Checking of fault susceptibility of cryptographic algorithms

M.Sc. Piotr NAZIMEK

Piotr Nazimek received the M.Sc. and engineering degrees from the Faculty of Electronics and Information Technology at Warsaw University of Technology in 2005. He is currently a Ph.D. student at the same place. His research interests are in reliability of computer systems and networks, software engineering and cryptography.

e-mail: P.Nazimek@ii.pw.edu.pl

Ph.D. Piotr GAWKOWSKI

Piotr Gawkowski received the Ph.D. in 2005 from the Institute of Computer Science, Warsaw University of Technology, where he is currently an assistant professor. He is the author and coauthor of more than 50 scientific publications. His research interests are in dependability of digital systems, fault tolerance, and experimental evaluation of dependability. In particular, he develops software fault injection methodology and techniques, and the software based fault detection/tolerance approaches.

e-mail: P.Gawkowski@ii.pw.edu.pl

1. Introduction

Cryptography is widely used in various information systems. An important issue is to assure its robustness, which results in two requirements: its availability and security. Depending upon the used cryptographic algorithms we can provide appropriate theoretical level of security. This is easily assured for systems operating in some isolation (with no possibilities to be overtaken or disturbed by external players). In the literature there are some reports on possible attacks with side band channels [1-5]. They simplify deciphering critical keys using diagnostic channels (e.g. JTAG [2, 6, 7]), monitoring supply current or disturbing the system operation with faults [8]. In this paper we concentrate on the last issue. Moreover, we are interested in software implementation of cryptographic algorithms. Disturbing cryptographic programs is easier than cryptographic chips. Hence, the fault injection vulnerability of the first approach poses the question of software implementation susceptibility to faults, which is neglected in the literature. In our Institute we have developed quite powerful fault injection systems targeted at software applications of different types. Such universal systems created the possibility of analysing cryptographic algorithms. We have designed appropriate test scenarios and performed various experiments, which revealed fault susceptibility and showed some possibilities of improvements.

Section 2 describes the main features of two popular cryptographic algorithms DES and RSA. Section 3 presents the fault injection testbed. Experimental results are presented and discussed in section 4. They are concluded in section 5.

2. Cryptographic algorithms

In practice, various cryptographic algorithms are used and implemented either in hardware or software, they are described in the literature e.g. [9, 10]. In our research we concentrate on two representative and popular algorithms DES and RSA. We take into account various software implementations of these algorithms.

DES (Data Encryption Standard) is a symmetric-key algorithm for encrypting information [11]. It ciphers 8 bytes data block with 64 bits secret key. Plain text is processed by initial permutation, divided into two blocks, processed by 16 identical rounds and finally permuted. In each round half of a text block is scrambled with round key using so called F-function which involves four stages: expansion permutation of a text block, mixing with round key, substitution that uses constant S-boxes and permutation. DES key is processed initially by permuted choice permutation which selects 56 bits from original key. Then, for each round, with permuted choice permutation subkey is generated. General structure of decryption algorithm is identical, but subkey for each round is created in reverse order way. DES algorithm is based on simple binary operations like logical and, or, xor and shifts. So, it can be easily implemented in hardware. DES key is too short for many critical applications. It is recommended to use 3DES.
variant, which combines original DES three times with different keys. In experiments we used DES algorithm implementation derived directly from its specification [11] (further denoted as DES) and its optimised (in speed and size) version (DESOPT). The static/dynamic code sizes of these versions for the considered data block encryption (8 bytes long) were: 781/1097586 and 641/11337 in machine instructions, respectively. Moreover, we have taken into account other versions available in program libraries: OpenSSL library (DESOSSSL, 1596/2764), PKCS#11 Firefox 2.x library (DESNSS2, 3185/17666), PKCS#11 Firefox 3.x library (DESNSS3, 3933/22071), and WinCrypt Windows XP subsystem (DESWINC, 3214/8559). In the brackets we specify acronyms for the used versions and their static/dynamic code sizes.

RSA (acronym is created from the initials of authors surnames: Rivest, Shamir, Adleman) is a public key algorithm designed for encryption and signing [12]. This algorithm consists of three steps: key generation, encryption and decryption. In the first step the public key and private key are generated in the following way: choose two big and distinct prime numbers \( p, q \) and find an integer \( e \), which is coprime with \( (p-1)(q-1) \). Then compute an integer \( d \) from \( de \mod (p-1)(q-1) = 1 \) and \( n=pq \). Public key is the pair of numbers \( (e, n) \). It is known for everyone and could be used for encrypting messages which can be decrypted only with private key. Private key involves the pair of numbers \( (d, n) \). Ciphertext \( c \) for message \( m \) is computed as \( c^{e} \mod n \). Plain text \( m \) could be recovered from ciphertext \( c \) corresponding to \( m^{d} \mod n \). RSA is used currently in many applications. It is secure if the private key is secret and sufficiently long. RSA algorithm is based on big integers operations. Unlike DES it is not so easy to implement using low level language and it is significantly slower. In the experiments we use RSA software implementation for five different lengths of the key: RSA256 (8523/879042), RSA384 (7836/1759077), RSA512 (8720/2401968), RSA768 (8887/6252900) and RSA1024 (8746/10504053). The acronyms comprise the number of bits in the key, in the brackets we give also the static/dynamic code sizes of the analysed versions for given input message to cipher.

The prepared versions of cryptographic algorithms have been used to check their susceptibility to transient faults (which dominate in contemporary technologies [13, 14]) using special fault simulator developed in our Institute. The main features of this simulator are presented in the next section.

3. Fault injection testbed

In the experiments we use software implemented fault injection system FITS developed in our Institute [15]. Its concept is based on the Debugging API of the Win32 operating systems family. FITS is a sophisticated tool which uses capabilities typical to debuggers to control the execution of the application under test (AUT). Configuring experiments in FITS we define the scope of the simulated faults (disturbances) to the selected areas of the AUT (called testing areas).

The experiment must be preceded by the golden run (undisturbed) execution of the AUT, to get its profile (e.g. a set of executed machine instructions, code coverage, resource usage profile, statistical information upon the AUT) and results. They are used as a reference for the analysis of AUT behavior during tests with injected faults. Tests are performed by disturbing states of registers or memory cells during AUT execution. We can provide application specific oracle procedures to qualify test results. During tests the injected disturbance (which emulates effects of the preconfigured fault e.g. bit flip, stuck-at-x) is generated at the specific moment of the execution of the tested application (fault triggering moment) and in an appropriate fault location e.g. some bits in CPU registers (R), memory with data (D) and instruction code (C).

Various experiment scenarios can be configured [15, 16]. In this research we generated in an automatic way single-bit flip faults in different system locations (R, D, C) with pseudorandom distribution. Fault triggering moments can be distributed equally in time or space. In the first cases the fault injection moments are equally distributed over the execution time of the tested application. In such case, the most frequently executed instructions (e.g. in the loops) become most frequently fault triggers. Hence, some rarely executed code parts within the AUT can be skipped as fault triggers. So, it is reasonable sometimes to use equal distribution of the fault triggers over the code space (equal number of fault triggers for all the machine instructions) in order to identify the most fault-sensitive, robust parts of the code [15, 16].

The experiment is composed of a set of tests (usually many thousands or millions) – each means a single execution of the AUT with a fault disturbance injected at the given triggering moment. FITS uses pseudorandom generator to select fault triggering moments, bit locations for disturbance etc. After the fault injection, the AUT execution is monitored – FITS collects all possible exceptions and messages (AUT can inform FITS that the error was detected and given actions taken with user messages [15]). All those information are then logged to the result file and are helpful in further analysis of fault effects or error detection/tolerance effectiveness.

The injected fault can provoke an unhandled exception within the AUT – in such case the execution is terminated by the operating system and the test is considered as system exception. Typically exceptions relate to memory access violation, invalid or privilege opcode fetch, parity errors, arithmetic or stack overflow, etc. Every test has given predefined time limit – in case of exceeding this limit, the test is aborted by FITS and considered as timeout. Injected fault can also has no observable impact on the execution of AUT, so, the AUT can terminate by itself in normal fashion. In such case however there are two possibilities: the result produced is acceptable and considered as correct or is incorrect. FITS provides statistics of test results (percentage of correct, incorrect, system exception and timeout type results) and on the application profiles.

4. Experimental results

The performed experiments were targeted at checking the behaviour of cryptographic algorithms in the presence of transient faults. All the analysed algorithms have been implemented as programs (section 2) running under Windows in IBM PC environment. Faults were injected into CPU registers, data and code. Two kinds of experiments have been developed: checking the operation of ciphering algorithms and checking these algorithms secured by the deciphering process. In the second case the encrypted text is sent out after some verification (described in a sequel). For each test (an injected fault) we qualify the algorithm behaviour as correct, incorrect, system exception and timeout (section 2). In the case of incorrect qualification we check test leakage probability and distinguish several levels of text leakage. The i-th leakage level relates to the situation of sending out the text comprising i explicit (not encrypted) bytes on the same positions as in the primary text. In the case of 8 byte blocks the most critical are 8-th level text leakage (complete block is not encrypted). This analysis is valid for DES algorithm. In RSA algorithm, depending upon the size of the key, we use the same input text block (8 bytes long – same as in DES) with appended padding bytes (to fulfill the RSA requirements according to the given cipher key length). However, in the leakage analysis we consider only the basic block bytes.

Fault sensitivity of different versions of DES and RSA algorithms was verified in fault injection experiments. Here we use the same notation as in section 2. However, for each version two experiment schemes were used:

1) S scheme – for each test pseudorandom distribution of fault triggering moments and fault location within the specified area (e.g. R, D, C) has been applied, however, the input data block and the key were the same for all the tests,
2) R scheme – fault triggering and fault location distribution the same as in S scheme but the input data block and the key were generated in a pseudorandom way for each test.

Tab. 1. Fault sensitivity of primary DES algorithm (for single bit-flip faults injected into CPU registers/data/code memory area)

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Correct ciphertext</th>
<th>Incorrect ciphertext</th>
<th>System exception</th>
<th>Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESOPT S</td>
<td>64/9/39/74/49</td>
<td>5/58/12/50</td>
<td>27/71/72/91</td>
<td>2.00/51.0</td>
</tr>
<tr>
<td>DESOPT R</td>
<td>65/1/40/8/43/4</td>
<td>5/57/72/26</td>
<td>27/71/8/29/1</td>
<td>2.00/51.0</td>
</tr>
</tbody>
</table>

These two test scenarios are specified in the acronyms of cryptographic algorithms as the S or R postfixes. In the basic experiments we analysed the cryptographic algorithms by disturbing their execution. The input text block and the key initializations are not disturbed by faults. For DESOPT version the results are shown in tab. 1. Each entry in the table column comprises the percentage of corresponding test results (specified in the description of the column) for three fault locations: CPU registers/memory data area/memory code area. It is worth noting that this primary algorithm generates 40-65% of correct results despite faults. However, in 5-58% of cases the generated encrypted blocks are incorrect. Most of incorrect tests relate to faults injected into memory data area (however this area is relatively low as compared with dominating code area). In almost 30% of tests system exceptions were generated and 1% of timeouts observed (faults injected into code). The most critical are cases with incorrect output but without signalization of errors. We have performed a more detailed analysis of incorrect blocks; in particular, we were interested in text leakage effect. The 8th level text leakage was observed 4 times within 22548 incorrect cases. The highest percentage (3.78% - 852 test cases) was observed for the 1st level text leakage (only one byte revealed). Similar results have been obtained for other cryptographic versions.

To improve the dependability and security of the cryptographic algorithms in the presence of faults we have introduced the verification phase. The general scheme of the so-called secured algorithm is given in fig. 1. The main idea is to encrypt the block of the input message and then verify ciphertext by decryption and comparing the result with the original input message. In the case of positive comparison the encrypted text is sent out from the system. In the opposite case it is blocked and an error signal is issued. We have analysed fault susceptibility of this approach for all versions described in section 2. The results for DES and RSA algorithms are given in tab. 2 and 3, respectively. The entries in table cells denote the percentage of test results in relevance to fault locations (as in tab. 1). As compared with tab. 1 we can observe that incorrect text blocks are not sent from the system. So it seems, that the algorithm is much more dependable and secure. It is assured by detecting faulty behaviour either by system exception, timeouts or comparison detection.

![Fig. 1. Secured cryptographic algorithm](image)

Fig. 1. Secured cryptographic algorithm

Rys. 1. Schemat zabezpieczenia algorytmu kryptograficznego

Depend upon DES algorithm implementation the percentage of correctly encrypted text blocks sent out fluctuates in the ranges 44-73%, 39-78% and 26-57% for faults injected into CPU registers, data and code memory areas, respectively. The percentage of blocked transmissions and exceptions was complementary to the correct blocks.

Tab. 2. Fault sensitivity of secured DES algorithm

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Comparison OK (encrypted text is sent out)</th>
<th>Comparison FAULTY (incorrect text is blocked)</th>
<th>System exception</th>
<th>Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES S</td>
<td>72/6/78/26/26</td>
<td>0/3/72/20/5</td>
<td>27/1/5/2/51</td>
<td>0.00/2/1.7</td>
</tr>
<tr>
<td>DES R</td>
<td>72/6/78/3/26</td>
<td>0/4/5/6/20</td>
<td>27/0/5/9/51</td>
<td>0.00/2/1.8</td>
</tr>
<tr>
<td>DESOPT S</td>
<td>64/9/39/8/34/43/5</td>
<td>5/4/5/8/25/26</td>
<td>27/3/5/8/51</td>
<td>2.00/4/1.0</td>
</tr>
<tr>
<td>DESOPT R</td>
<td>64/8/39/4/34/4</td>
<td>5/5/8/7/2/6</td>
<td>27/1/7/3/61</td>
<td>2.00/4/0.9</td>
</tr>
<tr>
<td>DESSSS L</td>
<td>5/4/3/7/7/47</td>
<td>1/4/4/0/3/24/1</td>
<td>31/3/1/2/81</td>
<td>0.00/0/0.1</td>
</tr>
<tr>
<td>DESSSS S</td>
<td>5/4/7/7/47</td>
<td>1/4/3/0/24/1</td>
<td>31/3/1/2/81</td>
<td>0.00/0/0.1</td>
</tr>
<tr>
<td>DESSSN S</td>
<td>55/5/9/5/46</td>
<td>0/5/4/5/8/2</td>
<td>36/4/3/5/84</td>
<td>0.00/0/0.3</td>
</tr>
<tr>
<td>DESSSS N</td>
<td>55/5/6/8/51</td>
<td>0/5/2/5/9/8</td>
<td>39/3/5/2/34</td>
<td>0.10/5/0.2</td>
</tr>
<tr>
<td>DESSSS N</td>
<td>55/6/8/51</td>
<td>0/5/2/5/9/8</td>
<td>39/3/5/2/35</td>
<td>0.20/5/0.2</td>
</tr>
<tr>
<td>DESSSN C</td>
<td>44/0/5/1/48/5</td>
<td>6/4/6/4/8/5</td>
<td>9/8/2/5/3</td>
<td>0.00/0/0.1</td>
</tr>
<tr>
<td>DESSSS C</td>
<td>44/0/5/2/46/9</td>
<td>6/4/4/6/4/8</td>
<td>9/5/2/2/5</td>
<td>0.00/0/0.1</td>
</tr>
</tbody>
</table>

Test result fluctuations for RSA algorithm were lower (tab. 3). All these fluctuations result from different usage of related resources (registers, data and code memory) in the implemented versions of the analysed algorithms. In fact we have to perform a deeper analysis of the encrypted text accepted by the comparison taking into account the fact that faults can disturb the encryption, decryption as well as comparison part of the secured algorithm (see fig. 1). Practically, there were no leakage observed in experiments with secured implementations.

Tab. 3. Fault sensitivity of secured RSA algorithm

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Comparison OK (encrypted text is sent out)</th>
<th>Comparison FAULTY (incorrect text is blocked)</th>
<th>System exception</th>
<th>Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA256 S</td>
<td>53/6/54/6/48</td>
<td>6/3/2/6/0/10/4</td>
<td>39/8/2/3/4/4</td>
<td>0.30/9/0.4</td>
</tr>
<tr>
<td>RSA256 R</td>
<td>54/3/55/3/57</td>
<td>6/2/8/2/9/2</td>
<td>39/8/0/2/3/4</td>
<td>0.60/3/0.6</td>
</tr>
<tr>
<td>RSA384 S</td>
<td>5/4/6/1/6/1/0/6</td>
<td>6/7/13/5/6/7/3</td>
<td>38/7/2/4/3/17</td>
<td>0.60/3/0.6</td>
</tr>
<tr>
<td>RSA384 R</td>
<td>5/4/6/2/6/1/1/0/6</td>
<td>6/5/13/9/7/6/3</td>
<td>39/1/2/1/3/13</td>
<td>0.20/0/0.0</td>
</tr>
<tr>
<td>RSA512 S</td>
<td>5/6/6/6/5/7/7/7</td>
<td>6/4/5/1/1/0/6/1</td>
<td>36/7/1/2/3/19</td>
<td>0.30/0/0.3</td>
</tr>
<tr>
<td>RSA512 R</td>
<td>5/3/6/6/9/6/0/6</td>
<td>7/1/13/9/1/0/2/6/3</td>
<td>39/1/8/7/2/8/8</td>
<td>0.20/5/0.4</td>
</tr>
<tr>
<td>RSA768 S</td>
<td>57/9/4/0/5/6/0/6</td>
<td>6/1/12/8/1/0/1/6/3</td>
<td>35/9/1/2/3/3/5</td>
<td>0.10/0/0.0</td>
</tr>
<tr>
<td>RSA768 R</td>
<td>57/3/7/5/9/2/2</td>
<td>5/2/9/8/9/2/6/3</td>
<td>37/4/1/2/3/1/2</td>
<td>0.10/0/0.7</td>
</tr>
<tr>
<td>RSA1024 S</td>
<td>59/1/7/9/5/5/5/0/6</td>
<td>5/2/9/5/9/6/2/3</td>
<td>35/7/1/5/4/5</td>
<td>0.00/0/0.3</td>
</tr>
<tr>
<td>RSA1024 R</td>
<td>55/1/7/9/5/9/5/9/5/2</td>
<td>6/1/7/6/9/0/2/3</td>
<td>38/8/1/2/3/4/7</td>
<td>0.00/0/0.4</td>
</tr>
</tbody>
</table>

It is also worth noting, that both considered algorithms are not input data dependent (regardless to the input data for encrypting same fault sensitivity was observed). Significant differences are between the tested implementations (off-the-shelf ones) of the functionally the same algorithms (see fault sensitivity of R and S variants). Another sensitive parameter is the cipher key length in case of RSA. Longer key increases the percentage of the cases in which the validating comparison succeeds from (54% to 79%) in case of faults in data memory. That require further research.

5. Conclusion

The presented experimental results confirm fault sensitivity of cryptographic algorithms implemented in software. Practically up to 30% of faults are signalled by system exception mechanisms. The most critical are situations with not signalled faulty operation. Moreover, they may involve sending plain (not ciphered) text resulting in unsecured operation. This situation can be limited by
checking the correctness of ciphered text with deciphering. The simulation results confirmed the usefulness of this approach. It is worth also noting that available on chip error detectors (e.g. exception mechanisms) also block sending incorrect ciphered text. The most fault sensitive is the algorithm code. In the case of transient faults this vulnerability can be reduced by storing the code in a non-volatile memory e.g. flash.

The performed research is not only an original contribution in the area of cryptography but also it enhances the capabilities of fault injection techniques. These techniques have been widely used for checking fault sensitivity with calculation oriented applications (e.g. [17]). We have successfully extended this technique for real time systems [15, 16]. The experiments presented in the paper showed new aspects of detailed qualifying test results, in particular by distinguishing different levels of text leakage.

Further research is targeted at finding simple assertions and using performance monitoring, which can check on-line the operation of the algorithms. Moreover, the developed fault injection technique seems to be useful for analyzing side channel attacks [18].

This work was supported by Ministry of Science and Higher Education grant 4297/B/T02/2007/33.

6. References


Artikel recenzowany

INFORMACJE

Zapraszamy do prenumeraty czasopisma PAK w 2010 roku

Cena prenumeraty rocznej: 192,00 zł netto/1 egz.

Prenumeratę i kolportaż prowadzą:

WYDAWNICTWO POMIARY AUTOMATYKA KONTROLA
ul. Świętokrzyska 14A, pok. 530,
00-050 Warszawa,
tel./fax: 022 827 25 40

Redakcja czasopisma POMIARY AUTOMATYKA KONTROLA
ul. Akademicka 10, pok. 30b,
44-100 Gliwice,
tel./fax: 032 237 19 45,
e-mail: wydawnictwo@pak.info.pl,
www.pak.info.pl