method. The results show that the digital signature verification is considerably faster than the generation, both processes being acceptable on any of the tested processors.

The obtained data imply that the first authentication method as well as the second one are practical but the first one can be used only with RSA keys smaller than 2048 bits.

The testing module is useful in determining if a certain hardware configuration is suitable to perform the digital signature operations of the system track data. For an input file with ASTERIX cat.062 data the necessary time values for generating the digital signatures are determined and then the tests values for the verifications.

4.3 The results analysis

Next we analyzed how practical the new securing method is by calculating the supported peak traffic value for the slowest processor (Pentium 4).

The number of active targets detected from the real traffic captured data is 323 and the number of data blocks(signatures) is 12432. Given the time needed for one signature we can compute the maximum number of aircrafts whose data can be handled by the given system in 2484 seconds.

Based on the previous experiments we have the following values for the signatures generation times according to the key lengths. As expected, the values

are similar with the ones from the first method, as they are measuring one signature.

Table 6: The necessary times (in ms) for generating one digital signature.

Provider/algorithm-key	SHA256 with RSA/512biţi	SHA512 with RSA/1024biţi	SHA512 with RSA/2048biţi
SunRsaSign version 1.5	5.11	20.52	118.47
BouncyCastle version 1.46	5.34	20.94	124.79

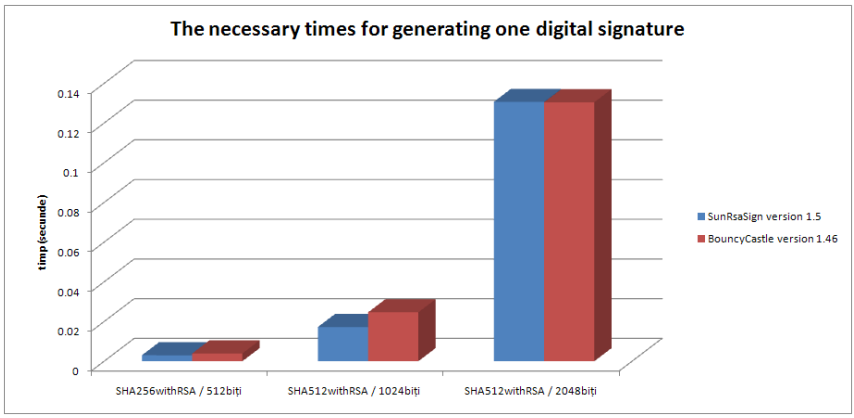

Figure 7: The necessary times for generating one digital signature on Pentium 4 system.

Using the values above the maximum number of flights for which the data authentication can be made in the analyzed period can be calculated.

For the first authentication method which generates one digital signature for each Record we get the following results for the peak traffic values (number of a/c) in the 2484 seconds interval. In order to keep the ASTERIX code compatibility only the 512 and 1024 bit signatures were tested.

Table 7: Peak traffic values using the first authentication method.

Provider/algorithm-key	SHA256 with RSA/512biţi	SHA512 with RSA/1024biţi
SunRsaSign version 1.5	1995	496
BouncyCastle version 1.46	1909	487

These values indicate that the slowest system can authenticate in real time more traffic data than the recorder sample (323 targets) using 512 and 1024 bit RSA keys.

Following are the peak traffic values using the optimized method.

Table 8: Peak traffic values using the optimized authentication method.

Provider/algorithm-key	SHA256 with RSA/512biţi	SHA512 with RSA/1024biţi	SHA512 with RSA/2048biţi
SunRsaSign version 1.5	12629	3145	544
BouncyCastle version 1.46	12085	3082	517

The results show that the optimized method allows for the authentication with 2048 bits keys of a greater number of a/c compared to the initial model with 1024 bit keys and the values are higher than the captured ones (+68%).

5 Conclusions

The authentication of the system track data in the code is a direct and efficient securing method of the radar data in aeronautical systems. Besides the original approach of this problem (Icriverzi and Cristea [2]) an optimized authentication method was added and both methods were testes on several processors.

The performed tests showed that the method presented in the article is a much faster alternative to the original approach. Authentication using 2048 bits RSA keys is possible for traffic values greater than the known peak values. The use of the first method is still an acceptable alternative for keys of maximum 2040 bits. A fast processing unit allows using this mechanism transparently for the ATC.

Using this model of direct in the code securing the other ASTERIX radar data categories besides cat.062 can be authenticated in the same way.

References

[1] Kelly, S. and McIntyre, J., *Radar Surveillance*, vol. 2 (chapter 3), Eurocontrol, 2003

[2] Icriverzi, G. and Cristea, V., *A Security Model For System Track Radar Data*, U.P.B. Scientific Bulletin., Series C, Vol. 74, Iss. 3, 2012

[3] Doukas, D., Berends, J., Rees, M. and Kerkhofs, G., *Surveillance Data Exchange*, Part 1, ASTERIX, Eurocontrol, 2007

[4] Doukas, D., Duflot, J.M., and Redeborn, B., *SDPS Track Messages*, Surveillance Data Exchange, Part 9 : Category 062, ASTERIX, Eurocontrol, 2007

[5] Still, J. and Bunjevac, S., *Data Processing Chain,* course, Luxembourg, 2005

[6] www.europepki.org, *European Libre Software Public Key Infrastructure*, 2003

[7] Schneier, B., *Cryptanalysis of SHA-1*, http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html, February 18 2005

[8] Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., Riele, H., Timofeev, A., Zimmermann, P., *Factorization of a 768-bit RSA modulus*, CRYPTO'10 Proceedings of the 30th annual conference on Advances in cryptology Pages 333-350, Springer-Verlag Berlin, Heidelberg 2010

[9] Beck, K., Gamma, E., *JUnit Cookbook*, http://junit.sourceforge.net/doc/cookbook/cookbook.htm, retrieved Oct. 2011
[10] Clark, M, *Continuous Performance Testing With JUnitPerf*, JavaProNews, 2003
[11] Miami Air Route Air Traffic Control Center, http://aspm.faa.gov/opsnet/sys/Tracon.asp, 2011