Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Computer Science and Engineering

Ing. Tomáš Rosa

MODERN CRYPTOLOGY: STANDARDS ARE NOT ENOUGH

(Doctoral Thesis)

Doctoral study program: Electrical Engineering and Informatics
Study branch: Informatics and Computer Science (2612V025)

Supervisor: Doc. RNDr. Ing. Petr Zemánek, CSc.

Prague, July 2004

Dedicated to my family. Their continuous patience and support allowed me to fully concentrate on research in the fascinating area of modern cryptology.

Content

Modern Cryptology: Standards Are Not Enough

Tomáš Rosa

Department of Computer Science and Engineering
Faculty of Electrical Engineering, Czech Technical University, Karlovo náměstí 13, 121 35
Prague 2, Czech Republic, EU

Abstract. Development and implementation of various standards represent the mainstream of contemporary cryptography. Standards such as AES, SHA-1, DSA, ECDSA, RSA or standards such as PKCS, etc. are good examples of that. These standards are kept up-to-date and made public. Does this mean that anyone with a basic knowledge of computer architecture and discrete mathematics can simply build up a secure cryptographic module following these standards? Also, does this tell us that all cryptographic modules using the same cryptographic standard have the same level of security? Unfortunately, it does not. The main focus of the thesis is to draw an attention on several topics of the area of applied cryptography, which are very often neglected by many security architects. These topics will be demonstrated mainly on practically feasible attacks which were or would be possible because of architects of security modules or even standards did not pay appropriate attention to certain key aspects of applied cryptography. It turns out that, despite of surviving belief of various experts, following even highly trusted security standards is simply not enough to build up a really secure security module. These standards can be used as useful hints of what we shall (not) do, but the definite responsibility of checking potential vulnerabilities of a particular security module designed is still left on their architects.

1. Introduction – State of the Art

There are two basic questions which seem to be so important for identifying and resolving potential vulnerabilities that even a high-skilled security architect should not regret of paying an appropriate attention to them. The first question is:

What environment shall the designed module be used in?

The main aim of every security module is to defeat certain vulnerabilities of a target system (for example an online banking application) to lower risks coming from potential threats. For this purpose, the threat is defined as an event which could cause a certain loss of subjects incorporated in using the particular application (here, it could be a threat of stealing an access to somebody's banking account, etc.). The vulnerability is then defined as a set of conditions which allow the particular threat to harm the system (here, it could be a security hole in an authentication module, etc.). Since the set of concrete threats together with their characteristics is given mainly by a concrete environment in which the designed module will be used, it is absolutely

necessary to answer the first question mentioned above and to make up an accurate threat model. In such a model, we must then carefully examine as many properties of the module as we can to verify whether the module will really remove all those vulnerabilities or not. Moreover, we must also check if there are not some new vulnerabilities which would be introduced by applying this module. Otherwise, it may happen that the designed module will have such property that would turn out to be a serious vulnerability allowing disastrous threat to occur. Although it may seem as nothing more than just repeating basis of the best designing practice, the reality shows that most of devastating attacks are possible mainly because of the fact that this code of best practice is being constantly underestimated and overlooked. For instance, ignoring physical properties of cryptographic modules (i.e. the environment which surrounds every physical device) motivated the development of a brand new, rapidly developing area of cryptanalytical techniques called *side channel cryptanalysis*. Roughly speaking, introduction of this theory (around 1996) was the time when devastating attacks returned back to the papers presented at conferences on cryptology. We may really say that it was a revolution in contemporary cryptology which, hopefully, changed the way of viewing and modeling cryptographic modules. However, it will probably take some time until this theory becomes also practice. At the time of completing the thesis (spring-summer of 2004), side channel attacks are still very dangerous and very few modules can be regarded as reasonably protected against them. Therefore, most of the papers included in this thesis are focused on side channel attacks to deeply illustrate their nature and some techniques to defeat them.

The second key question is:

What is the easiest problem an attacker has to solve to break the module in some way?

As security architects, we should answer this question when we have an accurate threat model constructed in the previous step. It is important to note that, for example, identifying potential side channels would be of no benefit if we underestimate the way they would help an attacker to break into the system. The core is that traditional theoretical cryptanalysis tends to be focused on well-known, "well-hard" problems (such as factorization, discrete logarithm, etc. c.f. [9], [20]), while the particular problems an attacker has to solve in practice to be able to say that "she broke the system" are often essentially easier. Consequences of overlooking this aspect can be again easily seen from unusually good results obtained by side channel attacks. However, side channels are not the only one area where we can see that. As an example, we have also included in chapter G (see organization notes bellow) a new kind of attack on the well-known signature schemes DSA and ECDSA [4], [9], [20]. This is not a side channel attack, but it can also introduce serious weaknesses in certain systems based on a growing phenomenon of electronic signatures. Furthermore, we did not have to solve any from those "well-hard" problems (here namely the discrete logarithm problem) to do our attack. What we actually did is that we exploited such a property of these schemes which tends to be constantly overlooked by many researchers.

Certain evidences, that answering the above mentioned questions is of a crucial importance, can be seen if we look carefully at the attacks studied and presented at various conferences in the past and nowadays. The attacks discussed in the past were

almost solely focused on cryptanalysis of intercepted cryptograms, while the ones presented nowadays are somehow mentioning playing an interactive game between an attacker and her victim. This naturally reflects the way in which cryptosystems are implemented into practical applications. Being in the role of the attacker, we do not have to rely solely on randomly intercepted cryptograms any more. Playing the interactive game with our victim, we can "adjust" the conditions of our attack to finally get as easiest mathematical problem to solve as possible. Although it can be perhaps a bit "disgusting" for a beautiful mathematical mind, this subject must be studied and understood properly to tightly grasp what the contemporary cryptology is all about, which is then necessary to be able to fight with modern attackers as effectively as possible. Author's opinion here is that even in this area of so-called *theory of applied cryptography*, one can find very interesting problems for any taste of mathematical complexity and-or engineering practice. This is the main motto behind the papers written and completed in this thesis.

2. Goals of the Doctoral Thesis

The main goals of the dissertation are:

- To investigate several selected security standards which are widely used in contemporary security modules in order to see if they are designed properly according to particular key issues of modern cryptology (c.f. §1 above).
- To propose, elaborate, and describe possible practical attacks based on vulnerabilities found in these standards. The main focus is on the area of side channel cryptanalysis which is highly promising and rapidly growing part of contemporary cryptanalysis.
- To design and-or suggest effective countermeasures against discovered attacks.
- To contribute to a general theory of side channel cryptanalysis. Since this kind of cryptanalysis is the main tool used in the thesis, together with the fact that it is still rapidly growing, it would be desirable to try to independently generalize certain new ideas which were discovered for the purpose of the attacks presented here. We note that this goal is mainly achieved in the overviewing part of the thesis (c.f. organization of the thesis bellow) where a practical enhancement of classification methodology is proposed. Certain general results and observation are also pointed out in detailed descriptions of particular attacks.

3. Organization of the Thesis and Results Summary

The thesis consists of extended versions of papers which reflect author's results obtained during his PhD research. Each paper represents one chapter of the thesis

indexed as A, B, …, G. The relevant information on how and where particular papers were published, is included as footnotes at their relevant starting pages. Short abstracts of each chapter showing the main author's results obtained follow.

3.1 Chapter A. Side Channel Cryptanalysis – An Overview

Growing theory of the side channel cryptanalysis shows the necessity of building and using general models of cryptographic modules when their security has to be examined. Traditional approach, which was used before, was to examine these modules as abstract mathematical functions without their connection to the objective physical reality. It shows that particular physical properties can prominently spread the set of vulnerabilities and available cryptanalytic techniques. From here follows their impact on the security. The information available due to particular physical properties is referred to as side information. The means, which the side information is transmitted by, are then referred to as side channels. Practically, side channels are often represented as physical magnitudes, which are in some ways related to an activity of the cryptographic module being examined (the amount of time it takes to perform some operation, the power trace, the electromagnetic emanation, etc.). This overviewing chapter presents various general aspects of the theory of side channel cryptanalysis. It introduces particular types of side channels, which are known up to now, and it sketches, how these side channels can be used for cryptanalytic purposes. It also proposes a general classification methodology which allows practically useful distinguishing between various channels and their analyses. Furthermore, it separates the terms channel, signal, analysis, and information which should also be practically beneficial.

3.2 Chapter B. Attack on Private Signature Keys of the OpenPGP format, PGP™ programs and other applications compatible with OpenPGP

In this chapter, we describe an attack on the OpenPGP format [17], which leads to a disclosure of private signature keys of the DSA [4] and RSA [18] algorithms. The OpenPGP format is used in a number of applications including PGP, GNU Privacy Guard and other programs specified on the list of products compatible with OpenPGP, which is available at http://www.pgpi.org/products. Therefore all these applications shall undergo the same revision as the actual program PGP™. The success of the attack was practically verified and demonstrated on the PGP™(*) program version 7.0.3 with a combination of the AES [5] and DH/DSS algorithms [17]. As the private signature key is the basic information of the whole system which is kept secret, it is encrypted using the strong cipher. However, we show that this protection is weak, as the attacker has neither to attack this cipher nor user´s secret passphrase. A modification of the private key file in a certain manner and subsequent capturing of one signed message is sufficient for a successful attack. A vulnerability coming from an insufficient protection of the integrity of the public as well as private parts of

(*) PGP is registered trade mark of Network Associates, Inc. All other registered and not registered trade marks listed in this document are owned by their appropriate owners.

signature keys in the OpenPGP format is analyzed. On the basis of this, a procedure of attacks is shown on both DSA and RSA private signature keys. The attacks apply to all lengths of parameters (modules, keys) of RSA and DSA. The cryptographic countermeasures for correction of the OpenPGP format as well as the PGP™ format are proposed.

3.3 Chapter C. Further Results and Considerations on Side Channel Attacks on RSA

The research presented in this chapter contains three parts. In the first part, we present a new side channel attack on a plaintext encrypted by EME-OAEP PKCS#1 v.2.1 [11]. In contrast with recent well-known Manger's attack [7], we attack directly that part of the plaintext, which is shielded by the OAEP method. In the second part, we remind that Bleichenbacher's [2] and Manger's attack on the RSA encryption scheme PKCS#1 v.1.5 and EME-OAEP PKCS#1 v.2.1 can be converted to an attack on the RSA signature scheme with any message encoding (not only PKCS). In the third part, we deploy a general idea of fault-based attacks (we introduce a notion of confirmation oracle) on the RSA-KEM [19] scheme which was suggested as a possible solution to implementation attacks (e.g. side channel attacks) which seem to be constant problems of the schemes from [11]. We present two particular attacks as examples to show that this solution is clearly not a definite one. The result of these attacks is the private key instead of the plaintext as with attacks on PKCS#1 v.1.5 and v.2.1. These attacks should highlight the fact that the RSA-KEM scheme is not an entirely universal solution to problems of RSAES-OAEP implementation and that even here the manner of implementation is significant.

3.4 Chapter D. Strengthened encryption in the CBC mode

Vaudenay [21] has presented a side channel attack on the CBC mode of block ciphers ([10], [16]), which use padding according to the PKCS#5 standard [12]. One of the countermeasures, which he assumed, consisted of the encryption of the message $M' = M \,||\, padding \,||\, hash(M \,||\, padding)$ instead of the original M, where *hash* is an appropriate cryptographic hash function. This can increase the length of the message by several blocks compared with the present padding. Moreover, Wagner [21] showed a security weakness in this proposal. The next correction, which Vaudenay proposed ("A Fix Which May Work") has a general character and doesn't solve practical problems with the real cryptographic interfaces used in contemporary applications. In this article we propose three variants of the CBC mode. From an external point of view, they behave the same as the present CBC mode with the PKCS#5 padding, but they prevent Vaudenay's attack. In this chapter, we also make use of the notion of confirmation oracle which has been introduced in chapter C.

3.5 Chapter E. Side Channel Attacks on CBC Encrypted Messages in the PKCS#7 Format

As shown by Vaudenay in [21] and also discussed in chapter D in this thesis, a CBC encryption mode ([10], [16]) combined with the PKCS#5 padding [12] scheme allows an attacker to invert the underlying block cipher, provided she has an access to a valid-padding oracle which for each input ciphertext tells her whether the corresponding plaintext has a valid padding or not. Having in mind the countermeasures against this attack, different padding schemes have been studied in [1]. The best one is referred to as the ABYT-PAD. It is designed for byte-oriented messages. It removes the valid-padding oracle, thereby defeating Vaudenay's attack, since all deciphered plaintexts are valid in this padding scheme. In this chapter, we try to combine the well-known cryptographic message syntax standard PKCS#7 [13] with the use of ABYT-PAD instead of PKCS#5. We also make use of a generalized notion of the confirmation oracle introduced in chapter C. Let us assume that we have access to a $\text{PKCS\#7}_{\text{CONF}}$ confirmation oracle that tells us for a given ciphertext (encapsulated in the PKCS#7 structure) whether the deciphered plaintext is correct or not according to the PKCS#7 (v1.6) syntax [3]. This is probably a very natural assumption, because applications usually have to reflect this situation in their behavior. It could be a message for a user, an API error message, an entry in the log file, different timing behavior, etc. We show that an access to such an oracle again enables an attacker to invert the underlying block cipher. The attack requires single captured ciphertext and approximately 128 oracle calls per one ciphertext byte. It shows that we cannot hope to fully solve problems with side channel attacks on the CBC encryption mode by using a “magic” padding method or an obscure message-encoding format. Strong cryptographic integrity checks of ciphertexts should be incorporated instead.

3.6 Chapter F. Attacking RSA-based Sessions in SSL/TLS

In this chapter, we present a practically feasible attack on RSA-based sessions in SSL/TLS protocols [15], [14]. These protocols incorporate the PKCS#1 (v. 1.5) [11] encoding method for the RSA encryption of a *premaster-secret* value. The *premaster-secret* is the only secret value that is used for deriving all the particular session keys. Therefore, an attacker who can recover the *premaster-secret* can decrypt the whole captured SSL/TLS session. We show that incorporating a version number check over PKCS#1 plaintext used in the SSL/TLS creates a side channel that allows the attacker to invert the RSA encryption. The attacker can then either recover the *premaster-secret* or sign a message on behalf of the server. Practical tests showed that two thirds of randomly chosen Internet SSL/TLS servers were vulnerable. The attack is an extension of Bleichenbacher’s attack on PKCS#1 (v. 1.5) [2]. We introduce the concept of a *bad-version oracle* (BVO) that covers the side channel leakage, and present several methods that speed up the original algorithm. Our attack was successfully tested in practice and the results of complexity measurements are presented here. Plugging a testing server (2x Pentium III/1.4 GHz, 1 GB RAM, 100 Mb/s Ethernet, OS RedHat 7.2, Apache 1.3.27), it was possible to achieve a speed of

67.7 BVO calls per second for a 1024 bits RSA key. The median time for a whole attack on the *premaster-secret* could be then estimated as 54 hours and 42 minutes. We also propose and discuss countermeasures, which are both cryptographically acceptable and practically feasible.

3.7 Chapter G. Key-collisions in (EC)DSA: Attacking Non-repudiation

A new kind of attack on the non-repudiation property [6] of digital signature schemes is presented. We introduce a notion of key-collisions, which may allow an attacker to claim that the message (presented to a judge) has been signed by someone else. We show how to compute key-collisions for the DSA and ECDSA signature schemes [4] effectively. The main idea of these attacks has been inspired by the well-known notion of message-collisions, where an attacker claims that the signature presented at the court belongs to a different message ([9], [20]). Both of these collision-based attacks significantly weaken the non-repudiation property of signature schemes. Moreover, they weaken the non-repudiation of protocols based on these schemes. It is shown that key-collision resistance of the (EC)DSA schemes requires the incorporation of a mechanism ensuring honest generation of (EC)DSA instances. The usage of such a mechanism shall be verifiable by an independent third party without revealing any secret information. We propose and discuss basic general countermeasures against key-collision attacks on the (EC)DSA schemes. We also show that the whole notion of key-collisions can be regarded as a platform for generalization of attacks discussed by Massias, Serret Avila, and Quisquater in [8]. The fact, that the area of key-collision attacks is not solved by the standard [4] itself, again emphasizes the main motto of modern cryptology saying that *standards are clearly not enough*.

Acknowledgements

First of all, the author is grateful to his postgraduate supervisor Petr Zemánek and his colleague Vlastimil Klíma. Their inspiring suggestions, encouragement, and close cooperation made many things possible.

I would also like to thank to Pavel Rydlo for a technical co-operation in practical implementation of the attack described in chapter B and to Ondřej Pokorný, Jiří Hejl, Roman Kalač, and Libor Kratochvíl for their help and consultations with a practical realization of the attack presented in chapter F. I found also beneficial those inspiring and encouraging comments from anonymous referees of the international workshops NATO CATE SPI 2001, NATO CATE SPI 2003, CHES 2002, and CHES 2003. Special thanks also belong to Pavel Třešňák, Jared Smolens, and the companies ICZ a.s. and eBanka a.s.

References

1. Black, J. and Urtubia, H.: *Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption*, In Proc. of 11th USENIX Security Symposium, San Francisco 2002, pp. 327-338
2. Bleichenbacher, D.: *Chosen Ciphertexts Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1*, in Proc. of CRYPTO '98, pp. 1-12, 1998
3. Extensions and Revisions to PKCS #7 (Draft PKCS #7 v1.6), An RSA Laboratories Technical Note, May 13, 1997
4. FIPS PUB 186-2: *Digital Signature Standard (DSS)*, National Institute of Standards and Technology, January 27, 2000, update: October 5, 2001
5. FIPS PUB 197: *Advanced Encryption Standard (AES),* National Institute of Standards and Technology, November 26, 2001
6. Landwehr, C.-E.: *Computer Security*, International Journal of Information Security, Vol 1, Issue 1, pp. 3-13, Springer-Verlag, 2001
7. Manger, J.: *A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1*, in Proc. of CRYPTO 2001, pp. 230-238, 2001
8. Massias, H., Serret Avila, X., and Quisquater, J.-J.: *Timestamps: Main issues on their use and implementation*, In Proc. of IEEE 8th International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises-Fourth International Workshop on Enterprise Security, pp. 178-183, June 1999
9. Menezes A.J., Oorschot P.C., and Vanstone S.A.: *Handbook of Applied Cryptography*, CRC Press, 1997
10. NIST Special Publication SP 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation, December 2001
11. PKCS#1 v2.1: *RSA Cryptography Standard*, RSA Labs, DRAFT2, January 5 2001
12. PKCS#5 v2.0: *Password-Based Cryptography Standard*, RSA Labs, March 25, 1999
13. PKCS#7 v1.5: *Cryptographic Message Syntax Standard*, RSA Laboratories, November 1, 1993
14. Rescorla, E.: *SSL and TLS: Designing and Building Secure Systems*, Addison-Wesley, New York, 2000
15. RFC 2246: Allen, C. and Dierks, T.: *The TLS Protocol*, Version 1.0, January 1999
16. RFC 2268: Baldwin, R. and Rivest, R.: *The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms*, October 1996
17. RFC 2440: Callas, J., Donnerhacke, L., Finney, H., and Thayer, R.: *OpenPGP Message Format*, November 1998
18. Rivest, R., L., Shamir, A., and Adleman L.: *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, pp. 120-126, 1978
19. Shoup, V.: *A Proposal for an ISO Standard for Public Key Encryption (version 2.0)*, September 17, 2001
20. Stinson, D., R.: *Cryptography – Theory and Practice*, CRC Press, 1995
21. Vaudenay, S.: *Security Flaws Induced By CBC Padding - Application to SSL, IPSEC, WTLS...*, EUROCRYPT '02, pp. 534-545, Springer-Verlag, 2002

A. Side Channel Cryptanalysis – An Overview*

1. Introduction

It is becoming a well-known fact that the proper design of a particular cryptosystem in "paper form" is one thing, but its implementation into a physical device is another one. The main security risk here comes from evaluating only the mathematical properties of the designed cryptosystem, while at the same time underestimating the physical properties the system will have after its implementation in the "real world".

Almost all physical properties (including an electromagnetic emanation) of the cryptographic module that can be precisely measured or carefully altered can be used for some kind of attack. These attacks however are not visible in the pure mathematical description of the given cryptosystem. This is why contemporary cryptography tends to be a highly interdisciplinary science requiring balanced skills in mathematics, physics and electrical engineering.

In this chapter, we will show the general concept of attacks based on the physical properties of attacked modules. We will try to keep the focus on the general properties of these attacks as much as it will be possible. However sometimes we will need to introduce some practical example. In such cases, we will use the asymmetrical cryptosystem RSA [46] because it seems to be the most illustrative cryptographic mechanism for such a purpose. Many of the attacks discussed later on have been successfully tested on this mechanism.

2. The Concept of Side Channels

There were times when cryptographic experts were primarily concentrated on designing and analysing basic cryptographic schemes, such as encryption algorithms, signature schemes, authentication protocols, etc. Interconnections among these mechanisms were very rarely considered and the problem of their physical implementation was nearly completely neglected. Since about 1996 (except for secret intelligence services), this approach has been belonging in museums and chronicles of cryptography only. The reason is simple: If we examine the basic mathematical description of a selected mechanism only, without considering its physical implementation, we are enclosed in an abstract world that is, less or more, different from the real world where the designed tool is to be used. Our view is thus sort of foggy and some serious defects can easily be overlooked.

---

* An extended version of the paper: Rosa, T.: *Future Cryptography: Standards are not Enough*, in Proc. of Security and Protection of Information 2001, Military Academy in Brno, pp. 237-245, NATO-IDET, Brno, 9. - 11. of May, 2001.

The entire situation is illustrated by figures 1 and 2. In the first one, we can see the mechanism being examined in the way we imagine it in the abstract mathematical world in which it was designed and analysed. In the second figure, the same mechanism is shown, however, considering appropriate implementation details. As a simplification, we can say that the difference consists in the fact that the second figure shows the communication channels that were not considered by the author and that can transfer information that was not even fancied by the cryptanalyst examining the security level. Additionally, these channels can be duplex under some circumstances. An attacker is allowed to not only monitor the internal behaviour of the attacked system; she can even make the system change its internal status according to her wishes. In view of this, a mathematically quality cryptoscheme may turn out to be completely useless.

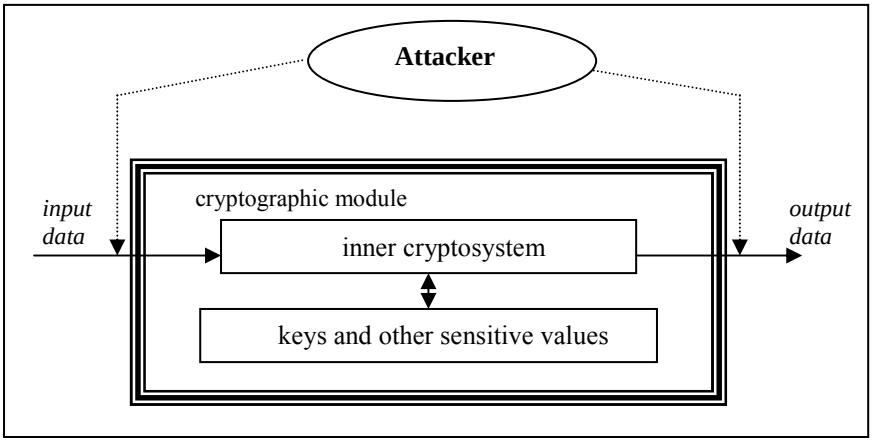

Fig. 1. An attack model without considering the physical module construction

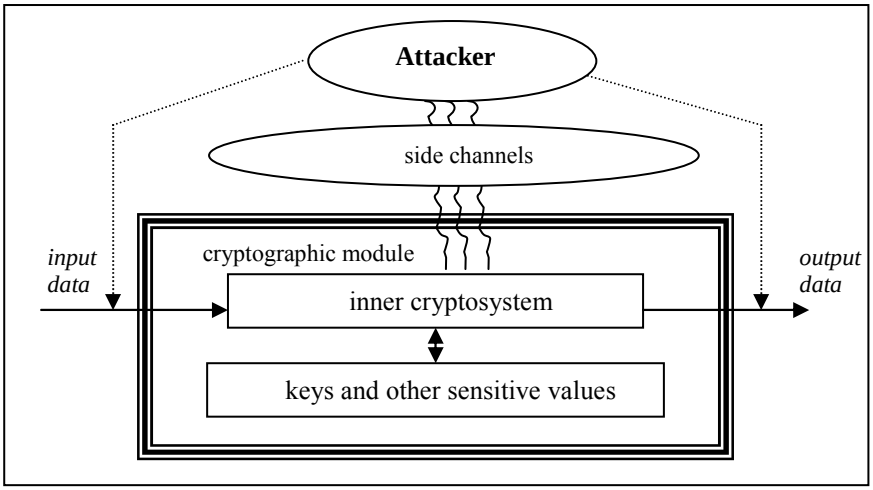

Fig. 2. An attack model with considering the effect of the module physical properties

2.1 Initial Experiments and Observations

One of the first works seriously discussing the influence of cryptographic modules architecture to their security was probably the article [29]. The author realizes the usefulness of the information that an attacker can get by carefully measuring the duration of an asymmetric cryptographic transformation which uses an unknown private key. This information turned out to be potentially very useful and can result in revelation of the private key for some cryptographic mechanisms, such as RSA, Diffie-Hellman's protocol, or DSS signature scheme (for an overviewing description of these mechanisms see [36], [53]). The article [5] was published in the same year. Its authors, more informally indeed, but very hard warn about the possibility of abusing the channels sketched in figure 2 for attacking chip cards. They weaken particularly the dogma established until that time that a chip card is an autonomous device whose physical protection provides a sufficient protection against external attackers for the internal cryptographic mechanisms of the card. While the attack described in [29] considered the channels shown in figure 2 as a simplex way from the module to the attacker (the attackers just "read" the computation time), the authors of this report demonstrate the possibility of an unauthorised affection of the operations of the attacked module and its consequences, i.e. they show that an attacker can also use side channels to “write” some information into the module. Such an injection of unauthorised information is often used as a primary step which then opens a simplex fault side channel going from the module to the attacker (c.f. fault side channel attacks later on in the thesis).

As an example, let us illustrate the attack by the article [5] which has a real base in the area of attacks on prepaid TV cards. Let us suppose that there is a sequence of commands (written in a C-like pseudocode) in the attacked device that is used for a serial transmission of the output data:

```
1. b = output_message_address
2. a = output_message_length
3. if (a == 0) goto 8
4. Send(*b)
5. b = b+1
6. a = a-1
7. goto 3
8. ... //continue after sending the message
```

In this case, the attacker aims at affecting the attacked device processor behaviour to either ignore the conditional jump in the 3rd line or not to decrement the counter in the 6th line. This can be achieved by, e.g. a short power impulse or by a short disturbance of the clock signal. The concrete technique strongly depends on the device (mostly a chip card) construction. If this is successfully completed, the originally harmless sequence of commands used for the output of a common message can then be used for reading of the memory contents over the extent of the data being sent. If the attacker is a bit lucky, she may get access to private keys or other sensitive information stored behind the addresses of that message.

As another illustration example of the existence of communication channels that are invisible at first sight, we will describe the experiment that was performed at the University of Cambridge and described in the article [7], c.f. also [52], [20]. We will show how a common workstation monitor can be used in order to create an electromagnetic side channel and to transmit selected information over that channel. A channel created by transmitting an amplitude modulated (AM) signal is used for the transmission. In general, with respect to a formal terminology introduced later on in this chapter, we talk about a kleptographic side channel here which, for example, can be used by a Trojan horse to transmit some secret information from a computer that is disconnected from any network an therefore mistakenly considered safe.

Let us briefly recall the mathematical function describing a general AM signal. Denoting $s(t)$ the value of the signal at time t, we can write:

$$s(t) = A*\cos(2\pi f_c t)*(1+m*\cos(2\pi f_t t)) = A*(\cos(2\pi f_c t) + (m/2)*\cos(2\pi(f_c - f_t)t) + (m/2)*\cos(2\pi(f_c + f_t)t)\ ).$$

In the formula above, we denoted the signal amplitude as A, the modulation depth as m, the carrier frequency as f_c, and the modulated (tone) signal frequency as f_t. We can see that, unlike for a voice signal transmission, a very simple cosine signal transmission is assumed here. However, this does not represent a serious limitation for us, as we are concentrating solely on a digital signal transmission. For example, we can use the frequency modulation (actually the FSK – Frequency Shift Keying method, since the frequency shift is discrete in this case) of the cosine signal transmitted assuming that a frequency f_t represents the value of 1 and a different frequency f_t' ($f_t' \neq f_t$) represents the value of 0. A possible way of implementation of the frequency shift will be described later on.

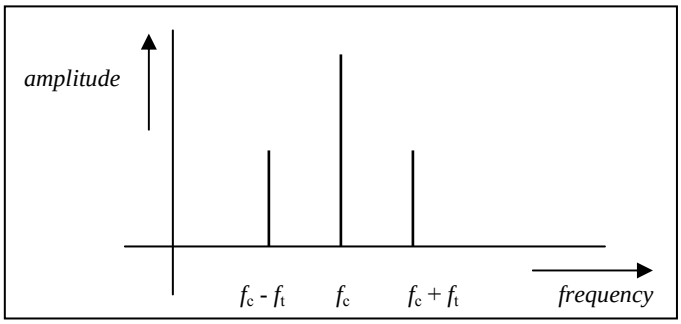

Fig. 3. Frequency spectrum of an AM signal

Looking at the expression describing the signal $s(t)$, we can see the well-known AM signal frequency spectrum distribution consisting of the carrier frequency f_c and the frequencies of so-called lower and upper sidebands: f_c-f_t and f_c+f_t. A graphical illustration of the distribution is shown in figure 3. Although this AM signal attribute was not utilised in the original experiment, it is good to mention it here because from the ratio of amplitudes in the figure, we can see that most energy is consumed for emission of the periodical signal with the frequency f_c that does not carry any useful information (it is independent on f_t). Therefore, there is a quite common practice

about transmitting an AM signal, that the carrier frequency together with one of the sidebands are suppressed prior to the start of transmission and only the remaining sideband is transmitted instead. This signal conditioning results in a more efficient use of the transmitter performance. The disadvantage of this solution is the necessity of restoring the original carrier frequency at the receiver's side, which is not always simple, especially if the original carrier frequency is not very stable. The main reasons why the experiment authors avoided this way of transmission were probably the impossibility of using a commonly available AM receiver. However, in case of elaborating their experiment with the aim of a practical application for intelligence purposes, this way of transmission should also be considered.

Let us discuss how the signal $s(t)$ can be emitted using a workstation monitor now. To be able to answer this question, we need to become familiar with the basic parameters that describe the process of displaying data on screen. The very basic value is the pixel frequency f_p, whose inverse value determines the time the beam needs for moving from a pixel (x, y) to the pixel $(x+1, y)$. Based on this frequency, the horizontal (row) frequency can be defined as $f_h = f_p/x_t$, where x_t is the horizontal resolution of the screen (including so-called hidden pixels). Analogically, the vertical (frame) frequency can be defined as $f_v = f_h/y_t$, where y_t represents the vertical resolution of the screen (including the hidden pixels).

The visible area of the screen is defined by the horizontal resolution x_d and the vertical resolution y_d (both in pixels). Due to technological reasons, this resolution is lower than the resolution defined by x_t and y_t. The time that would be needed for displaying the hidden pixels in both horizontal and vertical directions is used for returning the electron beam to the beginning of the next line or for returning it to the first row of the screen respectively (the row and frame beam flybacks). These seven values (f_p, f_v, f_h, x_t, y_t, x_d, y_d) are variables that depend on the displaying mode selected. Determining these values in some operating systems is easier than in others. For example, we can determine them quite simply in Linux, where in the file */usr/lib/X11/XF86Config* (or its equivalent in another X-server) a line like the following can be found:

```
ModeLine „1152x900“ 95 1152 1152 1192 1472 900 900 931 939.
```

We can see that the displaying mode mentioned uses the parameters $f_p = 95$ MHz, $x_d = 1152$, $y_d = 900$, $x_t = 1472$, $y_t = 939$. The frequencies f_h and f_v can then be computed basing on the formulas mentioned above: $f_h = f_p/x_t = 64.5$ kHz and $f_v = f_h/y_t = 68.7$ Hz.

Basing on these values, we can determine the essential formula that defines the absolute time when the electron beam will be in a pixel (x, y), where $0 \le x < x_d$, $0 \le y < y_d$, and we assume that at the time $t = 0$ the beam is in the pixel (0, 0). For the time $t_{(x,y)}$ needed for achieving a pixel (x, y) we can write:

$$t_{(x,y)} = x/f_p + y/f_h + n/f_v,\ n \in \mathbf{Z}.$$

The term n/f_v is related to the periodical screen refresh, the meaning of the remaining members is obvious. Let us denote $a_{(x,y)}$ the attribute value for a pixel at the position (x, y). If we want the electron beam (together with the monitor circuits) to emit a selected signal when displaying an image on the screen, we need to ensure that the attribute value $a_{(x,y)}$ corresponds with the amplitude of the emitted signal at the

time $t_{(x,y)}$. In other words, that $a_{(x,y)} \sim s(t_{(x,y)})$. To avoid an undesired phase shift of the emitted signal during the transition to a new frame, the value $1/f_v$ should be an integer multiple of the period of $s(t)$. In our case, it means that the frequencies f_c and f_t should be integer multiples of f_v.

In the experiment being described, the colour resolution of 8-bit grey scale was used and the attribute value was determined by the formula:

$$a_{(x,y)} = \lfloor 255/2 + s(t_{(x,y)}) + R \rfloor,$$

where R is a smoothing factor with an uniform distribution in the interval <0, 1) whose purpose is to distribute the noise created by the signal $s(t)$ quantization. The following parameters were chosen for the transmitted signal: $A = 255/4$, $m = 1$, $f_c = 2.0$ MHz, and two alternative frequencies $f_t = 300$ Hz and $f_t' = 1200$ Hz.

The signal described above could have been received by a simple AM receiver that (according to the authors' statement) when tuned up to the carrier frequency 2.0 MHz, markedly reproduced both 300 Hz and 1200 Hz tones in the entire lab room and the neighbouring rooms (unfortunately, no disposition plan has been supplied). This proves that selected information could have been successfully transmitted in this way. It is important to note that only a very simple receiver equipped with a basic un-tuned ferrite antenna was used. It is also very interesting that in the locations where the signal was starting to fade, getting the receiver closer to the power line the tested monitor was connected to was of a marked help. This is a good example illustrating how surprising the holes for leaking of information out of a computer can be.

Prior to trying to reproduce this experiment, we have to take into account that the radio receivers commonly available nowadays support the AM bandwidths ranging from 531 to 1602 kHz (MW) and from 153 to 279 kHz (LW) only. This means that we need to either change the experiment parameters or use a non-standard receiver. Higher frequencies seem to be better (considering the construction configuration). If the receiver type used does not limit us, the bandwidth from 3 MHz to 30 MHz appears to be the best for such an information transmission.

Using the FSK for the transmission of logical levels together with an appropriate error-control encoding, a transfer rate about 50 bits/s can be reached. To create the frequency shift of the transmitted signal, we can take an advantage of the same trick that is used in animations: Two different images for f_t and f_t' are prepared in two separate banks of the video RAM. By setting one of these areas active, an attacker's program can then quickly choose which frequency (i.e. logical level) should be transmitted.

The transfer rate mentioned above does not seem to be very high at first sight, however, we need to realize that the attacker will probably be most interested in values such as keys used for data encryption, etc. For this purpose, the transfer rate is high enough.

To simplify the experiment described above, we discussed (similarly to the authors of [7]) the use of a monochromatic signal only. However, most of the displaying devices used nowadays are colour monitors with RGB screens equipped with three parallel cathodes emitting the relevant components of the colour signal (red, green, blue) independently. Additionally, a separate electronic circuit controls quite independently each of these beams. This means that we could transmit three separate

signals $s_R(t)$, $s_G(t)$, and $s_B(t)$ using a colour signal, creating thus three independent parallel channels for leaking of information from the attacked workstation.

2.2 The Term "Cryptographic Module"

The topic discussed here might seem to be limited to hardware devices only but it is not. If we discuss the side channels problems generally further on, we will prefer using the common term *cryptographic module* as an elementary building unit used to transform an abstract description of a particular cryptographic scheme into somewhat "practical". This module can be realized by means of either software or hardware resources. Of course, all software needs to be run on a hardware device. This invokes the question if we should not better always be talking and writing about hardware modules-devices only. This wouldn't apparently be a very good choice, as our aim is (besides others) to determine the phase of the process of transition from a purely abstract description of the implemented cryptoscheme, during that the undesired side channels (by the definition below) emerge. In other words, we are wondering how many statements of a purely physical nature we need to add to an originally purely mathematical model to be able to "notice" emerging side channels also in the formal description of the examined module. In some cases, all we need to know are the properties of the software that implements the cryptographic scheme – we say that we found a side channel in a software module then. In other cases, we also need to add a physical description of the hardware device the software is run on – we are talking about a side channel in a hardware module then (the word "device" can be used instead of "module" here).

Of course, there are types of side channels that can be identified solely in hardware modules, such as a power side channel. However, there are also many types of side channels that can be found using the software description only. These types include many fault side channels. As concrete example of an attack on a software module using a fault side channel we can mention those ones described here in chapters B, C, E, and F. Moreover, basing on a general physical properties which are known to be exhibited by many hardware devices, we can point out a probable susceptibilities to "purely hardware" side channels by examining solely the software part of the module. Examples of such reasoning are given in chapter C.

Actually, basing on a recent research observations, we may conclude that the most suitable technique for designing and analysing cryptographic modules is the one based on so-called CODESIGN approach [24], [48].

2.3 Formal Definition

Similarly to other cryptanalytic techniques, the side channel attacks were initially treated as a special kind of "ad-hoc" invention and the efforts for a formal description and examination of these attacks were not paid much attention to (except for [29]). Later on, however, these attacks turned out to be forming so interesting category that it was worth examining them generally. Unfortunately, this does not mean that we have some exceptionally good theoretical materials available in this area nowadays.

Most of the authors are unfortunately too conscious of the value of this kind of information, so readers, after making a study of a very pretentiously looking material, sometimes finally realizes that it actually has not brought them anything new.

In this study, we can contribute to the enrichment of the topic discussed by introducing a classification scheme that will be general enough to cover all the present techniques and in the same time concrete enough to be useful for working with. The term *side channel* appeared for the first time in the article [28], where an informal definition was given. The following formal basis which (among others) separates the terms *side channel*, *side channel analysis*, *side information*, *side channel attack*, and which also shows how to incorporate the notion of fault attacks into the area of side channels, was (up to author's knowledge) for the first time given during the presentation of the paper [47].

Definition 1 (Side channel). *Any undesirable way of information exchange between a cryptographic module and its neighbourhood is referred to as a* ***side channel (SC)*****.**

There are plenty of ways in which a particular side-channel can be realized. Today researchers work mainly with timing characteristics, power consumption measurements, and newly also with an electromagnetic emanation. Also challenging is research in the area of so-called multi-channels which are constructed as a joint contribution of some different channels [2]. A distinguished kind of side channels are fault and kleptographic channels which will be discussed later on. Also note, that there may be output as well as input side channels, where the later ones allow an attacker to change the behavior of the module by precisely altering the medium which transmits the particular side channel signal. For example, one may use an electromagnetic field for such a purpose [45]. But these input side channels are not studied from the same point of view as the output ones, yet. A possible reason for doing so is that almost every such an input side channel is actually used to induce an output side channel which is usually then referred to as a fault side channel (i.e. the channel induced by computational faults; its own signal is then usually hidden in error messages normally communicated out from an attacked module). On the other hand, we may reasonably expect the input side channels to become also more formally studied in a future research. However, in this thesis, we will focus solely on the output side channels without further explicit mentioning of this scope.

In the relevant literature, a definition of the term *side channel* has mostly been left for its intuitive comprehension so far, which may result in serious misunderstandings. Please note for example, that in the definition mentioned above, a side channel does not necessarily disclose sensitive information, which is intuitively implied by most readers. Such misunderstanding may, for instance, cause serious problems in the area of power and or electromagnetic side channels. To clarify the problem, let us suppose we have a cryptographic device on which we have detected a power side channel able to disclose sensitive information. Let us also suppose we have modified the software part of the module to prevent it from disclosing this information. This modification has been performed (as has been usual hitherto) directly on the cryptographic scheme used, rather than on the hardware construction of the device itself. The danger of the sensitive information disclosure has thus been warded off. However, the side channel may persist! Only the information it transmits is useless for the present. Therefore, we must take into account that the persisting side channel might suddenly strike the

module again, when somebody finds another way on how to use the signal coming from it. So finally, eliminating a particular side channel attack does not necessarily mean eliminating a particular side channel itself.

If the requirement of sensitiveness of the information transmitted were included in the side channel definition, the modification mentioned above would simply make the entire channel invisible. But this would not reflect the reality. As said before, the channel can persist and we just don't know at the moment how to utilize (e.g. for an attack) the information it produces. This represents a substantial difference in the perspective of the problem, as now we are still motivated to think about the possibility to find a way of processing the information transmitted by the side channel to use it for an attack.

If we are discussing the problems of a practical use of the information transmitted by a side channel, it is the right time to define the term *side channel analysis* (SCA) now.

Definition 2 (Side channel analysis). *A procedure of getting information from a side channel is referred to as a* ***side channel analysis.***

Definition 3 (Side information). *The information obtained by a side channel analysis is referred to as* ***side information.***

Please note, we still are not saying anything about whether the side information distilled has to be sensitive or not. This will allow us to separate the problems of capturing the information transmitted by a side channel from the problems of utilizing this information for an attack on the module. This approach sounds useful, since the tools used for the side channel analysis are mostly represented by commonly used discrete signal processing techniques, while modified methods of the standard cryptanalysis are used for attacks. Now, we have finally reached the right moment to clarify the concept of a side channel attack.

Definition 4 (Side channel attack). *A process of using side information to attack a cryptographic module is referred to as a* ***side channel attack.***

Figure 4 summarizes the known types of side channels, side channel analysis techniques, and their combinations. It indicates how the side channel type names are combined with the analysis technique names or the side channel attack names. Please note, that the acronyms for an attack and for the corresponding analysis technique are the same, simply because the words "attack" and "analysis" have the same initials. Therefore we need to distinguish between an analysis and an attack based on the actual context. However, rather than a drawback, this is apprehended as an advantage. As mentioned before, most of the relevant literature is not based on such general definitions of concepts and intuitively uses acronyms established formerly (that were, more or less intuitively again, adopted by the authors of primary relevant paper). The fact that *attack* and *analysis* are not distinguished by the acronyms allows covering quite many existing papers (whose terms were not adopted based on this general approach) by the established terminology.

2.4 Simple and Differential Analysis

Roughly speaking, *simple analysis* usually refers to a process of extracting side information directly by observing the signal coming from a side channel. A good example of this could be direct reading of DES key bits by observing distinguished voltage peaks in a power consumption trace of an attacked device (c.f. [31]). However, with respect to power and or electromagnetic side channels, there are very few such cases in which the information is encoded in some directly "readable" format (though such setups usually give surprising results then - [22]). In all the other cases it is scrambled in some device-proprietary way. Despite this, it doesn't seem to be an unsolvable problem to decode these streams, mainly because of the availability of strong statistical techniques of signals' theory. Here comes the term *differential analysis* which is usually used to refer to a more complicated side information extraction where some sophisticated statistical tools need to be used (c.f. [30], [31]). We note that this categorization was primarily created during research on analyses of power side channels. However, it turns out that adopting it also for a study of other side channels is highly advisable. This can demonstrate the existence of some closer relations among the natures of these side channel types that have been being apprehended as separate problems in most of the research papers so far.

Side channel types

- Timing
- Power
- Electromagnetic
- Fault
- Kleptographic

Basic analysis types

- Simple Analysis
- Differential Analysis

Commonly used acronyms

- Timing side channels
  - Simple Timing Analysis/Attack - **STA**
  - Differential Timing Analysis/Attack - **DTA**
- Power side channel
  - Simple Power Analysis/Attack - **SPA**
  - Differential Power Analysis/Attack - **DPA**
- Electromagnetic side channel
  - Simple Electromagnetic Analysis/Attack - **SEMA**
  - Differential Electromagnetic Analysis/Attack - **DEMA**
- Fault side channel
  - Simple Fault Analysis/Attack - **SFA**
  - Differential Fault Analysis/Attack – **DFA**
- Kleptographic side channel
  - Kleptographic Analysis/Attack - **KA**

Fig. 4. Basic categorization of side channels and their analyses

2.5 Special Analyses and their Relations to the Differential Ones

Introducing the term differential analysis may appear as being not enough to grasp all statistically matured analyses including, for example, the analysis used for promising template attacks [16]. According to their authors, the attacks seem to be the strongest form of side channel attack possible in an information theoretic sense. They can break implementations and countermeasures whose security is dependent on the assumption that an adversary cannot obtain more than one or a limited number of side channel samples. At first look, it seems to be advisable to separate the analysis used here from the differential ones mentioned above. On the other hand, if we assume that term differential analysis is here to be used for *any* statistically matured approach to getting side information from a side channel signal, then there is no problem about that. Since the terms *simple* and *differential* are rather of engineering than mathematical origin, we will support this viewpoint by a rather heuristic argument. If we look closely at statistical techniques used for signal detection and information extraction, we see that they are actually based on a more or less sophisticated hypotheses testing. Here, these hypotheses are usually derived from assumptions placed on some inside variables whose secret values an attacker wants to learn. By proving or disproving the hypotheses the attacker also proves or disproves her assumptions about the secrets. For example, let the hypothesis be derived from an assumption that the first byte of a key belongs to the interval <0, 127>. By disproving the hypothesis, however, the attacker learns that the key byte probably lies in <128, 255>, so its most significant bit is probably 1. The attacker then continues in such a way until she learns the value of the whole key. Let us briefly look through common properties of hypotheses testing. In the heart of such a process, we can usually identify some factor ε whose value determines whether the particular hypothesis is to be accepted or refused. Therefore, heuristically speaking, there is a certain measurable distance between hypothesis acceptation and rejection which can be regarded as a *difference* between valid and invalid assumptions about the secret values. So, the process of learning the secret is intrinsically based on intensive working with such differences, which satisfies using the term *differential* analysis.

At least from a practical point of view, we suggest to broadly accept the term differential analysis together with its spread meaning. Rather then introducing brand new kinds of side channel analysis, we should then speak about special kinds of differential analysis. Here is also a place for already existing terms like high-order differential analysis (e.g. HO-DPA, HO-DEMA) [3], [37], etc. Using the above mentioned notation, the analysis used in [16] can be then, for example, referred to as *template differential analysis* (e.g. TEMPLATE-DPA, TEMPLATE-DEMA, etc.).

2.6 Fault Side Channels

There are two major kinds of fault side channels. The first ones are channels which are induced by computational faults occurring during cryptographic computation in an atacked module. These faults can be either random or intentional, caused, for instance, by a precise voltage manipulation [8]. Having the ability to introduce computational faults, this kind of attack can be used on almost every kind of

cryptographic mechanism and they are regarded as the most effective side channel attacks at all. The criterion here is the number of required interactions between an attacker and a cryptographic module being attacked.

We shall also note that although we expect the faulty behavior to be caused mainly by the effort of an attacker, there can be also errors of the "natural" type. It is as important to avoid these faults as to defend the module against attackers. This is because when the error, which causes the leakage of the secret information, happens, it doesn't matter what was its reason. We may imagine a worldwide spying agency, which passively monitors every important system (for example major certification authorities - CA - used for a certification of user's public keys) waiting for its faulty output. When this happens the agency gains the secret information from this particular system (usually the private key of this CA) and continues spying on the other stations. If designers of these systems were not aware of FA attacks, then this agency would sooner or later have private keys of the major part of certification authorities in the world.

We note that the side channels of this first kind can also be induced by errors in static ephemeral data used by the module. This data can be, for instance, keys and we give examples of attacks based on such channels in parts B and C. It follows that cryptographic modules must be not only protected against computational faults but they must also properly check the integrity of all static data which are used in cryptographic computations.

The second kind of fault side channels are those which are induced by sending an intentionally corrupted input data to the attacked module. For the module, this means a non-standard situation which must be handled in a special way. Usually the module has to use an error message to inform the user (the module can hardly know whether this is an ordinary user or an attacker) that the computation has been stopped due to some reasons. It turns out, that such error messages can also carry very sensitive information allowing an attacker to break the cryptosystem implemented by this module. Despite being sometimes a bit less effective than the attacks based on the first kind of side channels, these attacks can also pose a significant threat to a whole system. We give examples of such attacks in parts E and F.

Although we have introduced the notion of simple and differential analyses also for fault side channels, this was mainly for a purpose of completeness and, probably, some further research. Today, almost all fault side channels can be analyzed simply by a simple analysis, since, in fact, the side information is transmitted directly through output data or error messages. Therefore, these analyses and or attacks belong to the SFA class and are usually shortly referred to as FA (fault analysis/attack). However, it may by realized during some future research, that more complicated analysis of this data is needed and the term DFA may become useful. Actually, one such an example has been already described in the paper [11].

2.7 Kleptographic Side Channels

A kleptographic channel is a very special side channel type. This is mainly because of the perspective from which its existence is being viewed. Basically, the kleptographic channel is often a subliminal channel (see [50], [4]) and a subliminal channel cannot

be generally regarded as a side channel. Let us briefly mention the reasons why. The subliminal channels in cryptography were discovered to transmit an additional, hidden information in the standard output stream from a cryptographic module. From this point of view, these channels act in the same way as the subliminal channels used for transmitting hidden advertisements on TV or in movies (prohibited in most countries). In relation to the subliminal channels in cryptography, various forms of their use by intelligence services are speculated, see, for example, their outline in [4]. However, these channels are always intentionally designed and (from the module designer's viewpoint) desired, and therefore cannot be generally referred to as side channels. Having discussed the term subliminal channel, let us go on to another term now. The term *kleptography* [54] was adopted in connection with the trend of implementing cryptographic modules as black boxes used by users that do not (or even must not) know the internal processes in these modules. In relation with this trend, the problem of the trustworthiness of such modules for users arose. Users were asking if there was a possibility of transmitting (in the background of the standard output data) some hidden information that could be addressed e.g. to an intelligence service. The intelligence agency then would be able to use such information to decrypt the user messages, etc. The subliminal channels turned out to be a very good solution for these purposes. Using subliminal channels for cryptographic purposes is then referred to as a kleptographic side channel that is, within the meaning of the definition we introduced previously, a side channel from the user's point of view, as it is probably an undesired channel for her.

Note that in the case of kleptographic side channels, it does not seem desirable to introduce the notion of simple and differential analysis over them. Because of their nature and the way they are built into a cryptographic module, the analysis of their signal can almost always be referred to as a "very special", not having apparent properties of simple nor differential analysis. Maybe, there will be used the term *steganographic analysis* in a future.

2.8 A Note on an Information-Theoretic Approach and the Concept of Covert Channels

In the definitions related to side channels given in this text, we are, in a certain way, approaching an information-theoretic viewpoint of this problematic. It is well known in the area of computer security [36], [12] that information theory usually allows us to obtain strong theoretical results which are, however, hard to achieve in a practice. Nevertheless, it does not mean that we should not make an effort to get as closer as possible to an information-theoretic description of at least certain parts of the side channels area. As an interesting inspiration, we may use the notion of *covert channels*, c.f. [42], [41], [12]. This theory studies *undesired ways which allow a user or a group of users to exchange some information with another user or a group of users via a shared information system* [41]. Investigating all such ways is an important step, for instance, when we are building up an information system designed to work with sensitive classified information. Certain aspects of the concept are very similar to the theory of side channels which is circa 20 years younger. The main difference here is that covert channels are focused on the information flow activated

by a user's will to communicate with somebody else, while an information flow through side channels is usually activated spontaneously by physical properties of a cryptographic module. Of course, there are certain overlaps of both concepts. For instance, covert channels can be used for a creation of kleptographic side channels. In fact, a subliminal channel mentioned above is just a particular realization of a covert channel. On the other hand, it does not seem necessary to merge the concepts under one theory right now (though, we can easily imagine such merging in a future). For now, it would be enough to seek for some interesting ideas from the longer matured area of covert channels and try to implement them into the developing area of side channels.

2.9 Classification by the Level of Control over the Attacked Module

Besides the classification by the type of side channel and its analysis, we will introduce another classification scheme aimed to distinguish among concrete attacks. This scheme will classify attacks by the level of control over an attacked module that an attacker needs for performing a successful attack. We use the following two independent classifications.

By the control over the computation process, the attacks can be divided to:

Passive – the attacker just monitors the process.

Active – the attacker can alter the computation, e.g. by selecting input parameters.

By the way of accessing the module, the attacks can be divided to:

Non-invasive – no additional protection is trespassed.

Invasive – some of the additional protections (e.g. shielding) are trespassed.

The classification mentioned above will help us better understand under what circumstances a particular attack type can occur. For example, if we etched the protective layer of a chip card and connected to the processor's internal bus in order to monitor the data we would perform a *passive invasive* attack. However, the classification is suitable mainly for attacks on concrete physical devices. It is hard to give an exact attack category when discussing an attack on a cryptographic standard and or a protocol, since the standard and the protocol can usually be implemented in many different ways. According to this reason, the classification will not be explicitly used in the papers presented in this thesis, since these describe mainly the attacks on cryptographic standards and or protocols.

3. The Terms TAMPERING and TEMPEST

When reading the relevant literature, we often encounter the terms *tampering* and *tempest*. These terms belong in an area which is very close to the side channels but the relations between these areas have not been precisely defined yet. It is caused by the fact that the terms *tampering* and *tempest* were adopted much sooner than the side channels theory was created and their definitions are still quite vague.

The term *tampering* mostly refers to an attack involving a physical action on the device attacked. These actions include simple physical invasion (such as removing the cover or etching the protective layer of the chip attacked) or, e.g. hampering power pulses aiming at corrupting the device normal operation ([5], [6], [51]). For example, the conditions for creating a fault side channel can be ensured in this way. Using the classification introduced in §2.7, we can say that a proper prevention of tampering techniques will eliminate all those invasive side channel attacks. However, this is only a rather small part of the whole area of side channel attacks, and therefore the protection against tampering say actually very little about the resistance against side channel attacks.

The term *tempest* (*Terminal Electromagnetic Pulse Escape Safeguard Technique*) is related to the possibility of intercepting parasitic electromagnetic radiation from a cryptographic device [7]. However, concrete ways of utilizing such information are studied and discussed not until by the side channels theory again and, up to now, there seems to be a little effort to reflect research results obtained here back to the tempest standards [1].

The terms mentioned above are frequently encountered in connection with the standardization of requirements for the properties of cryptographic modules. As an example we can mention the FIPS 140 standard (see [21]), namely its latest version referred to as FIPS 140-2. This document is often used as a base for evaluating cryptographic modules (both software and hardware). However, as has been sketched above, high resistance against tampering and a good tempest do not necessarily imply that all side channels are completely eliminated. This fact is necessary to be taken into account if we are making a judgement on a cryptographic module's resistance against the side channel attacks, based on whether the module is certified as compliant with FIPS 140 or not. Even the authors of the standard themselves are aware of this problem and point this unresolved question out.

4. Conclusion

The discovery of side channels is definitely one of the most significant events in the recent cryptanalysis. In the rapidly nascent side channels general theory, it becomes obvious that studying only an abstract mathematical description of a particular (standardized) cryptosystem when evaluating security of a whole cryptographic module is insufficient any more. This description must always be enhanced with some description of a physical nature to get the abstract concept closer to the actual environment in which the examined module is to be operated. It also becomes obvious that the cryptology is a very multidisciplinary science that combines a very wide range of purely mathematical disciplines together with certain knowledge about physics, computer science, and electrical engineering. Briefly speaking, the cryptology itself balances on the edge between mathematics and physics. If we exaggerate a little bit, we can rather talk about sort of *physical mathematics* here. We borrowed the term *mathematical physics* here and reversed the order of the disciplines names. Even though we would not like to slip into profound philosophical thoughts, we can substantiate the use of this term by the fact that, while mathematical physics

accents whether the theories applied describe the physical reality precisely enough, in cryptography (i.e. physical mathematics) we need to take care of whether the resulting cryptographic module implementation truly corresponds with its mathematical description. Both of the disciplines thus seem to be balancing on the edge between the mathematics and physics, but originally in the reverse direction to each other.

The side channels theory also significantly influences some of the well-established cryptographic principles. As one example for all, let us mention the concept of a *perfect secrecy* (also known as an *absolute security*) [36]. Informally speaking, this term refers to cryptographic constructions for which the possibility of cryptoanalysis can be totally eliminated regardless of the attacker's computing power. Based on the information theory, we can show in these cases that the attacker do not have enough information to be able to solve the cryptoanalysis task (e.g. find the plaintext for a ciphertext) correctly. From this point of view, the *Vernam's Cipher* has become renowned which is absolutely secure if applied properly. However, if we do not consider side channels effect when evaluating a system, the concept of the perfect secrecy is not very beneficial for us. Information that we suppose not to be available to an attacker can actually leak out unnoticed from one of the side channels. Of course, this does not mean that the perfect secrecy term will necessarily die out. What it means is that constructing the proof of absolute security always has to be based on such a description of the evaluated cryptosystem, which corresponds with the physical reality (i.e. includes an accurate description of threatening side channels).

In the end, let us summarize general methodical procedures we should follow when designing cryptographic modules to make them resistant to threats of attacks based on side channels. First, we should aim at identifying all the side channels possible. Of course, we should try to eliminate these side channels then, so that they completely disappear if possible. As this is not always possible (see [18]), we need to apply corrections directly in the design of the cryptosystems implemented then. Concrete examples of such corrections will be mentioned throughout this study according to concrete attacks presented further on. We should comment here the incorrect but somehow still surviving opinion that a protection against side channels is always of a purely physical nature. If we accepted this opinion, we would lose a big part of what we are trying to build in the side channels general theory. We are trying to make us not only "also realize" the physical attributes of the examined module but also combine their descriptions with the abstract mathematical models of the cryptographic schemes used. This should form a complex theory based on which all that we need to know about the module can be deduced and proven. The type and placement of countermeasures designed to improve the module security are then not decided based on intuitive assumptions (e.g. hardware countermeasures against hardware side channels). The only decisive criterion is the countermeasure efficiency. Very efficient countermeasures can be (maybe a bit surprisingly) achieved by making corrections to the mathematical descriptions of cryptographic schemes. Of course, this definitely cannot be treated as a dogma either.

The general conclusion of this part should be that the security of a particular cryptographic module does not depend only on the cryptographic standards employed. It is also a question of the way in which these standards are implemented as well as of the environment in which the module is to be used [24].

References

1. Agrawal, D., Archambeault, B., Rao, J.-R., and Rohatgi P.: *The EM Side-Channel(s)*, in Proc. of CHES 2002, pp. 29-45, 2002
2. Agrawal, D., Rao, J.-R., and Rohatgi, P.: *Multi-channel Attacks*, in Proc. of CHES 2003, pp. 2-16, 2003
3. Akkar, M.-L., Bevan, R., Dischamp, P., and Moyart, D.: *Power Analysis, What Is Now Possible...*, in Proc. of ASIACRYPT 2000, pp. 489-502, 2000
4. Anderson, R.: *Security Engineering*, John Wiley & Sons, Inc., 2001
5. Anderson, R. and Kuhn, M.: *Tamper Resistance – a Cautionary Note*, in Proc. of 2nd USENIX Workshop On Electronic Commerce, pp. 1-11, 1996
6. Anderson, R. and Kuhn, M.: *Low Cost Attacks on Tamper Resistant Devices*, in Proc. of *Security Protocols '97*, pp. 125-136, 1997
7. Anderson, R. and Kuhn, M.: *Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations*, in Proc. of Information Hiding '98, pp. 124-142, 1998
8. Aumüller, C., Bier, P., Fischer, W., Hofreiter, P., and Seifert, J.-P.: *Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures*, in Proc. of CHES 2002, pp. 260-275, 2002
9. Bao, F., Deng, R.-H., Han, Y., Jeng, A., Narasimhalu, A.-D., and Ngair, T.: *Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults*, in Proc. of Security Protocols '97, pp. 115-124, 1997
10. Bellare, M. and Rogaway, P.: *The Exact Security of Digital Signatures – How to Sign with RSA and Rabin*, in Proc. of EUROCRYPT '96, pp. 399-416, 1996
11. Biham, E. and Shamir, A.: *Differential Fault Analysis of Secret Key Cryptosystems*, in Proc. of CRYPTO '97, pp. 513-525, 1997
12. Bishop, M.: *Computer Security – Art and Science*, Addison-Wesley, 2003
13. Boneh, D.: *Twenty Years of Attacks on the RSA Cryptosystems*, Notices of the American Mathematical Society, vol. 46, no. 2, pp. 203-213, 1999, available at http://crypto.stanford.edu/~dabo/pubs.html
14. Boneh, D., DeMillo, R.-A., and Lipton, R.-J.: *On the Importance of Checking Cryptographic Protocols for Faults*, in Proc. of EUROCRYPT '97, pp. 37-51, 1997
15. Chari, S., Jutla, C.-S., Rao, J.-R., and Rohatgi, P.: *Towards Sound Approaches to Counteract Power-Analysis Attacks*, in Proc. of CRYPTO '99, pp. 398-411, 1999
16. Chari, S., Rao, J.-R., and Rohatgi, P.: *Template Attacks*, in Proc. of CHES 2002, pp. 13-28, 2002
17. Coron, J.-S. and Goubin, L.: *On Boolean and Arithmetic Masking against Differential Power Analysis*, in Proc. of CHES 2000, pp. 231-237, 2000
18. Clavier, C., Coron, J.-S., and Dabbous, N.: *Differential Power Analysis in the Presence of Hardware Countermeasures*, in Proc. of CHES 2000, pp. 253-263, 2000
19. Dhem, J.-F., Koeune, F., Leroux, P.-A., Mestré, P., and Quisquater, J.-J. and Willems, J. - L.: *A Practical Implementation of the Timing Attack*, Technical Report CG-1998/1, 1998
20. van Eck, W.: *Electromagnetic Radiation from Video Display Units: an Eavesdropping Risk?*, Computers & Security vol. 4, pp. 269-286, 1985
21. FIPS PUB 140-2: *Security Requirements for Cryptographic Modules*, National Institute of Standards and Technology, Issued May 25 2001, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
22. Fouque, P.-A., Martinet, G., and Poupard, G.: *Attacking Unbalanced RSA-CRT Using SPA*, in Proc. of CHES 2003, pp. 254-268, 2003
23. Goubin, L. and Patarin, J.: *DES and differential power analysis*, in Proc. of CHES '99, pp. 158-172, 1999

24. Joye, M., Koeune, F., Preneel, B., Rohatgi, P., Seifert, J.-P., and Walter, C.: *Are software and hardware counter-measures winning the war against side-channel leakage?*, panel session at CHES 2003, 2003
25. Joye, M., Koeune, F., and Quisquater, J.-J.: *Further results on Chinese Remaindering*, Technical Report GC-1997/1, 1997
26. Joy, M., Lenstra, A.-K., and Quisquater, J.-J.: *Chinese Remaindering Based Cryptosystems in the Presence of Faults*, Journal of Cryptology, Volume 12, Number 4, pp. 241-245, Autumn 1999
27. Joye, M. and Quisquater, J.-J.: *Faulty RSA Encryption*, Technical Report CG-1997/8, 1997
28. Kelsey, J., Schneier, B., Wagner, D., and Hall, C.: *Side Channel Cryptanalysis of Product Ciphers*, in Proc. of ESORICS '98, pp. 97-110, 1998
29. Kocher, P.: *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, in Proc. of CRYPTO '96, pp. 104-113, 1996
30. Kocher, P., Jaffe, J., and Jun, B.: *Introduction to Differential Power Analysis and Related Attacks*, Technical Report, 1998, http://www.cryptography.com/dpa/technical
31. Kocher, P., Jaffe, J., and Jun, B.: *Differential Power Analysis*, in Proc. of Crypto '99, pp. 388-397, 1999
32. Kömmerling, O. and Kuhn, M.: *Design Principles for Tamper-Resistant Smartcard Processors*, in Proc. of USENIX Workshop on Smartcard Technology, pp. 9-20, 1999
33. Lenstra, A.-K.: *Memo on RSA Signature Generation in the Presence of Faults*, manuscript, September 28 1996, partially published in [26]
34. Manger, J.: *A Chosen Ciphertext Attack On RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized In PKCS #1*, in Proc. of CRYPTO 2001, August 2001
35. Mayer-Sommer, R.: *Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards*, in Proc. of CHES 2000, pp. 78-92, 2000
36. Menezes, A.-J., van Oorschot, P.-C., and Vanstone, S.-A.: *Handbook of Applied Cryptography*, CRC Press, 1996, online at http://www.cacr.math.uwaterloo.ca/hac/
37. Messerges, T.-S.: *Using Second-Order Power Analysis to Attack DPA Resistant Software*, in Proc. of CHES '00, pp. 238-251, 2000
38. Messerges, T.-S.: *Securing the AES Finalists Against Power Analysis Attacks*, in Proc. of FSE 2000, pp. 150-164, 2000
39. Messegers, T.-S., Dabbish, E.-A., and Sloan, R.-H.: *Investigations of Power Analysis Attacks on Smartcards*, in Proc. of USENIX Workshop on Smartcard Technology, pp. 151-161, 1999
40. Messerges, T.-S., Dabbish, E.-A., and Sloan, R.-H.: *Power Analysis Attacks of Modular Exponentiation in Smartcards*, in Proc. of CHES '99, pp. 144-157, 1999
41. Millen, J.: *Covert Channel Capacity*, in. Proc. of 1987 EEE Symposium on Research in Security and Privacy, pp. 60-65, 1987
42. Millen, J.: *20 Years of Covert Channel Modeling and Analysis*, in Proc. of the 1999 IEEE Symposium on Security and Privacy, pp. 113-114, 1999
43. Muir, J.-A.: *Techniques of Side Channel Cryptanalysis*, A thesis presented to the University of Waterloo, Canada, 2001, http://www.math.uwaterloo.ca/~jamuir/sidechannel.htm
44. PKCS#1 v2.1: *RSA Cryptography Standard*, RSA Laboratories, DRAFT2 – January 5 2001, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/
45. Quisquater, J.-J. and Samyde, D.: *Eddy current for magnetic analysis with active sensor*, In Proc. of Esmart 2002, 3rd edition, 2002
46. Rivest, R.-L., Shamir, A., and Adleman L.: *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, pp. 120-126, 1978

47. Rosa, T.: *Future Cryptography: Standards are not Enough*, in Proc. of Security and Protection of Information 2001, Military Academy in Brno, pp. 237-245, NATO-IDET, Brno, 9. - 11. of May, 2001
48. Schaumont, P. and Verbauwhede, I.: *Domain-Specific Codesign for Embedded Security*, Computer, April 2003, Vol. 36, No. 4, pp. 68-74, IEEE Computer Society, 2003
49. Schindler, W.: *A Timing Attack against RSA with the Chinese Remainder Theorem*, in Proc. of CHES 2000, pp. 109-124, 2000
50. Simons, G.-J.,: *The History of Subliminal Channels*, IEEE Journal of Selected Areas in Communications, vol. 16, n. 4, pp. 452-462, April 1998
51. Skorobogatov, S.: *Copy Protection in Modern Microcontrollers*, Technical Report, 2000, http://www.cl.cam.ac.uk/~sps32/mcu_lock.html
52. Smulders, P.: *The Threat of Information Theft by Reception of Electromagnetic Radiation from RS-232 Cables*, Computers & Security vol. 9, pp. 53-58, 1990
53. Stinson, D., R.: *Cryptography – Theory and Practice*, CRC Press, 1995
54. Young, A. and Yung, M.: *Kleptography: Using Cryptography Against Cryptography*, in Proc. of EUROCRYPT '97, pp. 62-74, 1997

B. Attack on Private Signature Keys of the OpenPGP format, PGP™ programs and other applications compatible with OpenPGP*

1. Introduction

The OpenPGP format is defined in the document [3] since the year of 1998. Its objective was to publish all necessary information about the OpenPGP format so that various interoperable applications could be created on its basis. It describes the massage formats (data structures) and a manner of how they are to be created. In this chapter, we show a serious vulnerability of the OpenPGP format which consists in insufficient security of integrity of both public and private parts of signature keys of DSA and RSA algorithms. We show that this vulnerability may be used to reveal the private signature keys.

In order to protect the private key, its value is stored in the so-called private *keyring* (file) `secring.skr` in an encrypted form. A strong symmetric cipher (e.g. AES, CAST5, IDEA) with sufficiently long key is usually selected for this purpose by the user. The key is derived from a secret access password (referred to as a *passphrase*) which is known only by the user. In this study, we show that if an attacker has an access to this file (or a record), she may obtain the user's private signature key without the need to know her passphrase or without having to attack on it. The attack consists in a special modification of parameters of the signature algorithm and obtaining of a signature of any file (e.g. e-mail message) by such a modified signature key. We show that the attacker is able to compute the private signature key based on this procedure. As the corrupted record or file (`secring.skr`) can be then restored back to its original condition, this intrusion is very dangerous. Private keys transferred in encrypted form on floppy disks or over the network are endangered in the same way. The attack was practically tested on the PGP™ v. 7.0.3† program with a combination of AES and DH/DSS algorithms. Revealed private signature key of the DSA algorithm was the result of the attack.

We also propose certain security and cryptographic measures for correction of the OpenPGP format as well as changes in the PGP™ program. All other applications which are compatible with OpenPGP format shall undertake a detailed revision. This applies for instance to the GNU Privacy Guard and other applications specified on the list of applications compatible with OpenPGP, which is available at http://www.pgpi.org/products.

---

* An edited version of the paper: Klíma, V. and Rosa, T.: *Attack on Private Signature Keys of the OpenPGP format, PGP (TM) Programs and Other Applications Compatible with OpenPGP*, IACR ePrint archive 2002/076, version 1, March, 2001.

† It was the latest version of the PGP™ program at the time of writing the original paper.

The following text is organized as follows: First we recall a definition of the signature scheme of DSA and the storage format of the private keys according to OpenPGP [3]. Then we describe an idea of the attack on the private signature DSA key and the specific procedure of the attack as we performed it in the PGP™ program. Furthermore, we describe an idea of the attack on the signature key of RSA, stored according to the format OpenPGP. Then we specify the basic temporary measures for protection of the private keys in the PGP™ program and proposals for revision of the OpenPGP format. In appendices, we present the technical details of the attacks. The ideas of fault attacks presented in this work and [9] were later on employed in [4] where it has been shown that they apply to the widely used standard PKCS#11 as well. Some of them were also independently verified in [1] and two years later re-discovered and slightly extended in [11].

2. DSA Signature Algorithm

For the purpose of this article we will briefly recap the procedure of creation of the key pair and the signature using 1024bit DSA algorithm (see, for instance, [7], page 452) and the signature verification procedure and introduce the necessary variables.

2.1 Creation of a Key Pair

Let us refer to SHA-1 [7] hash function as h. Each user generates the private and public key together with public parameters as follows:

1. Select 160bit prime number q, such that $2^{159} < q < 2^{160}$.
2. Select 1024bit prime number p, such that $q \mid (p\text{-}1)$ and $2^{1023} < p < 2^{1024}$.
3. Select generator g of the cyclic subgroup of order q in $\mathbf{Z}_p^*$ (that means a random element $\alpha \in \mathbf{Z}_p^*$ will be selected, such that $g = \alpha^{(p-1)/q} \bmod p$ and $g \neq 1$, otherwise another α is selected).
4. Select a random number x, such that $1 \leq x \leq q\text{-}1$.
5. Compute $y = g^x \bmod p$.
6. The public key is y, the public parameters are (p, q, g), the private key is x.

2.2 Creation of a Digital Signature

When creating the signature of the message m (actually its hash value $h(m)$) the user uses her private key x and public parameters (p, q, g) according to the following procedure:

1. Select a random secret number k, $0 < k < q$.
2. Calculate $r = (g^k \bmod p) \bmod q$, check if $r > 0$, otherwise go to (1).
3. Calculate $kInv = k^{-1} \bmod q$.
4. Calculate $s = [kInv * (h(m) + x*r)] \bmod q$, check if $s > 0$, otherwise go to (1).
5. Digital signature of the message m is the pair (r, s).

Let us note that r, s, and q are generally 160bit numbers, whereas p, g, and y are 1024bit numbers.

2.3 Verification of a Digital Signature

For verification of the digital signature (r, s) of the message m, we use signer's public key y together with public parameters (p, q, g) according to the following procedure:

1. Verify that $0 < r, s < q$. In the opposite case the signature is invalid.
2. Calculate $sInv = s^{-1} \bmod q$ and the hash value $h(m)$.
3. Calculate $u_1 = sInv * h(m) \bmod q$, $u_2 = sInv * r \bmod q$.
4. Calculate $v = (g^{u_1} * y^{u_2} \bmod p) \bmod q$.
5. The signature is valid if and only if $v = r$.

3. Description of the *Secret Key Packet* Data Structure for Storage of the Private Signature Key According to OpenPGP

We describe here the data structure (so-called `Tag`) `Secret Key Packet` in which the primary signature key (RSA or DSA) is stored. There are two versions of this format. Version 3 applies only for RSA keys, version 4 may include both DSA and RSA keys. We will describe a side channel attack on the RSA signature key using both format versions and on the DSA signature key using version 4 of the format. The version 4 of this format is used also by the program PGP™ 7.0.3 which prefers DSA to RSA. Specifically, we will therefore work with RSA keys in the version 3 format and with DSA keys in the version 4 in practice. Let us note that the versions 3 and 4 are different in an encryption method of the private data and therefore also the attacks on both algorithms are different. In both versions of the format, the structure of the `Secret Key Packet` contains in the beginning the data from the structure of the `Public Key Packet`, concerning the public key, and then the data concerning the private key. Description and the content of the individual items are illustrated in Table 1 for the DSA algorithm and Table 2 for the RSA algorithm. Regarding the content of the table, let us recall that the MPI format (multi-precision integer) contains prefix and then the actual (big) integer number in the BIG ENDIAN encoding. The prefix forms two octets in BIG ENDIAN and indicates the number of valid *bits* of the subsequent number [3].

Table 1. Content of the Secret Key Packet structure for 1024bit algorithm DSA

Public	1 octet indicating the version number	
	4 octets indicating the time, when the key was created	
	1 octet indicating the algorithm for creation of a digital signature. Then the fields (numbers in MPI format) follow containing public parameters and public key for 1024bit DSA algorithm:	
	prime number p (2 + 128 octets long in figure 1)	

<table>
<tr><td rowspan="3">Key Packet</td><td>prime number q (2 + 20 octets long in figure 1)</td><td rowspan="6">Area of publicly available data</td></tr>
<tr><td>number g (2 + 128 octets long in figure 1)</td></tr>
<tr><td>user’s public key y (2 + 128 octets long in figure 1)</td></tr>
<tr><td colspan="2">1 octet (string-to-key usage), indicating whether and how the private key is encrypted. The value 0xFF is preferred indicating that the following three optional items are completed.</td></tr>
<tr><td colspan="2">[Optional] In case that string-to-key usage is 0xFF, there is 1 octet here, identifying a symmetrical encryption algorithm for protection of the private key.</td></tr>
<tr><td colspan="2">[Optional] If the string-to-key usage is 0xFF, there is a "string-to-key specifier" which says how the password of the user is processed for a symmetrical key. The value 0x03 is preferred indicating the so-called iterated and salted string-to-key identifier. Typically the following data is stored here with the following meaning:
1 octet: 0x03 (iterated and salted string-to-key identifier)
1 octet: identifier of the hash algorithm (for SHA-1 it is 0x02)
8 octets: salt (random data which are hashed together with the user’s passphrase to diversify the symmetrical key derived)
1 octet: the number of hashed octets of the data (the so-called "count").</td></tr>
<tr><td colspan="2">[Optional] If the private key is encrypted, the initialization vector (IV) is stored here. This is a random data in the length of the cipher block (8 octets for 64bit block ciphers, 16 octets for AES algorithm).</td></tr>
<tr><td colspan="2">Algorithmically dependent numbers in the format MPI. For DSA, only the private exponent x is stored here:</td><td rowspan="4">Area of Sensitive Data</td></tr>
<tr><td colspan="2">2 octets, prefix of the number x (encrypted)</td></tr>
<tr><td colspan="2">20 octets, the x value (encrypted)</td></tr>
<tr><td colspan="2">2 octets, checksum as an arithmetic sum of 22 previous octets in plaintext modulo 65536 (encrypted).</td></tr>
</table>

Now, let us stop at files of `secring.skr` type in programs of PGP™ where the PGP™ program saves the `Secret Key Packet` structure. Apart from the `Secret Key Packet` field, this file usually stores several other records such as:

- `UserID Packet` (contains an identifier of the user, i.e. its name and e-mail address),
- `Signature Packet` (contains time of signature, key expiration and so on),
- `Secret Subkey Packet` (contains similar data such as `Secret Key Packet`, but this time about asymmetric key and algorithm for encryption of the data),
- another `Signature Packet` (contains the data such as, for instance, a time of signature of this key by the signature key, amount/size of trust to it and so on).

These other records, however, do not contain any check of integrity of the whole `Secret Key Packet` record. This opens a door for successful attacks on the `Secret Key Packet`.

4. Attack on DSA Signature Algorithm

Let us notice that the integrity of the `Public Key Packet` field is not visibly secured anywhere in the format of OpenPGP, and as it became apparent by carrying out a practical attack, not even in PGP™ programs. Nevertheless, when creating the digital signature it is public parameters of this field that are just utilized (in the event of PGP™ program, the `Secret Key Packet` is stored specifically in `secring.skr` file). These parameters could be read from the record of the public key (the file `pubring.pkr`), but it is logical that if the record of the private key is open, they will be read from here. In the record of `Secret Key Packet` the value of the private signature key is protected, but the mistake is that here the value of public parameters or public key is not protected anyhow. Specifically in the case of DSA, the numbers p, q, g, and y are unprotected of which we will use only p, g for the attack.

4.1 Attack Description

The main idea of the attack on DSA consists in the following steps. The attacker will:

1. prepare special numbers referred to as *PGPrime* and *PGGenerator*,
2. obtain the record of `Secret Key Packet` of the given user and replace p, g values stored in the structure `Public Key Packet` inside the `Secret Key Packet` by the values $p' = PGPrime$ and $g' = PGGenerator$,
3. capture the first not enciphered message or the file which the user signed with such false parameters and will keep its signature,
4. calculate the private key of the user (the x value) on the basis of the message obtained and its signature,
5. return the p, g values to the original `Secret Key Packet`.

The procedure of the attack on the DSA algorithm will be now described in detail and specifically as we carried it out utilizing the PGP™ 7.0.3. program. Examples and the procedure are specified for 1024 bit DSA. In the text below, we will mark the foisted values together with the values computed on the basis of these false values with an apostrophe.

1st step

We selected the constant prime number *PGPrime*, such that

1. *PGPrime* has 159 bits, so the condition $PGPrime < q$ is trivially fulfilled,
2. $PGPrime = t*2^s + 1$, where 2^s is as big as possible and t is a small prime number.

Specifically, we selected $s = 151$ and $t = 167$, i.e. *PGPrime* has the following binary format 10100111000....(150 zeros)....0001. Hexadecimally, it is written as 0x5380 0000 0000 0000 0000 0000 0000 0000 0000 0001.

Then we selected the constant number *PGGenerator*, such that

1. $1 < PGGenerator < PGPrime - 1$,
2. *PGGenerator* is a generator of the multiplicative group $\mathbf{Z}^*_{PGPrime}$.

Specifically, we chose *PGGenerator* = 0x31AC8529 1FF814E6 25E4B88C 8C5047A7 DB2F0E45 and verified that $(PGGenerator)^{(PGPrime-1)/2} \bmod PGPrime \neq 1$ and $(PGGenerator)^{(PGPrime-1)/t} \bmod PGPrime \neq 1$.

2nd step

Now, we took the file `secring.skr` and in its record of the `Secret Key Packet` we changed the values (p, g) to the values ($p' = PGPrime$, $g' = PGGenerator$). We then adjusted lengths of these numbers in the MPI format and cut short the overall length of the `Secret Key Packet` (values in the beginning of the record `secring.skr`) in such a way, that it correspond to shorter false values p' and g'. The situation is illustrated in figures 1 and 2.

3rd step

Foisting such faked values on the user being attacked, we waited till the user signs any file known to us (message m). We then captured its signature - values (r', s'). Now, let us denote k the randomly chosen number, unknown to us, which the user program chose for the signature captured (c.f. the description of DSA in §2 above).

4th step

In this step, we calculated the value of the private key x of the given user on the basis of values p', g', m, r', and s'. From the definition of the signature value (r', s') it holds that

(1) $r' = [(g')^k \bmod p'] \bmod q$, where according to the choice $p' < q$ we get
(1a) $r' = (g')^k \bmod p'$,
(2) $s' = ([k^{-1} \bmod q] * [h(m) + \boldsymbol{x}*r']) \bmod q$, thus
(2a) $\boldsymbol{x} = ([s'* k - h(m)] * [(r')^{-1} \bmod q]) \bmod q$.

The key issue is now that we are able to calculate the unknown randomly chosen number k thanks to the choice of *PGPrime* and *PGGenerator*. The prime number p'= *PGPrime* was selected in such a way, that the equation (1a), i.e. the problem of the discrete logarithm in $\boldsymbol{Z}_{p'}^*$, is easy to solve. The specific procedure of calculation of the discrete logarithm in this special group is specified separately in Appendix 1. On the basis of this procedure, we calculated the value of k from the equation (1a) and then computed the value of x from the equation (2a).

We checked the correctness of x according to the relationship $y = g^x \bmod p$ with original values y, g, p. The private key value is therefore calculated and its validity is verified against the value of the public key.

5th step

To the user, we returned her original file `secring.skr`.

4.2 Practical Implementation of the Attack

The attack was applied to the PGP™ program v. 7.0.3 for Windows 95/98/NT/2000. From the nature of the attack and the data formats used, it follows that this attack should be also successful on other platforms. The procedure of the attack was carried out precisely according to the aforesaid description. The adjustment of parameters was carried out in the field `Secret Key Packet` in the `secring.skr` file,

where the private signature key is stored as we can see in figures 1 and 2. On the first figure, original values are indicated and the second one shows false values.

```
  95**01D7**043A8D29DF11**0400**F2E02A396A14E137085DA859B3569AF4027EA37
9682F46780920B72127C88787DDC1BFF9FDB59E564B741FD5FC98856679F1C04
1CB71895CB6975E7FE6E15A6D4B70514560E11A25637F3FBA35E89E5F1FA272A
2707F4865EA106EE402973D4969A276DA4911005B968B81548621CEBBB5771A3
5C5A785F7F480E47277D2BAB500A0FF04303152BD2A9AD963E063A3FE34A8A55
34F3F**0400**CD8580F20AA821A6D2FF5255DFD02E4F4C8D8DA3731517476BEE096
F7B104B01B6CE1C4DE586BAEA30D82B50DCB3F0D20B0F0D07D8384C09F12CBF0
79887CEB696E822D753A48584F2BC573C84E8490AB310FDBCC40EAEBCD05973B
3F2A1A479FFE0E4B63026E066B6E936F1B2B7F1C91C65CBA0F27B4C0D22254BB
C852DEDE10400AC756BB6EB82313A0FE91F47A36D1425D89FB124CD0ACBA082E
8B2C2B048BE92C5CE7A5FAA5AF317DCC086150B98AB504C0DA6BF1D87FAB73C8
F8D0FC821BD8902CA6927338CF0D682E7C9E3E8D89A3D00D53224C301E6C932A
DA7562FA15E9027E105F803043D4CBC08807A8FB71FEF9B27EE6A0722C4BF601
D032CC59F6FE4FF09030213CF38106B7BCA3F603F59437C3860B98DA3A1A3F02
A4D2754075B494CAC156E38E1282705FB0BBD68940A1653457E161AB00187B42
8566C617374696D696C204B6C696D612C20445353203C762E6B6C696D6140646
563726F732E637A3EB00303FFFF89005A04101102001A05023A8D29DF050901E
13380040B090807021901051B03000000000A09109B89D5F084A0EAD7A35C009
F5B643D5D2C37F4B2CFC9F399873B747CB3FBFE6800A0A8FE0A2498E332586F6
BED3BB88F278B0C5CF079B001679D014E043A8D2A78100400C904D0246F86223
52D6A60F67F1B4AAA4E94562BB00595A67DDB853AEF3F421CEE2D5FCD5AF1818
0872FE5029600938159010460967927 4CC92770C6DEBCF391A39B92270D71E7C
5EBEC66B3A3BF1BDF5217E9F609F5D011B9D648A930998C61CD462F3BAAAFD91
6FDBFFFAA01FCBD2E42F1BC5C406BB0763B3D4830240841390002020 3FE2B39B
802893DE670D745D2AE4DF802BDA707E829B7B0FF7438FFB88EAB76189AA90A1
43FB11C1DCC5149046C913AB114D9775563BA0103E65C951D6AC9199D52818BD
2B8A8BF07A6E9F8C242811FF9522EE168207F1EC5D49B441C63D473F7C83D89C
3B6F43A3D80B1B38F7195DE45A55807207159A70CA883493532CD4D8802FF090
30221D3B8EC1276C0D3601B745E982D01201DA87DB47FD3B9C8CDD8F6BC857F5
6B6F4370AB8A94C7A4E528D209A80B365A416AF80E1198BAC1AE4175A0F90B00
1878900520418110200120 5023A8D2A78050901E13380051B0C000000000A091
09B89D5F084A0EAD7CD04009F7056BA18F5907E36E4ED9A79B4160AE8C6338D9
8009B043443B6665E860719768B49382DEC95FD7F96BBB00167
```

Fig. 1. File `secring.skr` hexadecimally, original values *p*, *g* are indicated with their lengths together with the total length of the record

```
  95**00FF**043A8D29DF11**009F**53800000000000000000000000000000000000000
100A0FF04303152BD2A9AD963E063A3FE34A8A5534F3F**009E**31AC85291FF814E
625E4B88C8C5047A7DB2F0E450400AC756BB6EB82313A0FE91F47A36D1425D89
FB124CD0ACBA082E8B2C2B048BE92C5CE7A5FAA5AF317DCC086150B98AB504C0
DA6BF1D87FAB73C8F8D0FC821BD8902CA6927338CF0D682E7C9E3E8D89A3D00D
53224C301E6C932ADA7562FA15E9027E105F803043D4CBC08807A8FB71FEF9B2
7EE6A0722C4BF601D032CC59F6FE4FF09030213CF38106B7BCA3F603F59437C3
860B98DA3A1A3F02A4D2754075B494CAC156E38E1282705FB0BBD68940A16534
57E161AB00187B428566C617374696D696C204B6C696D612C20445353203C762
E6B6C696D6140646563726F732E637A3EB00303FFFF89005A04101102001A050
23A8D29DF050901E13380040B090807021901051B03000000000A09109B89D5F
084A0EAD7A35C009F5B643D5D2C37F4B2CFC9F399873B747CB3FBFE6800A0A8F
E0A2498E332586F6BED3BB88F278B0C5CF079B001679D014E043A8D2A7810040
```

```
0C904D0246F8622352D6A60F67F1B4AAA4E94562BB00595A67DDB853AEF3F421
CEE2D5FCD5AF18180872FE5029600938159010460967927 4CC92770C6DEBCF39
1A39B92270D71E7C5EBEC66B3A3BF1BDF5217E9F609F5D011B9D648A930998C6
1CD462F3BAAAFD916FDBFFFAA01FCBD2E42F1BC5C406BB0763B3D48302408413
900020203FE2B39B802893DE670D745D2AE4DF802BDA707E829B7B0FF7438FFB
88EAB76189AA90A143FB11C1DCC5149046C913AB114D9775563BA0103E65C951
D6AC9199D52818BD2B8A8BF07A6E9F8C242811FF9522EE168207F1EC5D49B441
C63D473F7C83D89C3B6F43A3D80B1B38F7195DE45A55807207159A70CA883493
532CD4D8802FF09030221D3B8EC1276C0D3601B745E982D01201DA87DB47FD3B
9C8CDD8F6BC857F56B6F4370AB8A94C7A4E528D209A80B365A416AF80E1198BA
C1AE4175A0F90B0018789005204181102001205023A8D2A78050901E13380051
B0C000000000A09109B89D5F084A0EAD7CD04009F7056BA18F5907E36E4ED9A7
9B4160AE8C6338D98009B043443B6665E860719768B49382DEC95FD7F96BBB00
167
```

Fig. 2. File `secring.skr` after intrusion, modified values *p´*, *g´* are indicated with their lengths together with the total length of the record

5. Attack on RSA Signature Algorithm in OpenPGP

5.1 Brief Description of RSA Signature Transformation

Here, we briefly recall the definition of RSA signature algorithm (see for instance [8]) and introduce common labeling of its variables. Let us refer to RSA modulus as n and let $n = p*q$, where p, q are prime numbers. Let us denote public exponent e and private exponent d. Then let us define $pInv = p^{-1} \bmod q$. The pair (n, e) will be referred to as a public key. For the needs of OpenPGP format, a private key is considered to be represented by the quadruple (d, p, q, $pInv$). Signature of the message m is created as the value $s = m^d \bmod n$, where it is assumed that the message m has been already formatted in advance in a certain manner. The signature s of the formatted message m is valid if and only if $m = s^e \bmod n$.

5.2 Description of Attack on the RSA Signature Key

We anticipate that the following thoughts can be applied to a signature key of RSA algorithm implemented according to the general OpenPGP format. However, the PGP™ programs have extended built-in mechanisms for integrity check of this key prior to its use for a signature, which successfully thwarts the attacks. In this case, we therefore attack only on the OpenPGP format.

As with DSA, the private key is stored in the `Secret Key Packet` structure. At the time of writing this analysis, versions 3 and 4 of this format are used which differ in how the private data is encrypted. Both versions of `Secret Key Packet` contain in the beginning the data on the public key (in the `Public Key Packet`

structure) and then the data of the private key follows. The structure `Public Key Packet` has also two versions of the format (3 and 4), but these differ only by the `time` field of data (c.f. Table 2). Therefore, let us describe only the content of the newer version 4 (see Table 2).

The `Public Key Packet` structure contains:

1. 1 octet version number
2. 4 octet number indicating time, when the key was created
3. 1 octet identifying asymmetric algorithm (RSA here) belonging to this key
4. A sequence of MPI integers containing a public key. Here it is :
   - number *n* (RSA modulus) in MPI format,
   - number *e* (RSA public exponent) in MPI format.

After the `Public Key Packet`, other data of `Secret Key Packet` follows, while the data of the private key is already encrypted as Table 2 shows. In the case of RSA, the following data is concerned:

- private exponent *d*,
- prime number *p*,
- prime number *q*, satisfying $p < q$,
- *pInv* ($= p^{-1} \bmod q$),
- two octets of the *checksum*.

Table 2. Content of the structure `Secret Key Packet` version 3 and 4 for RSA algorithm (modulus 1024 bits)

<table>
<tr><td rowspan="6">Public Key Packet</td><td>1 octet indicating the version number</td><td rowspan="10">Area of Publicly Available Data</td><td></td></tr>
<tr><td>4 octets indicating the time when the key was created</td><td></td><td></td></tr>
<tr><td>(in version 3 of <code>Public Key Packet</code>, 2 octets more indicating the number of days of the key validity)</td><td></td><td></td></tr>
<tr><td>1 octet indicating RSA algorithm</td><td></td><td></td></tr>
<tr><td>n (RSA modulus) in MPI format</td><td></td><td></td></tr>
<tr><td>e (public RSA exponent) in MPI format</td><td></td><td></td></tr>
<tr><td colspan="2">1 octet (string-to-key usage)</td><td></td><td></td></tr>
<tr><td colspan="2">[Optional] If the string-to-key usage is 0xFF, there is 1 octet here identifying symmetric encryption algorithm for the private key protection (c.f. Table 1).</td><td></td><td></td></tr>
<tr><td colspan="2">[Optional] If the string-to-key usage is 0xFF, there is "string-to-key specifier" (c.f. Table 1).</td><td></td><td></td></tr>
<tr><td colspan="2">[Optional] If the private key is encrypted, initialization vector (IV) is saved here. This is a random data in the length of the block of the used block cipher (8 octets for 64-bit block ciphers, 16 octets for AES algorithm).</td><td></td><td></td></tr>
<tr><td colspan="2">Algorithmically dependent numbers in MPI format. The content for RSA (modulus 1024 bits) is:</td><td>version 3</td><td>version 4</td></tr>
</table>

2 octets, prefix of the number *d*	plaintext	encrypted
128 octets, the value of *d*	encrypted	
2 octets, prefix of the number *p*	plaintext	
64 octets, the value of *p*	encrypted	
2 octets, prefix of the number *q*	plaintext	
64 octets, the value of *q*	encrypted	
2 octets, prefix of the number *pInv*	plaintext	
64 octets, the value of *pInv*	encrypted	
2 octets (H_{Sum} , L_{Sum}) of *checksum*, a sum of previous items (MPI numbers) in open form (mod 65536)	plaintext	

Table 2 specifies how the private data encryption differs in versions 3 and 4 of the `Secret Key Packet`. Both formats will be observed separately. A common element of both formats is a calculation of the checksum as a simple arithmetic sum of the individual bytes of the private data modulo 65536: *checksum* = ($d_1 + d_2 + ...+ d_r$) mod 65536. In both versions the user uses for encryption the chosen symmetric block cipher in the so-called specific (PGP™) mode CFB.

Special feature of the version 3 of `Secret key Packet` format is that the prefixes of MPI numbers forming the private key and the checksum are not enciphered. Moreover, in the beginning of every MPI, the state of the CFB is resynchronized in such a way, that the new block starts precisely from the new MPI value. In the `Secret Key Packet` format – version 4, all private MPIs including prefixes and *checksum* are encrypted without the resynchronization.

5.2.1 Attack on Version 3 of Secret Key Packet (RSA) Format

The attack which may be launched on version 3 of the format consists in the possibility of changing the length of the individual MPIs, i.e. of the MPI prefixes, as they are not encrypted and neither is the *checksum*. For instance, it is possible to decrease the prefix of the MPI number *pInv* by one together with reducing also the *checksum* by one at the same time. The validity of the checksum of the record `Secret Key Packet` (with new *pInv´*) will be preserved (all octets of *pInv* are summed up, regardless of how much bits are valid in them), but the value of the private information *pInv* is changed, as this number is cut short by 1 bit. Likewise, the values of other private numbers *d*, *p*, *q* can be changed. As soon as the attacker gets the message and its signature, which will be created using a private key (*d*, *p*, *q*, *pInv´*), she will be capable of computing the whole RSA private key. A detailed procedure is specified in Appendix 2. In this sense, the situation is actually analogical to the attack on DSA (c.f. §4).

5.2.2 Attack on the Version 4 of Secret Key Packet (RSA) Format

When attacking on the version 4 of the `Secret Key Packet` format, it is not possible to utilize the previous procedure directly as the prefixes as well as the *checksum* are encrypted. However, these values can be still modified. The modification consists in using the CFB properties in encryption of the last block of plaintext. For instance if we use a block cipher with the block length of 16 octets (e.g.

in the case of AES) together with an 1024 bits RSA modulus, the last incomplete block of the ciphertext will contain eight last octets of the *pInv* number (let us denote them B_1, B_2,..., B_8) and two octets of the *checksum* (let us denote them H_{Sum}, L_{Sum}). This plaintext will be encrypted by a simple XOR operation with the key material (encrypted image of the preceding ciphertext). If we carry out the change of "XOR CONST" type in the last block of the ciphertext, it will appear precisely as "XOR CONST" after decryption in the plain text, since the XOR operation is both commutative and associative. The objective is to carry out the change in *pInv* (to be more accurate, in the last eight octets of *pInv* number) and simultaneously in the *checksum* (in L_{Sum} octet) for *pInv´* and *checksum´*, such that the integrity check agrees after deciphering. The result will again be a disrupted value *pInv*, while the `Secret Key Packet` format will have a correct *checksum*. The use of this fake *pInv´* is the same as in the previous case. We will obtain a signature of some message with this false key and from here we will compute the value of the RSA private key.

Now, we will show how it is possible to change the individual bits of B_1 to B_8 octets and L_{Sum} in such a way, that the check of integrity of the private key agrees after their modification. The changes of the individual bits of the specified octets will be carried out on the ciphertext and as we have already stated, with regards to the CFB mode, these changes will project in the same way into the corresponding bits of octets of the plaintext. Now, let us anticipate that in the open format of some octet B_i from the set of octets $\{B_1, B_2, ..., B_8\}$ some *j*-th bit (where $j \in <0, 7>$) *equals* to the *j*-th bit in L_{Sum} octet. Then it only suffices to change this *j*-th bit in the enciphered octet B_i and simultaneously in the enciphered octet L_{Sum} and the check of integrity will agree. If this bit was 1, B_i octet will change to $B_i - 2^j$ octet and L_{Sum} octet will change to $L_{\text{Sum}} - 2^j$. The new checksum will thus be valid. Likewise, if *j*-th bit of B_i and L_{Sum} octets was 0, the octet B_i will change to $B_i + 2^j$ and L_{Sum} octet to $L_{\text{Sum}} + 2^j$. The new checksum will be valid again.

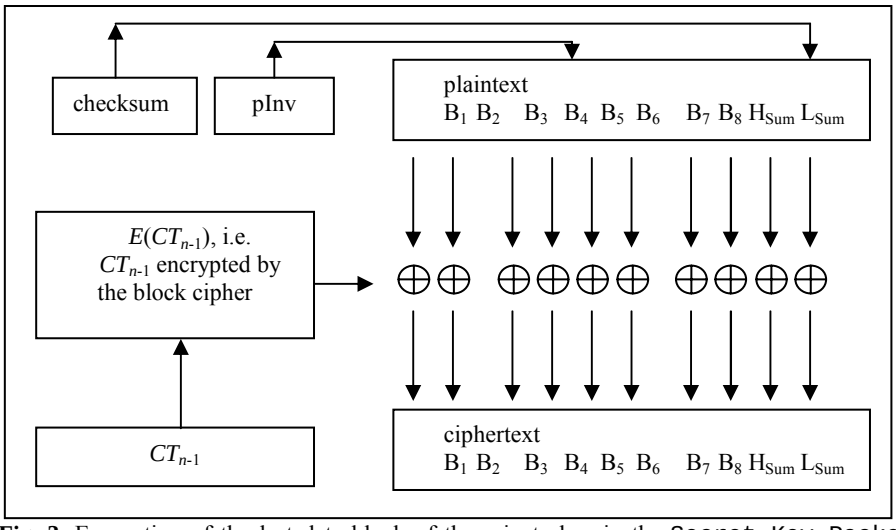

Fig. 3. Encryption of the last data block of the private key in the `Secret Key Packet`, containing the private value *pInv*

As we do not know whether j-th bit of the selected octet B_i is the same as j-th bit of L_{Sum}, we have to examine several possible combinations. We can change a total of 64 bits and the probability of not succeeding is very low (of order 2^{-64}). Let us note that we will learn of the success of the specified change only after the user tries to sign a message with this false key. Nevertheless, on average, two attempts should suffice (one attempt succeeds with probability circa 2^{-1}). Another method is to change the j-th bit simultaneously always in two octets of any choice from the set $\{B_1, B_2, ..., B_8\}$, whereas L_{Sum} will be left unchanged. This time we search for a situation, when these bits are different in the plaintext. Their simultaneous change will annul their influence in the *checksum*. Likewise, we can carry out variations with four or eight j-th bits (e.g. affecting B_a, B_b, B_c, and B_d simultaneously for different index values $a, b, c, d \in <1, 8>$). The result of this change is a disruption of *pInv* with the same consequences as in the previous cases, i.e. getting a value of the RSA signature private key.

6. Attack on the Private Keys after their Export

It is necessary to note that, apart from the private keyring `secring.skr`, it is possible to use the same method to attack also on a private key which is exported to the file of the type "ASCII Key File" and then transferred through the network or on a removable media (e.g. floppy disk, CD, etc.). This file has, apart from an additional encoding, the same content as the `secring.skr` file and therefore the same attack can be applied on it as on the `secring.skr` file. This means that the transfer of the private key through this file over the network or on a removable media is highly dangerous operation and shall be avoided, until the vulnerabilities discussed here are fixed.

7. Countermeasures

7.1 Basic Temporary Countermeasures

The main cause of the just presented attacks is insufficient control of integrity of the public as well as private data in the file containing the private key of the user. As a logical countermeasure a necessity results from this condition to introduce a better control of integrity of the saved records. We emphasize that this control must secure also the integrity of the public values which may, however, not be necessarily enciphered.

The requirement for introduction of a robust integrity check may not be easy to implement in a short-time horizon. Until the adjustment of the OpenPGP [3] format occurs for records of the private keys (`Secret Key Packet`), it is possible to use temporarily at least the following control tests in the PGP™ programs and others which implement the OpenPGP format. These are proposed in such a way, that the keys for the DSA and RSA algorithms which fulfill the below-mentioned relations are

not vulnerable to aforesaid attacks. It is presumed, that this test will be carried out as an additional check of integrity after reading the respective parameters from the file with the private key. For the actual signing operation only such a key may be used the values of which pass this test. We stress out that the aforesaid test is not to be a substitute for the missing check of integrity of the file with the private key but it is to serve as a temporary measure which prevents the herein specified attacks.

7.2 Temporary Test for DSA

We propose the following temporary test for DSA. The following relations should be verified:

1. $p, q, g, x, y > 0$
2. p is odd, q is odd
3. $2^{159} < q < 2^{160}$
4. $1 < g < p$
5. $1 < y < p$
6. $x < q$
7. $q \mid (p\text{-}1)$
8. $g^q \bmod p = 1$
9. $g^x \bmod p = y$

It can be easily verified that the tests effectively detect the attack described in §4. However, let us also note that whereas such type of tests is usually trusted a lot, for instance in RSA, in case of DSA we have to be careful. Unlike RSA there is only one value here (private key x), unknown to the attacker. Other parameters are known by the attacker and she can change them at her own discretion. This is a reason, why this test is considered only a temporary solution, which must be replaced as soon as possible with another type of check of integrity of the discussed records, as hereafter specified.

7.3 Temporary Test for RSA

We propose this temporary test for RSA. The following relations should be verified:

1. $e*d \bmod (p - 1) = 1$
2. $e*d \bmod (q - 1) = 1$
3. $pInv*p \bmod q = 1$
4. n (from the record of the public key) $= p*q$
5. $e \in E$, where E is a set of possible values planned for e, i.e. for PGP™ for instance {17, 65537, ...}

Let us note that in the program PGP™, the tests 1 to 4 together with other checks are already implemented. However, in the OpenPGP format, these controls are not anticipated, which makes applications written strictly according to the OpenPGP description vulnerable to the attack presented in §5.

7.4 Other Topics for the OpenPGP Format

Here we state some other topics which would essentially contribute to increasing safety of the OpenPGP format as well as PGP™ program. However, it is necessary to perceive them as recommended ideas only. Before specific adjustments are implemented, they should undergo at least a basic independent analysis. The analysis of the whole OpenPGP format is however much more complex and reaches beyond the frame of this paper. The proposed measures are:

1. Enciphering modus CFB should be replaced with a CBC modus
   - will make the described attack on the last block of enciphered private data more difficult
2. Replace the *checksum* (sum of bytes mod 65536) with HMAC based on SHA-1 or on another safe hash function (for instance SHA-256, 384, 512, and so on)
   - will make the attacks on protected data and simultaneously on the integrity check code more difficult
3. The new checksum (HMAC-based) should be:
   a) saved in the length of at least 160 bits,
      - will make the attacks using the birthday paradox adequately difficult
   b) computed from all data of the record `Secret Key Packet` (not only from the private but also from the public values),
      - will make the integrity attacks on public as well as private parts of keys in `Secret Key Packet` more difficult
   c) using the key derived from passphrase in another manner than the key for symmetric cipher
      - will make the attack on HMAC more difficult
   d) encrypted together with the private data by a symmetric cipher similarly as it is in case of checksum in Version 4 of the `Secret Key Packet` format
      - will make the integrity attacks on public as well as private parts of the key in `Secret Key Packet` more difficult
   e) operated in such a way that side channel attacks on the CBC modus itself (c.f. parts D and E in this thesis) are eliminated
4. Use the format of EMSA-PSS type specified in [8] for the RSA signature scheme
   - will make a number of attacks including attack described in Appendix 2 more difficult

8. Impacts

The types of attacks demonstrated show a considerable impact to security of the programs based on the format OpenPGP (e.g. PGP™ program itself). Anybody, who

can change a file with private key, can get the private key value in algorithms DSA and RSA based on a single incorrect signature. Moreover, this change need not occur only in the workstation of attacked user at all. Sensitive point of the system can be also seen in the files with exported private keys, which are used by the user for a transfer of her private keys between various stations. The fact that private key is stored in an encrypted form may provide the user with some feeling of security, which is false, however. If the attacker gets at e.g. such a removable disk during its transport, a security of the user's private key is endangered considerably.

Another scenario which is very efficient in case of the attack described above can be used in situation, when the file with private key is stored on a shared device. In such a case, the attacker can be e.g. server administrator who foists a modified version of this file upon the user for certain time period. Further on, she waits until the user uses it for signing (time period can be determined by monitoring of user's station network activity with relatively high accuracy), and finally she returns the original content of the file. She can identify the private key value for the signature that has been generated. When having sufficient control over the entire system, the attacker can even cover the tracks of the attack, when she sends a message with valid signature instead the original message furnished with the false signature. This can be done easily, because at this moment the attacker knows both the message itself and the proper private key.

Users of the programs based on OpenPGP have to face to a very difficult situation, when they find out that a faulty signature value was generated. Definitely, they may wonder, whether they are facing to an impact of intentional attack, or it is "only" some technical failure. Of course, it is clear that proper care has to be paid to every file with false signature, as if it was a file containing private key in an open form! Subsequent treatment includes, first of all, irreversible wiping of the file from the station or even the server in question.

9. Conclusion

The above mentioned attacks leading to a disclosure of the most sensitive system information (private signing keys of RSA and DSA algorithms), show clearly the importance of protection of private keys and public parameters of asymmetric algorithms in security systems. Let us mention also the fact that it was right the research study aimed to general problems and principles of asymmetric keys protection, in the framework of which we saw "how the PGP™ program makes it", without a primary intention to target to it.

Our analysis was based on general documentation of OpenPGP [3]. We have revealed serious insufficiencies in it, which can lead to easy vulnerability of the applications based on it. As a practical example, we can mention the PGP™ program, which shows resistance to attacks to RSA thanks to sufficient protection beyond the scope of the OpenPGP, but which is easily vulnerable by the attack to DSA signature algorithm.

Note that although we limited our study to RSA and DSA algorithms in relation to OpenPGP, similar vulnerabilities can be expected also in other asymmetric

cryptosystems, including the systems based on elliptic curves, if sufficient protection of private keys and public parameters is not ensured (e.g. we can substitute a weak elliptic curve similarly to the substitution of the weak *PGPrime* in §2, etc.). And the OpenPGP format as well as the PGP™ program are probably not the only cases, when attacking a given system is enabled by improper protection of the respective parameters. As a whole, this document calls for attention, which must be paid to designing protections of asymmetric cryptosystems keys and parameters and to storing them within a given system.

References

1. Aumüller, C., Bier, P., Fischer, W., Hofreiter, P., and Seifert, J.-P.: *Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures*, in Proc. of CHES 2002, pp. 260-275, 2002
2. Boneh, D., DeMillo, R.-A., and Lipton, R.-J.: *On the Importance of Checking Cryptographic Protocols for Faults*, in Proc. of EUROCRYPT '97, pp. 37-51, 1997
3. Callas, J., Donnerhacke, L., Finney, H., and Thayer, R.: *RFC 2440: OpenPGP Message Format*, November 1998
4. Clulow, J.: *On the Security of PKCS #11*, In proc. of CHES 2003, pp. 411-425, 2003
5. Joye, M., Lenstra, A.-K., and Quisquater, J.-J.: *Chinese remaindering cryptosystems in the presence of faults*, Journal of Cryptology, Volume 12, Number 4, pp. 241-245, Autumn 1999
6. Lenstra, A.-K.: *Memo on RSA signature generation in the presence of faults*, manuscript, Sept. 28, 1996
7. Menezes, A.-J., van Oorschot, P.-C., and Vanstone, S.-A.: *Handbook of Applied Cryptography*, CRC Press, 1996, online at http://www.cacr.math.uwaterloo.ca/hac/
8. PKCS#1 v2.1: *RSA Cryptography Standard*, RSA Laboratories, DRAFT2 – January 5 2001, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/
9. Rosa, T.: *Future Cryptography: Standards are not Enough*, in Proc. of Security and Protection of Information 2001, Military Academy in Brno, pp. 237-245, NATO-IDET, Brno, 9. - 11. of May, 2001
10. Rosen, K.-H., Michels, J.-G., Gross, J.-L., Grossman, J.-W., and Shier, D.-R.: *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, 2000
11. Yen, S.-H., Moon, S.-J., Ha, J.-C.: *Permanent Fault Attack on the Parameters of RSA with CRT*, In Proc. of ACISP 2003, pp. 285-296, 2003

Appendix 1: Determination of a Private Key Value by Changing the DSA Public Parameters

Assumption P1 (Method of DSA signature calculation). *Let us assume the DSA parameters (p, q, g, y, x), where p, q are prime numbers, $g \in \boldsymbol{Z}_p^*$, ord(g) = q, $y = g^x \bmod p$, $0 < x < q$, x is signer´s private key. Signature for the message with hash code h(m) is then calculated as follows:*

1. *select random k, $0 < k < q$*
2. *calculate $r = (g^k \bmod p) \bmod q$*
3. *calculate $s = k^{-1}(h(m) + xr) \bmod q$, where $k*k^{-1} \equiv 1 \pmod q$*
4. *the signature is the pair (r, s)*

The attack described below assumes that the attacker changes the DSA parameters (p, q, g, y, x) for (p', q, g', y, x), where p' is 159 bit prime number, $2^{158} < p' < 2^{159}$ and g' is a generator of group $\boldsymbol{Z}_{p'}^*$ such that for any $r \in \boldsymbol{Z}_{p'}^*$, we can easily determine the integer value w, $w < p - 1$, satisfying $(g')^w \equiv r \pmod p$, i.e. $w = \log_{(g')} r$. So, we can solve a problem of discrete logarithm in $\boldsymbol{Z}_{p'}^*$ easily.

In the following two steps, we show that the value of private key x can be easily determined from the known hash code $h(m)$ and signature (r, s) which was acquired by algorithm DSA with spurious parameters (p', q, g', y, x).

Step 1: Determination of the set *K*

In this step, we are working with the value of r. It follows from equation P1.2, that $r = [(g')^k \bmod p'] \bmod q = (g')^k \bmod p'$, since $p' < q$. According to the above given assumption, the attacker is capable to determine the value of $w = \log_{(g')} r$ easily for any r. For the unknown value of k we have $k = w + b(p' - 1)$, where $b \in \boldsymbol{Z}$, $b \geq 0$. The condition from P1.1 shall be added, assuming that $0 < k < q$. As p' is 159 bits long and q is 160 bits long, so we obtain a set of the values admissible for k, i.e. $\boldsymbol{K} = \{w + b(p' - 1): b(p' - 1) < q - w, 0 \leq b \leq 3\}$. This way, we get a set of at most four possible k_i values, and the value k under search is surely among them.

Step 2: Determination of the value *x*

Now, we will test $k_i \in \boldsymbol{K}$ one after another and compute the values of x_i from P1.3 as $x_i = r^{-1}(k_i*s - h(m)) \bmod q$, where $r*r^{-1} \equiv 1 \pmod q$. Note that $\gcd(r, q) = 1$, so the value r^{-1} exists and is unique. This way we obtain a set of the values $\boldsymbol{X} = \{x_i: k_i \in \boldsymbol{K}\}$, which must include the private key value x being searched for.

All we have to do now is to choose the required value x from the set $\boldsymbol{X}$. It can be done easily with the help of the relation $y = (g^{x_i} \bmod p)$, where p and g are the original public parameters of DSA and y is the public key. By testing four different values (as

the maximum) of x_i, we can eliminate remaining uncertainty introduced by a low value of p' and obtain the value x under search. Note that the element g has the order q within the group $\boldsymbol{Z}_p^*$. As a result, we can say that only one value of $0 < x_i < q$ exists, for which it holds that $y = (g^{x_i} \bmod p)$. That is why the x value determined by this procedure is unique.

Note. *The main principle of the entire attack is obvious from the above given description. It is based on such modification of DSA public parameters, thanks to which very weak instances of the problem of discrete logarithm occur in signature calculations. Instead of solving this problem in multiplicative cyclic subgroup of the group $\boldsymbol{Z}_p^*$ having the 160-bit prime order q, this problem can be solved sufficiently in the cyclic group $\boldsymbol{Z}_{p'}^*$, where p' is a 159-bit prime number with a suitably chosen structure.*

The complexity of such instances of the discrete logarithm problem is considered as absolutely insufficient from the cryptological point of view. Moreover, for a practical implementation of the above described attack, a generally applicable group $\boldsymbol{Z}_{p'}^*$ has been found. This group has a special structure, which enables us to solve the instances generated in the above given problem very quickly, even on a PC of common (at the time of writing the original paper) office type - see the description of algorithm A1 given below.

Algorithm A1: Calculation of w, $w = \log_g r$, for special type of $\boldsymbol{Z}_p^*$

In the following text, we describe an efficient algorithm for calculation of a value of discrete logarithm, which can be used for multiplicative group $\boldsymbol{Z}_p^*$ that have a certain special structure (here p equals to the spurious value of p' mentioned above). It is assumed that a certain carefully selected group with that structure will be generally used for all practical implementations of the above described type of attack against DSA. A structure of the group will be given in a form of the following proposition:

Proposition P2. Let us have *a multiplicative group $\boldsymbol{Z}_p^*$, where p is a prime number with a structure of $p = t*2^s + 1$, where t is a prime number. Further on, let g be a generator of $\boldsymbol{Z}_p^*$. In the following, we show a procedure of calculation of the w value, which is efficient for low t (for practical implementation of the foregoing attack, the generally applicable group with the parameters of **t = 167, s = 151** was found).*

Before we start describing single steps of the algorithm itself, we should recall some useful elementary formalisms to be used later on for explaining particular operations.

Definition D1. [10, page 277] *If p and k are positive integers and b is an integer relatively prime to p, then b is k-th power residue of p if the congruence $x^k \equiv b \pmod p$ has a solution. (In case of k=2 we often use an expression of quadratic residue modulo p.)*

Theorem T1. [10, page 279] *Let p be a prime number, k an integer positive number, and b an integer number fulfilling the condition of gcd(b, p) = 1. Then b is k-th power residue modulo p, iff* $b^{(p-1)/d} \equiv 1 \pmod{p}$*, where d = gcd(k, p-1).*

Lemma L1. *Let p be a prime number and g the generator of the group* Z_p^**. Then the value of y,* $y = g^w \bmod p$*, is k-th power residue modulo p, where k|(p-1), iff k|w.*

Proof. If y is k-th power residue modulo p and $k|(p-1)$, then from the theorem T1 we have $y^{(p-1)/k} \equiv 1 \pmod{p}$. From the assumption $y = g^w \bmod p$, we get $(g^w)^{(p-1)/k} \equiv 1 \pmod{p}$. As g is the generator of $\boldsymbol{Z}_p^*$, it holds that $w*(p-1)/k \equiv 0 \pmod{(p-1)}$. From this expression, we can directly determine that $k|w$.
Verification of implication in the reverse order is easy: Let $k|w$, i.e. $w = k*c$, where c is a positive integer. Then it holds that $y = g^w \bmod p = (g^c)^k \bmod p$ and it follows directly from the definition D1 that y is k-th power residue modulo p.

■

In the following text, we describe three steps of an algorithm for a calculation of the discrete logarithm $w = \log_g r$ under search. It is based on a modified version of Pohling-Hellman algorithm (see [7]), which would be also very efficient in the given type of multiplicative group. In our effort for the most specific structure of the group employed, we derived the following procedure.

Step 1: Determination of the value $^s w = w \bmod 2^s$

Let us assume that $w = w_n*2^{n-1} + w_{n-1}*2^{n-2} + ...+ w_1$, where n is a number of bits of binary expansion of w and $w_i \in \{0, 1\}$, for $1 \le i \le n$. Let us concentrate our attention to determining of the bit w_1. If this bit is zero, it holds that $w = 2*b$, for some integer b. For the value $r = g^w \bmod p$, we can derive that $r \equiv (g^b)^2 \pmod{p}$, so that r is a quadratic residue modulo p. On the other hand, if the value of bit w_1 is one, then the value of w is odd, and r is not a quadratic residue modulo p according to the lemma L1. Based on this analysis and the theorem T1, we can set the following conditions for w_1:

- $[r^{(p-1)/2} \bmod p] = 1 \Rightarrow w_1 = 0$
- $[r^{(p-1)/2} \bmod p] \neq 1 \Rightarrow w_1 = 1$

Let us continue with determination of w_2 and recall that in our setup $4|(p-1)$. First, we adjust r for $r_2 = (r*g^{p-1-w_1}) \bmod p$ based on the known w_1. By this adjustment, we obtain the value $r_2 = g^{w'} \bmod p$, where $w' = w_n*2^{n-1} + w_{n-1}*2^{n-2} + ...+ w_2*2$. Now, if it holds that $w_2 = 0$, we obtain a equation for the value r_2, saying that $r_2 \equiv (g^b)^4 \pmod{p}$, where b is an integer. This means that the value r_2 is a *4-th power residue modulo p* in this case. If it holds that $w_2 = 1$, then w' is not divisible by four and the value r_2 is not a *4-th power residue modulo p* according to the lemma L1. By this, we can derive the following conditions for w_2:

- $[r_2^{(p-1)/4} \bmod p] = 1 \Rightarrow w_2 = 0$
- $[r_2^{(p-1)/4} \bmod p] \neq 1 \Rightarrow w_2 = 1$

After determining w_2, we adjust r_2 again for r_3 as $r_3 = (r_2 * g^{p-1-2*w_2} \bmod p)$ and we continue in determining the values of w_i, such that $2^i \mid (p - 1)$. As $(p - 1)/2^s = t$, where t is an odd prime number, we can determine the values of w_i for $1 \le i \le s$. By this, we obtain the binary notation of the value ${}^s w = w_s 2^{s-1} + w_{s-1} 2^{s-2} + ... + w_1$, where ${}^s w = w \bmod 2^s$. This is the value, we wished to identify in this step. The entire procedure is illustrated in Fig. 4.

Procedure of calculation of ${}^s w = \log_g r \bmod 2^s$

1. Let us assume that:
   a. the binary notation of ${}^s w$ is ${}^s w = w_s w_{s-1} ... w_1$
   b. p is a prime number, $p = t*2^s + 1$
2. set $i = 1$; $f = g^{p-2} \bmod p$; $v = p - 1$
3. set $v = v/2$; $y = r^v \bmod p$
4. **if** $(y = 1)$ **then** set $w_i = 0$ **else** set $w_i = 1$; $r = r*f \bmod p$
5. set $f = f^2 \bmod p$
6. $i = i + 1$
7. **if** $(i \le s)$ **then goto** 3
8. **return** ${}^s w = w_s w_{s-1} ... w_1$

Fig. 4. Step 1 of the algorithm A1

Step 2: Determination of the value ${}^t w = w \bmod t$

It is easy to demonstrate that for the integer j fulfilling the condition of

$$r^{(p-1)/t} \equiv (g^{(p-1)/t})^j \pmod p$$

it holds that ${}^t w \equiv j \pmod t$. When $j \le t - 1$, it follows directly that ${}^t w = j$. We can identify the value ${}^t w$ in this step by testing the numbers of j, $0 \le j \le t$-1, sequentially until we identify the j value fulfilling the above written congruence. Such j is then the value of ${}^t w$ under search.

Step 3: Determination of the value $w = \log_g r$

In the preceding steps, we have obtained a set of the following congruencies:

- $w \equiv {}^s w \pmod{2^s}$
- $w \equiv {}^t w \pmod t$

It also holds that $\gcd(t, 2^s) = 1$, and so, according to the Chinese Remainder Theorem (CRT), an unique value $0 \le w < t*2^s$ exists, such that it fulfils both congruencies. As the value $t*2^s$ is also the order of the group $\boldsymbol{Z}_p^*$ for $p = t*2^s+1$, the value of w is also the discrete logarithm of the value r under search. Direct procedure leading to determination of w follows:

1. set $\gamma \equiv (2^s)^{-1} \pmod{t}$; note that the value exists and is unique, because $\gcd(t, 2^s) = 1$
2. set $v = ({}^tw - {}^sw)*\gamma \bmod t$
3. return $w = {}^sw + v*2^s$

Proof (of correctness of foregoing procedure). For factor 2^s, it is obvious from the expression for w that $w \equiv {}^sw \pmod{2^s}$. For factor t, we obtain that $w \equiv {}^sw + ({}^tw - {}^sw)(2^s)^{-1}*2^s \pmod{t}$, and so $w \equiv {}^tw \pmod{t}$. Moreover, $w = {}^sw + v*2^s < 2^s + (t - 1)*2^s = t*2^s$. By this, we have verified that the above given procedure actually corresponds to application of CRT on the above given set of congruencies.

■

Experimental Results

The procedure described in steps 1 through 3 was applied in various configurations of low-cost office PCs. Table 3 shows average time of calculations for randomly chosen values of r. We can see that the entire calculation takes some hundreds of milliseconds in general.

Table 3. Times of calculation of the value $w = \log_g r$ for a special type of $\boldsymbol{Z}_p^*$

Configuration	Mean time of calculation of a single discrete logarithm (in milliseconds)
Pentium III/ 500MHz 128 MB RAM Windows NT 4.0 SP 6a	96
Celeron 400 MHz 128 MB RAM Windows NT 4.0 SP 6a	113
Pentium II 400 MHz 128 MB RAM Windows 2000 Advanced Server	116
Pentium II 300 MHz 128 MB RAM Windows NT 4.0 SP 6a	150
Pentium 166 MHz 96 MB RAM Windows NT 4.0 SP 6a	535
Pentium 75 MHz 46 MB RAM Windows NT 4.0 SP 4	1020

Appendix 2: Attack on a Private RSA Key

In this appendix, we describe briefly how the value of a private RSA key can be obtained from the value of a faulty signature which was calculated using the affected private key. The attack is based on an analysis of the OpenPGP format. At the time o writing the original paper, it could not be applied on the PGP™ directly, because the PGP™ program performs additional check of the private key integrity going beyond the scope of OpenPGP definition. However, such an attack is possible in case of applications implemented strictly according to OpenPGP, and then it has the same effects as the above presented attack to DSA.

By OpenPGP, the private RSA key consists of the following set of six values (n, p, q, $pInv$, e, d), where p, q are prime numbers, $n = p*q$ is a public modulus, $p*pInv \equiv 1 \pmod{q}$, e is a public exponent and d is a private exponent, i.e. $e*d \equiv 1 \pmod{\text{lcm}(q - 1, p - 1)}$. Basing on this structure, we can assume that the RSA signature transformation is calculated for the specific value of formatted message m by the following procedure:

1. $s_1 = m^d \bmod p$
2. $s_2 = m^d \bmod q$
3. $h = pInv*(s_2 - s_1) \bmod q$
4. $s = s_1 + p*h$
5. s is the result of signature transformation; it holds that $s = m^d \bmod n$

This procedure corresponds to application of the Chinese Remainder Theorem on the signing transformation, and it makes it possible to calculate this transformation efficiently. As it was shown in [6] and later published in [5], application of this technique is quite susceptible to fault side channel attacks making use of an error in the signature calculation. Note that the first public discussion of fault attacks on RSA was probably in the article [2]. However, we use a special variant of these attacks here, which is based on the thought presented for the first time in [6]. And these errors can be implemented not only by affecting of the attacked device during calculation of signature, but also e.g. by affecting of certain values forming private key [9]. Here we draw our attention to one specific type of attack, which should be considered in OpenPGP. In particular, our aim is here to induce a faulty computation producing a signature s' of a known formatted message m, such that

$$(s' - s) \bmod \alpha = 0 \text{ and } (s' - s) \bmod \beta \neq 0,$$

where α, β are particular prime factors of n, $\alpha\beta = n$, and s is a valid (i.e. computed without faults) signature of m.

Given such a set of equations, it is then easy to compute the prime factors of the modulus n as $\alpha = \gcd(n, (s' - s))$ and $\beta = n/\alpha$. An obvious obstacle here is that the attacker doesn't know the value of s. However, it was shown [6] that if the attacker knows the value of the formatted message m (which is true when the victim sends her message together the signature and there are no random values added during

formatting the message; these conditions usually both holds in the case of OpenPGP c.f. bellow), then the equations can be rewritten as

$$((s')^e - m) \bmod \alpha = 0 \text{ and } ((s')^e - m) \bmod \beta \neq 0,$$

where we used that for the valid signature s it holds that $s^e \equiv m \pmod{\alpha\beta}$.

The prime factors of the modulus n are then computed as $\alpha = \gcd(n, ((s')^e - m) \bmod n)$ and $\beta = n/\alpha$. Let us note that the reduction modulo n in the expression for α is used solely for the purpose of effective computation and it is of no other special algebraic magnitude. It easy to observe, that a faulty computation leading to the above written equations can be induced by affecting certain values of the private key (n, p, q, *pInv*, e, d). In particular, we focus here our attention on substituting *pInv* with *pInv'*$\in \boldsymbol{Z}$, such that (*pInv'* – *pInv*) mod $q \neq 0$. Let us mention that random change of *pInv* will fulfill this condition with a high probability. Other parameters of the private key are left without changes.

Now, let us have a pair of values (m, s'), where the value of s' was obtained as a result of above described signature transformation in which the affected value of *pInv´* was used. It applies that:

1. $s_1 = m^d \bmod p$
2. $s_2 = m^d \bmod q$
3. $h' = pInv' * (s_2 - s_1) \bmod q$
4. $s' = s_1 + p * h'$
5. s' is the result of signing transformation

Regarding the factor p for this value as it results from the equation given in point 4, it holds that $(s' - m^d) \bmod p = 0$ implying that $((s')^e - m) \bmod p = 0$. Furthermore, it is very likely (with a probability close to $1 - q^{-1}$) that for the factor q it holds that $(s' - m^d) \bmod q \neq 0$ implying that $((s')^e - m) \bmod q \neq 0$. According to the above given analysis, we can factor n as $p = \gcd(n, ((s')^e - m) \bmod n)$ and $q = n/p$. By this procedure, we have obtained the particular prime factors of n from a single faulty signature. The remaining secret values of the private key can be determined from it easily, which completes our attack.

Let us mention that the above given procedure is based on a condition that the attacker knows the value of formatted message m which directly enters the RSA signature transformation. This condition need not be fulfilled for all types of formats, but the format RFC 2313 (alias PKCS#1, version 1.5) is recommended in OpenPGP, where this condition is met.

Similarly as in case of attack to DSA, we recommend to introduce more powerful check of data integrity in the private key files into the format OpenPGP. No correction is necessary directly in the program PGP™ 7.0.3, as subsequent checks of algebraic relations between the values of private key are employed, which defeat the attempts for attack of this type.

C. Further Results and Considerations on Side Channel Attacks on RSA*

1. Introduction

In 1998, Bleichenbacher [5] described an attack on the PKCS#1 v.1.5 encoding and in 2001 Manger [16] described an attack on the improved scheme EME-OAEP PKCS#1 v.2.1, called also RSAES-OAEP. These attacks underline the significance of the theorem of RSA individual bits [14] which states that: *If RSA cannot be broken in a random polynomial time, then it is not possible to predict the value of any selected bit of the plaintext with a probability not negligibly different from 1/2*. A negligible difference for the purpose of this theorem is such $\varepsilon(n)$ that for any constant $c > 0$ it holds that $\varepsilon(n) < L(n)^{-c}$, where $L(n)$ is the length of an appropriate sufficiently large RSA modulus n. From the standpoint of side channels, it is important to understand this theorem as saying: *If the value of any chosen bit of the plaintext can be predicted with a probability not negligibly different from 1/2 then RSA can be broken within a random polynomial time*. Breaking RSA [22] is understood here to mean that a value of the plaintext is obtained. Bleichenbacher's and Manger's attacks use side channels which provide the attacker with a relatively large amount of information about the plaintext (at least that the two most significant bytes are 00 02 or the first one is 00, respectively).

In this part of the thesis, plaintext will always mean a value of m which is created immediately after an operation with a private RSA key, $m = c^d \bmod n$, not the value of M obtained after decoding m.

In Section 2, we present another possible attack on the RSAES-OAEP (PKCS#1 v.2.1) scheme. It is a chosen ciphertext based side channel attack using only the side information about Hamming weight of certain 32-bit words produced in the process of decoding m by the EME-OAEP-DECODE procedure according to PKCS#1 v.2.1. Theoretically, it is a weakening of the assumptions of Manger's and Bleichenbacher's attacks. From the practical point of view, the new attack can be used especially on smart cards. It follows from the theorem of RSA individual bits that it is necessary to prevent the leakage of any information about the individual bits of the plaintext. Our attack demonstrates that the Hamming weight of a part of the plaintext can be used to carry out a successful attack.

In Section 3, we present a very simple but efficient conversion of the Manger/Bleichenbacher breaking oracle to a universal (signature) oracle. The principle that a private RSA key should not be used simultaneously for encryption and for digital signature is well known but is very often violated in practice. Typical examples include some of the current implementations of Public Key Infrastructure

* An extended version of the paper published as Klíma, V. and Rosa, T.: *Further Results and Considerations on Side Channel Attacks on RSA*, In proc. of CHES 2002, San Francisco Bay, USA, August 2002, pp. 245-260, Springer-Verlag, 2003.

(PKI), the SSL/TLS protocol etc. We show that if we can perform Bleichenbacher's or Manger's attack on the encryption scheme using PKCS#1 (v.1.5 or v.2.1) in such a way that we can obtain the plaintext then we can also obtain the digital signature of any message (encoded in any way) using the same private RSA key. In the SSL protocol this means at least a theoretical ability (c.f. discussion in chapter F, §3.4 of the thesis) the ability to create signatures with the server-side private key and even create false servers with the identity of the original server, provided that sufficient decrypting speed can be ensured.

In Section 4, we present a new fault side channel attack on the RSA-KEM. RSA-KEM attempted to remove the structural relations in order to prevent leaking of information about the plaintext. Despite this we discovered a natural method of obtaining such information. Input plaintext for RSA-KEM consists of symmetric encryption keys, information about which can be obtained by means of an integrity check of the messages they encrypt (e.g. checking the PKCS#5 [19] padding). The result produced by the attack that uses this information is a private RSA key whilst the attacks on PKCS#1 v.1.5 and 2.1 always discovered only a plaintext.

2. Side Channel Attack on RSAES-OAEP Plaintext

In this section, we will demonstrate a new method of attacking the RSAES-OAEP scheme (PKCS#1 v.2.1 [18]) at the time when decoding operation EME-OAEP-DECODE(*EM*, *P*) is performed, see Fig. 1. The attack is based on the assumption that there is a side channel carrying some information about the plaintext. In particular we assume that the attacker can obtain the Hamming weight w(*x*) (i.e. the number of '1' bits) of a word *x* during the time when the plaintext *m* is being processed in the MGF operation (to be specified later). As it was shown in [17], this assumption is realistic, for instance, in power side channels which tend to leak this information in a relatively readable way. We note that this attack is possible with some modifications even when we have access to the Hamming distance of processed data rather than the Hamming weight.

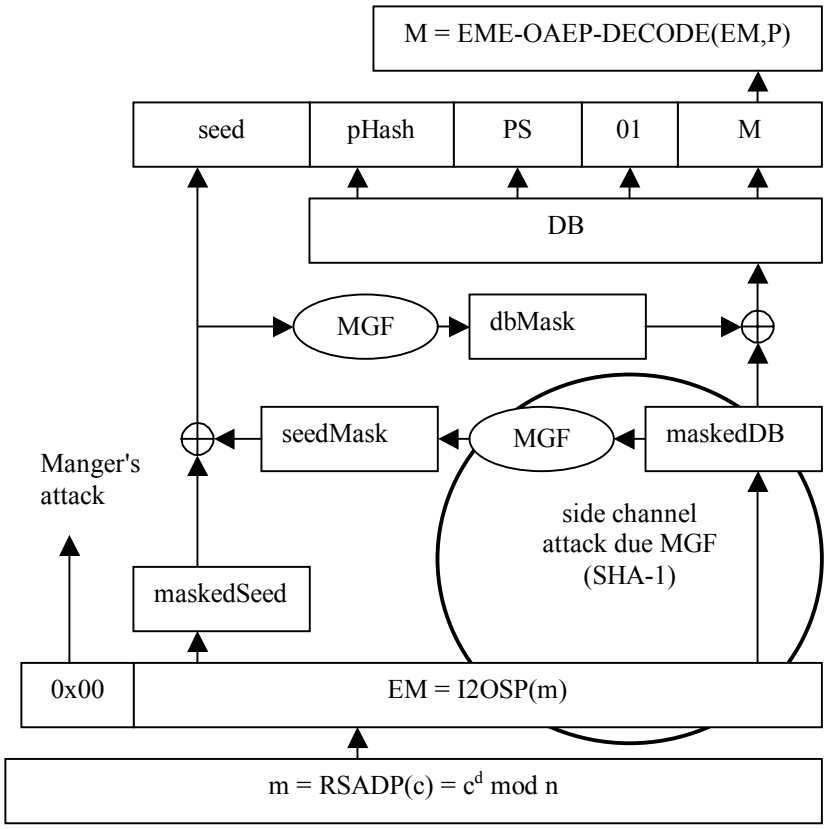

Fig. 1. New side channel attack against RSAES-OAEP

2.1 Attack Description

Consider RSA with a modulus n which has the length of L(n) bits where L(n) is the multiple of 512, i.e. L(n) = 512*k, where k is a natural number. The attack will target the RSAES-OAEP scheme during the processing of the plaintext immediately after the RSA decryption operation c^d mod n, see Fig. 1. *SeedMask* will be computed according to [18] as *seedMask* = MGF(*maskedDB*,20) = SHA-1(*maskedDB* || 00 00 00 00), where the four zero bytes (we will write constants mostly in the hex. notation) are appended to the message by the MGF function. It follows from the definition of OAEP encoding that *maskedDB* always contains 64*k-1-20 bytes, so that 64*k-17 bytes (4 extra zero bytes) enter SHA-1. By the definition of SHA-1 [23] the message is divided into blocks of 64 bytes, which are processed sequentially by the compression function. Note that the least significant bit of the original message m is processed in the last block. It is followed by four zero bytes and 17 bytes of the SHA-1 padding. For various values of L(n) the particular value of the padding is different,

but it is a constant known to the attacker. To present an example, we will consider n, such that L(n) = 1024. Let us denote the i-th byte of the plaintext as $m[i]$ where $m[0]$ is the least significant byte. The last block entering the SHA-1 compression function is in this case equal to m[42.....0] 00 || 00 00 00 80 || 00 00 00 00 || 00 00 00 00 || 00 00 00 00 || 00 00 03 78, where m is followed by 4 zero bytes (from MGF) and the SHA-1 padding. The padding consists of bit 1, 71 zero bits and a 64-bit representation of the message bit length. The length is 888_{10} = 0x00000000 00000378 bits in this case (64*2–17 = 111_{10} bytes). The SHA-1 compression function fills this last block into 32-bit variables W_0, ..., W_{15}, where W_8 = m[10] m[9] m[8] m[7], W_9 = m[6] m[5] m[4] m[3], W_{10} = m[2] m[1] m[0] 00, W_{11} = 00 00 00 80, W_{12} = 00 00 00 00, W_{13} = 00 00 00 00, W_{14} = 00 00 00 00, W_{15} = 00 00 03 78. And then expansion to words W_{16}, ..., W_{79} is performed according to the following relations (where S^1 denotes the left cyclic shift by one bit) $W_{16} = S^1(W_{13}$ xor W_8 xor W_2 xor $W_0)$, $W_{17} = S^1(W_{14}$ xor W_9 xor W_3 xor $W_1)$, $W_{18} = S^1(W_{15}$ xor W_{10} xor W_4 xor $W_2)$, etc. When calculating W_{16}, the first operation performed is W_{13} xor W_8, where W_{13} is a known constant. This moment is an example of a general situation when D-1 known parameters and one unknown enter a D-ary operation. Here, various side channels are often applicable, especially the power side channel.

We assume that the attacker is able to gather the Hamming weight w(W_8) ∈ {0, ..., 32} of word W_8 during the W_{13} xor W_8 operation (W_8 is the only unknown operand in it). The same situation arises in the following two operations as well, so we are able to gather w(W_9) and w(W_{10}).

We number the bits of the word W_i (from the msb to the lsb) as $W_{i,31}$ $W_{i,30}$ $W_{i,29}$ ... $W_{i,0}$. We will show that now we can predict the value of $W_{10,8}$ with a probability not negligibly different from 1/2. Note that this is the value of the least significant bit (lsb) of the plaintext m. Hence, using the theorem of RSA individual bits [14] we can design an attack on the entire plaintext. It is widely known that information about the lsb of the plaintext leads to very efficient attacks [27, p.144].

2.2 Obtaining the Least Significant Bit of a Plaintext (Building an lsb-Oracle)

The procedure which leads to obtaining the value of $W_{10,8}$ (i.e. lsb(m)) is as follows. We denote the ciphertext to be attacked as c, the modulus as n and the public RSA exponent as e. First, we let the attacked device decrypt and decode the original ciphertext c. During decoding phase, we gather the values of Hamming weights w(W_8), w(W_9), and w(W_{10}). In the next step, we request the equipment to decrypt and decode the value $c' = c*2^{-e} \bmod n$. Plaintext m' is the result of this and during the calculation we will obtain Hamming weights w(W_8'), w(W_9'), and w(W_{10}'). If the bit $W_{10,8}$ is zero, then the decryption returns the value $m' = m >> 1$, where ">> 1" means a shift one bit to the right. Otherwise $m' = (m + n) >> 1$. If we assume $W_{10,8} = 0$ then (W_8', W_9', W_{10}') will be created of (W_8, W_9, W_{10}) by a shift one bit to the right (with the exception of W_{10}, where the shift only affects the leftmost bits which are then independently complemented by eight zero bits). The difference between appropriate Hamming weights w(W_8), w(W_9), w(W_{10}) and w(W_8'), w(W_9'), w(W_{10}') is therefore 0 or 1. More precisely w(W_8') = w(W_8) - $W_{8,0}$ + $W_{7,0}$, w(W_9') = w(W_9) - $W_{9,0}$ + $W_{8,0}$,

$w(W_{10}') = w(W_{10}) - W_{10,8} + W_{9,0} = w(W_{10}) + W_{9,0}$ and therefore the three relations included in exactly one of the eight rows of Table 1 are valid.

Table 1. Possible relations among random variables W and W' when $W_{10,8} = 0$

$W_{9,0}$	$W_{8,0}$	$W_{7,0}$	Possible relations		
0	0	0	$w(W_{10}') = w(W_{10})$	$w(W_9') = w(W_9)$	$w(W_8') = w(W_8)$
0	0	1	$w(W_{10}') = w(W_{10})$	$w(W_9') = w(W_9)$	$w(W_8') = w(W_8) +1$
0	1	0	$w(W_{10}') = w(W_{10})$	$w(W_9') = w(W_9) +1$	$w(W_8') = w(W_8) -1$
0	1	1	$w(W_{10}') = w(W_{10})$	$w(W_9') = w(W_9) +1$	$w(W_8') = w(W_8)$
1	0	0	$w(W_{10}') = w(W_{10}) +1$	$w(W_9') = w(W_9) -1$	$w(W_8') = w(W_8)$
1	0	1	$w(W_{10}') = w(W_{10}) +1$	$w(W_9') = w(W_9) -1$	$w(W_8') = w(W_8) +1$
1	1	0	$w(W_{10}') = w(W_{10}) +1$	$w(W_9') = w(W_9)$	$w(W_8') = w(W_8) -1$
1	1	1	$w(W_{10}') = w(W_{10}) +1$	$w(W_9') = w(W_9)$	$w(W_8') = w(W_8)$

However, if $W_{10,8} = 1$, m' is not created by a shift of m, but produced as $(m + n) >> 1$. This, with a high probability, destroys the linear relations in the Table 1. By the obtained weights $(w(W_8), w(W_9), w(W_{10}))$ and $(w(W_8'), w(W_9'), w(W_{10}'))$ we determine whether they fit all relations in any single row. If so, we adopt a hypothesis that $W_{10,8} = 0$, otherwise we refuse it and assume that $W_{10,8} = 1$. The probability of establishing the bit $W_{10,8}$ correctly is close to 1 for an ideal side channel. It will be sufficient to realize that m is randomized by a hash function in MGF and n is assumed to be common, not specially constructed. Therefore, the probability of adopting the hypothesis that $W_{10,8} = 0$ if it was $W_{10,8} = 1$, can be estimated as the probability that the random variables W_8, W_9, W_{10} and W_8', W_9', W_{10}' (with the properties that lower nine bits of W_{10} are 100000000_2 and lower eight bits of W_{10}' are 00000000_2) will fit all the relations in some row of Table 1, which is approximately 0.008. That enables us to obtain the least significant bit of the plaintext m with a high probability and therefore, in accordance with [14] we can establish the remaining part of m.

For the demonstration purpose, the procedures in [14] can be used directly, in particular we suggest the methods based on computing gcd (for details see [2]). However, some improvements of these procedures are necessary when planning a real practical attack (mainly with respect to a minimization of oracle calls, because some devices may limit the total amount of RSA decryptions). First, we need to compute our oracle's advantage, which we define in the following way: Let the $lsb(m)$ be the least significant bit of the plaintext m corresponding the ciphertext c and let the $O_{lsb}(c)$ be the oracle's estimate of $lsb(m)$. We assume that the oracle works according to the procedure described above. The advantage *adv* is defined as $adv = |P[lsb(m) = O_{lsb}(c)] - 1/2|$, where the probability of correct estimation, $P[lsb(m) = O_{lsb}(c)]$, is computed over the probability space of all possible ciphertexts and all possible oracle internal coin tosses. From [14], we have that the *adv* must be at least non-negligible (c.f. above). The higher advantage the better oracle we have. Of course, better oracle leads to a more efficient attack. For instance, if we have an oracle with $adv = 1/2$, then we can use well known and rather quick methods, needing approximately $O(L(n))$ oracle calls (c.f. for example [27, p.144]).

If $adv < 1/2$, we have to employ some methods, which are equipped with a built-in error correction. In fact, these methods must have been already employed in the proofs of theorems in [2], [14]. But these proofs have rather existential form, which is not suitable for a practical attack. However, there are stronger proofs developed in [10] and improved later in [11], which can be used to mount practically feasible attacks. In particular, we suggest to use the *RSA inversion* algorithm ([11, p.226]) which describes a randomized algorithm for the RSA decryption, which needs approximately $O(\mathrm{L}(n)^2 adv^{-2})$ oracle calls ([11, p.223]).

Note that using the absolute value for *adv* (c.f. definition above) is possible here since there is no dependence between previous oracle responses and further oracle calls in the *RSA inversion* algorithm. Therefore, we can run this algorithm (in particular parts 2. and 3. – c.f. [11, p.223]) twice, once for $O_{lsb}(c)$, once for $\mathrm{neg}(O_{lsb}(c))$, where we use simple inversion of the responses captured in the previous run. Such a method induces only a constant multiplicative slow down in the computational part of the algorithm, without an increase of the number of total oracle calls. On the other hand, this method allows to exploit any correlation between oracle response and the correct value of $\mathrm{lsb}(m)$. This further relaxes requirements on the quality of particular side channel used in this attack.

There are other questions, which have to be carefully answered when developing an efficient attack – namely on how to measure Hamming weights, whether to do some error corrections during a measurement phase or whether to let it all on a majority decision used in the *RSA inversion* algorithm, etc. In this study, we strive to show that such an attack is possible and that it operates in a random polynomial time, having in mind that its concrete efficiency strongly depends on a particular implementation. From here, we would like to emphasize the importance of a thorough implementation, which cannot simply be reduced to the problem of finding "the right encoding method" as was perhaps deemed earlier.

3. Note on Converting the Deciphering Oracle to a Signing Oracle

In this section, we will demonstrate that if the attacker can use Bleichenbacher's or Manger's attack on the PKCS#1 v.1.5 or 2.1 encryption scheme, she is also able to create false signatures using the same private RSA key with any encoding of the message to be signed. This conversion is technically very simple but it has interesting practical consequences on the applications where the same key is used both for encryption and for digital signature. One example is the SSL/TLS protocol used to secure access to web servers. In its application, the public key certificate at the server sometimes permits the use of the key both for encryption and for signature. That means that a signature made by the server's private key is meaningful in the PKI system and it is not appropriate that it should be forgeable. Conversion will be demonstrated for both Bleichenbacher's attack on PKCS#1 v.1.5 and for Manger's attack on PKCS#1 v.2.1. Manger's attack uses only one element of the EME-OAEP PKCS#1 v.2.1 encoding - whether a zero occurred in the most significant byte (MSB) of the plaintext decrypted by the private key. We will denote the oracle which tells the attacker this as "*Partial information oracle*" $\mathrm{PIO}_{\mathrm{MSB}}$: $\mathrm{PIO}_{\mathrm{MSB}}(c)$ = "yes" iff $c = m^e$

mod n, MSB(m) = 0x00. Using this oracle a decryption machine (*Whole information oracle*) WIO_{MSB} is constructed in [16]. If the plaintext has a format of m = 00 || ...., then the WIO_{MSB} (using PIO_{MSB}) can extract from the ciphertext c the original plaintext $m = WIO_{MSB}(c) = c^d \bmod n$. Now, we will assume that the same private key (d) is used in another RSA scheme (with any encoding) for digital signature. The attacker can now easily forge the digital signature of any message using the same private key (d) if she has access to PIO_{MSB}. Let c be the message that the attacker prepares for signing. She then selects different random natural numbers $r = r_1, r_2, ...$ smaller than n and sends $c' = c*r^e \bmod n$ to the oracle PIO_{MSB} successively. After decryption there is calculated $m' = m*r \bmod n$ on the recipient's side. Unless the most significant byte of m' is zero, it is rejected by PIO_{MSB}: $PIO_{MSB}(c')$ = "no". Because the most significant byte of m' is random, it is zero with a probability of 1/256. After several hundreds of trials, the value of c' will conform with the initial condition of Manger's attack and WIO_{MSB} then decrypts c': $m' = WIO_{MSB}(c') = (c')^d \bmod n$. The attacker then only has to calculate $m = m'* r^{-1} \bmod n$ as a valid signature of the message c. The particular type of encoding for a signature is irrelevant here. The attacker follows the same procedure when converting Bleichenbacher's attack. This attack assumes the oracle $PIO_{PKCS\text{-}CONF}$ which tells the attacker whether the plaintext produced by decryption is “PKCS#1 conforming” [5]. That means that the two most significant bytes of the plaintext must be equal to 00 || 02 and from the 11th byte onwards some byte must be zero (the separator). On the basis of $PIO_{PKCS\text{-}CONF}$ a decryption machine $WIO_{PKCS\text{-}CONF}$ is then constructed. If the plaintext is “PKCS#1 conforming”, then $WIO_{PKCS\text{-}CONF}$ can use $PIO_{PKCS\text{-}CONF}$ on the corresponding ciphertext c to obtain the original plaintext $m = WIO_{PKCS\text{-}CONF}(c) = c^d \bmod n$. Using the same procedure as above, i.e. by a randomly selected r, we test whether $PIO_{PKCS\text{-}CONF}$ on $c' = c*r^e \bmod n$ responds “yes”. This time, the probability of such answer is several hundred times lower than in the case of Manger's attack (depending on the number of bits of n; for 1024 it is approximately 715-times less, see [16]). As soon as such a situation occurs, the attacker can again compute $m = m'* r^{-1} \bmod n$ as a valid signature of the message c. Note that this conversion is actually implicitly contained in the full list of Bleichenbacher’s attack procedure [16], so we only recall it here explicitly. Also note that the attack described in Section 2 of this paper does not place any special requirements on the ciphertext. It is therefore suitable for forging signatures even without any changes.

In the case of the SSL/TLS protocol, the concrete threat of this attack depends not on the protocol itself, but rather on the PKI, which the particular server works in. This PKI manages the server certificate and this PKI decides (via certificate attributes) whether signatures on behalf of that server are meaningful or not. In practice, we have seen many server certificates, which were attributed for the purpose of document signing as well.

4 Side Channel Attack on RSA-KEM

After Bleichenbacher's attack on the scheme PKCS#1 v.1.5, the new scheme PKCS#1 v.2.1, based on the EME-OAEP encoding, was recommended for use. However,

Manger's attack [16] showed that RSAES-OAEP is also vulnerable to side channel attacks. After that Shoup [24] proposed the new key encapsulation mechanism RSA-KEM. This mechanism was believed to have eliminated problems with side channels. We show that RSA-KEM is also vulnerable to some types of side channel attacks, and therefore has to be implemented carefully. Next, we will describe an RSA *confirmation oracle* (CO) based on RSA-KEM and show how to use a CO to obtain a RSA private key.

4.1 Confirmation Oracle

The purpose of RSA-KEM is to transmit the symmetric key to the receiver, and so it is natural to consider the properties of the whole hybrid public-key encryption scheme H-PKE$_{KEM, DEM}$, consisting of the Data Encapsulation Mechanism (DEM) and the Key Encapsulation Mechanism (KEM) (c.f. [24]). Our attack on RSA-KEM is based on the behaviour of the entire hybrid scheme. Its requirements are sufficiently general and make it easily realizable in practical applications. We will start by reviewing some important terms from [24] in a simplified form:

The Key Encapsulation Mechanism (KEM) has this abstract interface:

KEM.Encrypt(*PubKey*) $\rightarrow$ (K, C_0) - generates a symmetric encryption key K and by using the public key *PubKey* creates a corresponding ciphertext C_0

KEM.Decrypt(*PrivKey*, C_0) $\rightarrow$ (K) - decrypts C_0 using the private key *PrivKey* and derives the symmetric key K by applying the key derivation function KDF to that result

The Data Encapsulation Mechanism (DEM) has this abstract interface:

DEM.Encrypt(K, M) $\rightarrow$ (C_1) - encrypts the message M with the symmetric key K and returns the corresponding ciphertext C_1

DEM.Decrypt(K, C_1) $\rightarrow$ (M) - decrypts the ciphertext C_1 with the symmetric key K and returns the plaintext M

The hybrid public-key encryption scheme H-PKE$_{KEM, DEM}$ is a combination of the KEM and DEM schemes. The algorithm for the encryption of a message M by the public key *PubKey* resulting in the ciphertext C is as follows:

1. (K, C_0) = KEM.Encrypt(*PubKey*)
2. C_1 = DEM.Encrypt(K, M)
3. Ciphertext $C = C_0 \parallel C_1$

On the receiving end, the decryption of the ciphertext C with the private key *PrivKey* is carried out as follows:

1. Let $C = C_0 \parallel C_1$
2. K = KEM.Decrypt(C_0)
3. M = DEM.Decrypt(K, C_1)

We assume that there is no integrity check for the key K (e.g. analogous to a check used in the encoding method OAEP), however, an integrity check exists for the

message *M* in the third step. It can be based on the message padding check, as in PKCS#5 [19], on the usage of labels as described in [24], or on any other technique. We assume that the attacker will find out whether the receiver's integrity check rejects a ciphertext *C*. In this situation, we can expect that the receiver will send an error message to the sender. Acceptance or rejection of a ciphertext *C* defines the *receiver oracle* (RO). On the basis of RO we can define the *confirmation oracle* (CO). This term may be defined more generally, however, we will only define the RSA confirmation oracle (RSA-CO) here (c.f. its other variants in parts D and E of the thesis).

We assume that the private key *PrivKey* is a private exponent *d* and *n* is a public modulus. Later on, we will show that the modulus *n* should be a part of the private key rather than independently taken from the public key, as it is recommended in [24].

Definition: RSA confirmation oracle RSA-$CO_{d, n}(r, y)$.
Let us have a receiver oracle RO that uses RSA in the hybrid encryption H-$PKE_{KEM,DEM}$. We will construct a RSA confirmation oracle RSA-$CO_{d, n}(r, y) \rightarrow$ (ANSWER = "yes/no") as follows:

1. *$K = KDF(r)$; KDF - Key Derivation Function*
2. *$C_0 = y$; for simplicity we omit the conversion between integers and strings*
3. *$C_1 = DEM.Encrypt(K, M)$; where M contains an integrity check*
4. *$C = C_0 \,||\, C_1$*
5. *Send the ciphertext C to the receiver oracle $RO_{d, n}$. RO then continues:*
   a. *Compute $K = KEM.Decrypt(d, C_0)$ following these steps:*
      i. *Check if $y = C_0 < n$. If not, an error has occurred.*
      ii. *Compute $r' = (y^d \bmod n)$*
      iii. *$K' = KDF(r')$*
   b. *$M' = DEM.Decrypt(K', C_1)$*
   c. *Check the integrity of M'*
   d. *If it is correct, the answer of RO is "yes", otherwise it is "no"*
6. *The answer of RSA-$CO_{d, n}(r, y)$ is "yes", if RO returned "yes", otherwise it is "no"*

We note that whenever $r = (y^d \bmod n)$, the oracle returns "yes". If $r \neq (y^d \bmod n)$ then the oracle returns "no" with a high probability close to 1 (the value depends on collisions in the function KDF and the strength of the integrity check). The key point is that an attacker may use the oracle RSA-$CO_{d, n}(r, y)$ to check the congruence $r \equiv y^d \pmod n$ without knowledge of the particular value of the exponent *d* used in the step 5.a.ii above.

4.1.2 Note on Oracle Errors

Let us recall, that the following implication holds with the probability 1:

$$\text{if } r = (y^d \bmod n) \text{ then RSA-}CO_{d, n}(r, y) = \text{"yes"}.$$

However, we should generally assume even if $r \neq (y^d \bmod n)$ then there is still certain nonzero probability P_{err} that RSA-$CO_{d, n}(r, y)$ = "yes". Generally, we may also assume that the better hybrid encryption scheme is used (mainly with respect to

integrity checks), the closer to 0 the value of P_{err} is. For instance, if P_{err} is to be determined by integrity check strength and SHA-1 based HMAC is used for that purpose, then P_{err} is close to 2^{-160}, since this is the a-priori probability that an incorrect unknown symmetrical key would lead to a correct HMAC computation. Therefore, in many practical situations, we can get around the problem by simply ignoring the effect of P_{err}, which is the reason why a deeper elaboration of this subject will be omitted in the following practical attacks description.

4.2 Fault Side Channel Attacks

The congruence $r \equiv y^d \pmod{n}$, where d is the private exponent, can be confirmed with the public key as well. However, using RSA-$CO_{d, n}(r, y)$ is the natural way of exploiting the receiver's behaviour. The oracle becomes far more interesting when an error occurs in step 5.a.ii of the algorithm above. This confirmation oracle can be used to design many attacks. Therefore, we will only present a brief description of two examples to illustrate the core of this problem. We note that these attacks are targeted at the private key, rather than the plaintext. This is paradoxically caused by the absence of structural checks of the plaintext in RSA-KEM, which is really a positive quality in other contexts.

4.2.1 Faults in the Bits of the Private Exponent *d*

The impact of faults in the bits of the private exponent RSA was described in [3]. We will show that the confirmation oracle RSA-$CO_{d, n}$ can be used to mount these attacks on the hybrid encryption scheme based on RSA-KEM. As an example, we will assume that the attacker is able to swap the i-th bit d_i of the receiver's private exponent d (in step 5.a.ii), and this change will go undetected by the receiver. Such a situation can occur, for instance, with chip cards and similar devices [26].

Let us assume that a fault occurred in the i-th bit d_i and let us denote by d' the defect value of the private exponent. Depending on the value of d_i, either $d' = d + I$ or $d' = d - I$, where $I = 2^i$. Let $\alpha \equiv y^I \pmod{n}$ and $\alpha^* \alpha^{-1} \equiv 1 \pmod{n}$. For the value $r = y^{d'} \bmod n$ we have:

$r = (y^d * \alpha \bmod n)$ if $d_i = 0$
$r = (y^d * \alpha^{-1} \bmod n)$ if $d_i = 1$

Using the access to the confirmation oracle RSA-$CO_{d', n}$ we can find out the value of d_i in this way:

1. Randomly pick x, $0 < x < n$
2. Compute $y = x^e \bmod n$, where e is the corresponding public exponent RSA
3. Compute $\alpha = y^I \bmod n$, if $\alpha = 1$ goto 1
4. Compute $r = x * \alpha \bmod n$
5. If RSA-$CO_{d', n}(r, y)$ returns "yes" then set $d_i = 0$ else set $d_i = 1$.

We can repeat this procedure for various bit positions (and their combinations) and thus obtain the whole private key d. In the case of irreversible changes we will gradually carry out an appropriate correction in step 3 using the previously obtained

bits. In this way, the corruption of d is allowed to be irreversible. Moreover, it is enough to obtain only a part of d from which the remaining bits can be computed analytically in a doable time, see overview in [7]. In [3] and [8], we may find other sophisticated attacks of this type which were further extended in [6]. We have presented the confirmation oracle as an "interface" that allows the attacker to apply some general attacks on "unformatted RSA" to RSA-KEM.

4.2.2 The Usage of Trojan Modulus

We have mentioned that in the RSA-KEM scheme, the modulus n is not a part of the private key. This would allow for a change of the modulus n without any security alarm. The following attack shows the need to change this set up.

Let us assume that we can obtain the value $x = g^d \bmod n'$ for an unknown exponent d and arbitrary values of g and n'. It is widely known that one such value x is sufficient to discover d. We can, for instance, choose a modulus n' to be a prime in the form $n' = t*2^s + 1$, where t is a very small prime number and 2^s is a very large natural number. Further we choose g to be a generator of the multiplicative group $\boldsymbol{Z}_{n'}^*$. Now we can solve the discrete logarithm problem in $\boldsymbol{Z}_{n'}^*$ by a simple modification of the Pohlig-Hellman algorithm [20] (c.f. also algorithm A1 in chapter C - Appendix 1 of the thesis). However, this algorithm requires the value of x, $x = g^d \bmod n'$, directly, which we cannot obtain from the confirmation oracle. We can only ask the oracle whether the pair of integers (r, y) satisfies the equation $r = y^d \bmod n'$. On a closer look at the Pohlig-Hellman algorithm, we can notice that it can be modified so that the value of x is not needed directly, but only in comparisons of the type $r =^? (x^\alpha \bmod n')$ for some integers r, α. It means that we only want to know whether $r =^? ((g^d)^\alpha \bmod n')$, which can be obtained by calling the confirmation oracle $\text{RSA-CO}_{d, n'}(r, g^\alpha \bmod n')$. This is the main idea of the modification. The complete algorithm A1 is presented in the next subsection.

This attack is also possible even if the modulus n is part of the private key. However, in this case, we can expect that it will be a little bit more difficult to plant a false value of n'. This idea can also be extended to the case when a method based on the Chinese Remainder Theorem is used for operations with the private key.

4.2.3 Algorithm A1: Computation of the Private Exponent Using the Access to a RSA Confirmation Oracle

In the following text, we will describe an efficient algorithm for a private exponent d computation, based on a modified Pohlig-Hellman algorithm for the discrete logarithm problem in the multiplicative group $\boldsymbol{Z}_p^*$. This group has a special structure chosen by an attacker, because the value of p is taken to be the fraudulent modulus n'. The algorithm is also inspired by the ideas presented in Appendix 1 of chapter C of the thesis. It actually uses the same type of $\boldsymbol{Z}_p^*$ to make the whole computation of the discrete logarithm trivially feasible. The main difference here is that the following algorithm is primarily designed to be used with the RSA Confirmation Oracle, therefore, a slightly different algebraic approach to its development is used.

Proposition. *Let us assume to have an access to a confirmation oracle* $\text{RSA-CO}_{d, p}$*, where* p *is a prime such that* $p = t*2^s + 1$ *and* t *is a small prime. Let* g *be the*

generator of Z_p^*. (We note that the order of Z_p^* has to be larger than the highest possible value of d.)

The following procedure computes the private exponent d in the three steps.

Step 1: Computation of the value $D_s = d \bmod 2^s$

Let $d = d_{b-1}*2^{b-1} + d_{b-2}*2^{b-2} + ...+ d_0$, where b is the number of bits of the binary form of d, and $d_i \in \{0, 1\}$, for $0 \le i \le b$-1. Let us denote $I = 2^i$ and $J = 2^j$. We assume that $2^i \mid (p - 1)$ and we define $\delta = g^d \bmod p$ and $D(i) = d \bmod I$. Then
$\delta^{(p-1)/I} \equiv [g^d]^{(p-1)/I} \equiv [g^{(p-1)/I}]^d \equiv [g^{(p-1)/I}]^{d \bmod I} \equiv [g^{(p-1)/I}]^{D(i)} \pmod p$, and hence

$$\delta^{(p-1)/I} \equiv [g^{(p-1)/I}]^{D(i)} \pmod p . \quad (1)$$

The value of $D(i)$ can be expressed as $D(i) = d_{i-1}*2^{i-1} + d_{i-2}*2^{i-2} + ...+ d_0$. We will show that having access to the confirmation oracle we can easily compute the lowest s bits of the private exponent d (one bit of d per one oracle call). We will start with the lowest bit d_0 and inductively go to the bit d_{s-1}. For $i = 1$ from (1) we have $\delta^{(p-1)/2} \equiv [g^{(p-1)/2}]^{d_0} \pmod p$. From the definition of δ, we have $\delta^{(p-1)/2} \equiv [g^{(p-1)/2}]^d \pmod p$, and so

$$[g^{(p-1)/2}]^d \equiv [g^{(p-1)/2}]^{d_0} \pmod p . \quad (2)$$

We note that $g^{(p-1)/2} \equiv p$-1 (mod p), and therefore $[g^{(p-1)/2}]^{d_0} \bmod p$ can achieve only two possible values, depending on the bit d_0. Using the confirmation oracle, we can either confirm or refute the value of d_0 in (2). Let $d_0 = 1$ and let us make the oracle call RSA-$CO_{d, p}(p$-1, p-1), which represents the congruence (2). If the oracle returns "yes", we set $d_0 = 1$, otherwise we set $d_0 = 0$. We note that a correctly generated private exponent RSA should induce $d_0 = 1$, therefore this step can be omitted. We determine the remaining bits of $D(s)$ inductively. Let us assume that we know the value $D(j)$ for some $0 < j < s$. Next, we will compute the value $D(j$+1). From (1) we have

$$\delta^{(p-1)/(2J)} \equiv [g^{(p-1)/(2J)}]^{D(j+1)} \pmod p . \quad (3)$$

Let $\alpha = d_j * 2^j = d_j * J$. Then $D(j$+1) = $d \bmod 2^{j+1} = \alpha + D(j)$. For the value on the right-hand side of (3), we have that $[g^{(p-1)/(2J)}]^{D(j+1)} \equiv [g^{(p-1)/(2J)}]^{\alpha} * [g^{(p-1)/(2J)}]^{D(j)} \equiv [g^{(p-1)/2}]^{d_j} * [g^{(p-1)/(2J)}]^{D(j)} \equiv (p\text{-}1)^{d_j} * [g^{(p-1)/(2J)}]^{D(j)} \pmod p$, so we get $\delta^{(p-1)/(2J)} \equiv$
$\equiv (p\text{-}1)^{d_j} * [g^{(p-1)/(2J)}]^{D(j)} \pmod p$. Using the substitution $\delta = g^d \bmod p$, we obtain

$$[g^{(p-1)/(2J)}]^d \equiv (p\text{-}1)^{d_j} * [g^{(p-1)/(2J)}]^{D(j)} \pmod p . \quad (4)$$

On the right-hand side of (4), almost entirely known values appear, with the exception of the value of d_j. We will again use the confirmation oracle to decide between the two possible values of the bit d_j. We guess that $d_j = 0$ and call the oracle in the form RSA-$CO_{d, p}([g^{(p-1)/(2J)}]^{D(j)} \bmod p$, $g^{(p-1)/(2J)} \bmod p)$, which represents the congruence (4). If the oracle returns "yes", we set $d_j = 0$, otherwise we do the correction $d_j = 1$. The inductive step is finished and we have obtained $D_s = D(s)$.

Step 2: Computation of the value $D_t = d \bmod t$

It is easy to show that an integer j, under the condition $\delta^{(p-1)/t} \equiv [g^{(p-1)/t}]^j \pmod p$, satisfies that $D_t \equiv j \pmod t$. If $j < t$, then we directly obtain that $D_t = j$. Therefore, we can identify the value D_t in this step by successive testing every number $j = 0, ..., t\text{-}1$, until we find the j that satisfies the congruence $\delta^{(p-1)/t} \equiv [g^{(p-1)/t}]^j \pmod p$. This j is then the sought value of D_t. In order to determine this value we rewrite the congruence (using the definition of δ) as follows:

$$[g^{(p-1)/t}]^d \equiv [g^{(p-1)/t}]^j \pmod p \qquad (5)$$

and use the oracle in the form $\text{RSA-CO}_{d,\,p}([g^{(p-1)/t}]^j \bmod p,\ g^{(p-1)/t} \bmod p)$ gradually for $j = 0,..., t\text{-}1$. The correct value of j is reached when the oracle returns "yes", and we set $D_t = j$.

Step 3: Computation of the value d

In the previous steps, we have obtained two congruencies $d \equiv D_s \pmod{2^s}$ and $d \equiv D_t \pmod t$. It also holds that $\gcd(t, 2^s) = 1$, and so by the Chinese Remainder Theorem, there exists a single value $0 \le d < t*2^s$, satisfying both congruencies. The value of d can be computed directly as bellow:

1. Compute γ, $\gamma*2^s \equiv 1 \pmod t$, a unique value exists because $\gcd(t, 2^s) = 1$
2. Compute $v = (D_t - D_s)*\gamma \bmod t$
3. $d = D_s + v*2^s$

Note that this attack requires at most $s + t$ oracle calls together with a trivially feasible number of group multiplications on $\mathbf{Z}_p^*$.

4.2.4 Other Computational Faults

So far, we have only considered the attacks based on modifications of the private exponent d and the modulus n. However, similar attacks may be developed, considering general permanent or transient faults that appear during RSA computations within the function KEM.Decrypt. A discussion on these attacks, however, is beyond the scope of this paper. For more details, the reader may consult papers [3], [8]. We can realistically assume that certain types of attacks described there can be used on RSA-KEM with the use of the confirmation oracle.

4.2.5 Comparison of Attacks on RSA Schemes

Manger [16] showed that the RSAES-OAEP scheme has certain problems with the most significant byte. These problems must be avoided by proper implementation. We have shown that RSA-KEM has similar problems, when fault side channel attacks can occur. Whenever we use RSA-KEM it is therefore essential to exclude fault side channels. We must carry out reliable private key integrity checks (the modulus should be a natural part of the private key) as well as using fault tolerant computations. We still need to consider the consequences of the RSA individual bit theorem and make sure that no information about any individual bit of the plaintext has leaked. Table 2

below contains a brief overview of the current state of most used RSA schemes when side channel attacks are considered.

Table 2. RSA schemes and side channel attacks

	PKCS1 v.1.5	**RSAES-OAEP**	**RSA-KEM**
Public attack	Yes	Yes	Yes
Side channel (information) used in attack	The information about whether the plaintext is PKCS#1 v.1.5 conforming	– The information about whether the most significant byte of plaintext is zero – Hamming weight of processed data	Fault side channel
Information obtained in attack	Plaintext	Plaintext	Private key

4.3. General Countermeasures

When we consider the state-of-the-art in cryptanalysis, we can specify three basic security criteria that need to be satisfied in every cryptosystem design on the RSA basis. These are:

(a) Resistance to adaptive chosen ciphertext attacks
(b) Resistance to side channel information leakage
(c) Resistance to fault side channels

Imperfect resistance to any of these types of attack can result in the ability to decrypt ciphertext (mainly (a)) or to obtain directly the value of the private key (mainly (c)). We have purposely omitted from the list resistance to purely algebraic attacks, such as problems with a low value of the private or public exponent, among other similar ones (their overview appears in [7]), since most successful attacks are based on an incorrect use of RSA and implementation faults. The problem of the correct use of RSA is rooted in the mathematics underlying the algorithm (for details see [14], [2], [7], [8], [16], [5], [10], [11] and attacks presented there) and thus it should be examined from a mathematical perspective. It seems too risky to leave the issue in the hands of implementators. We also note that cryptanalysis has gradually accepted the assumption that an attacker has nearly unlimited access to an attacked system. We do not merely consider attacks on "data passing through" but direct attacks on autonomous cryptographic units.

Furthermore, we can see that it is not possible to satisfactorily solve the defence against the types of attacks specified above by a single universal encoding of data being encrypted. This is a consequence of the fact that the encoding mechanism is only a part of the whole scheme and as such can only affect part of its properties.

Now we will look at basic defence mechanisms against the above types of attacks. The first category, adaptive chosen ciphertext attacks, has not been considered in this paper. We think that a satisfactory solution is the random oracle paradigm [4], which has been successfully applied in [24], [25], and [12]. For category (b), we need to

constantly bear in mind the claim in [14], and prevent any leakage of plaintext information. It is not possible to limit our attention only to the easily visible information such as the value of the most significant byte of plaintext in RSAES-OAEP. In Section 2, we showed that the leakage of information from completely other part of the scheme has also a negative effect on security. Power side channel attacks ([15], [17], [1]) and nascent theory of electromagnetic side channel attacks ([21], [13]) is necessary to be considered a particularly high threat. However, defence measures against these channel attacks [9] are beyond the scope of the analysis presented here. It was our aim to show that these countermeasures need to be used in every single function that deals with individual parts of the plaintext. Here we focused our attention on the function SHA-1 as an example.

Finally, the last category are fault attacks. The vulnerability of RSA to these attacks does not originate directly from the theorem [14]. However, it seems to be an innate quality of the RSA system ([3], [7], [8]), too. As well as with the other types of attacks, certain types of encoding can more or less eliminate fault attacks. We showed that RSA-KEM, despite it seems to be well resistant to other types of attacks [24], can be easily and straightforwardly affected by fault side channel attacks. To avoid fault attacks it is recommended especially:

(i) To consistently check the integrity of the private key and of the other parameters used with it in its processing
(ii) To minimize the range of error messages
(iii) Wherever possible, to use platforms equipped with fault detection and eventually also correction facilities (fault tolerant systems)

As a rather strong countermeasure to prevent the attacks described in §4.2.1 and §4.2.2, we can recommend to check every result $x' = \mathrm{RSADP}(y)$ as $y \stackrel{?}{=} [(x')^e \bmod n]$, where RSADP is the RSA decryption primitive [18], e is the public exponent and n is the modulus. The values of e and n shall be both taken from a trusted record of the public key independently on the record of the private key. Let $y > 0$ and let $y = x^e \bmod n$, $0 < x < n$. If there are no faults, then $\mathrm{RSADP}(y) = y^d \bmod n$, where d is the private exponent, and the above given check trivially passes. If there are errors due to the faults expected in §4.2.1, then $x' = \mathrm{RSADP}(y) = x*y^\varepsilon \bmod n$, such that with a very high probability $y^\varepsilon \bmod n \neq 1$, implying $x \neq x'$. Since the mapping $f: x \rightarrow (x^e \bmod n)$ is injective ([22], [27]), it follows from $x \neq x'$ that $y \neq [(x')^e \bmod n]$. In case of the modifications applied in §4.2.2, the behavior of RSADP is randomized by the substitution of a prime n' which is independent on the original modulus n. Therefore, it holds with a very high probability that $x' = \mathrm{RSADP}(y) \neq x$, so the test again discovers the attack successfully.

5 Conclusion

The RSA individual bits theorem [14] is generally considered to be a good property of RSA. However, it also shows the way for attacks based on side channels [5], [16].

We have presented another possible attack on the encryption scheme RSAES-OAEP where, in contrast with the previous work [16], we attack that part of the

plaintext “shielded” by the OAEP method. In this, we use the algebraic properties of RSA, rather than some weakness of the OAEP encoding. To prevent this attack, we need to eliminate the parasitic leakage of information from individual operations in partial procedures of the entire scheme. This goes well beyond the scope of the general description of the OAEP encoding method. Next, we presented a new side channel attack on the RSA-KEM. This scheme was built to prevent the parasitic leakage of information about the plaintext, especially under the consideration of chosen ciphertext attack. However, we managed to point out a side channel that allows the leakage of this information. Unlike previous attacks that returned the plaintext, this time the attacker obtains the RSA private key. The attack was again made possible by the basic multiplicative property of RSA.

Our contribution underlines the significance of the known algebraic properties of RSA in relation to rapidly evolving attacks based on side channels. Consequently, it is possible to expect similar side channel attacks in other RSA schemes that may employ different message encoding. Therefore, it is necessary to pay more attention to side channel countermeasures in implementations of these cryptographic schemes.

As a small note in our analysis, we pointed out the rule of keeping RSA keys for encryption and digital signature strictly separated, which is often neglected. We assumed that the rule is not adhered to, and mentioned an approach to convert both Manger's and Bleicherbacher's oracles for ciphertext decryption into oracles that can create valid digital signatures for arbitrarily encoded messages.

References

1. Akkar, M.-L., Bevan, R., Dischamp, P., and Moyart, D.: *Power Analysis, What Is Now Possible...*, in Proc. of ASIACRYPT 2000, pp. 489-502, 2000
2. Alexi, W., Chor, B., Goldreich, O., and Schnorr, C.: *RSA and Rabin functions: Certain parts are as hard as the whole*, SIAM Journal on Computing, 17(2), pp. 194-209, 1988
3. Bao, F., Deng, R.-H., Han, Y., Jeng, A., Narasimhalu, A.-D., and Ngair, T.: *Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults*, in Proc. of Security Protocols '97, pp. 115-124, 1997
4. Bellare, M. and Rogaway, P.: *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, October 20, 1995, originally published in Proc. of the First ACM Conference on Computer and Communications Security, ACM, November 1993
5. Bleichenbacher, D.: *Chosen Ciphertexts Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1*, in Proc. of CRYPTO '98, pp. 1-12, 1998
6. Blömer, J. and May, A.: *New Partial Key Exposure Attacks on RSA*, in Proc. of CRYPTO 2003, pp. 27-43, 2003
7. Boneh, D.: *Twenty Years of Attacks on the RSA Cryptosystems*, Notices of the American Mathematical Society, vol. 46, no. 2, pp. 203-213, 1999
8. Boneh, D., DeMillo, R.-A., and Lipton, R.-J.: *On the Importance of Checking Cryptographic Protocols for Faults*, in Proc. of EUROCRYPT '97, pp. 37-51, 1997
9. Chari, S., Jutla, C.-S., Rao, J., and Rohatgi, P.: *Towards Sound Approaches to Counteract Power-Analysis Attacks*, in Proc. of CRYPTO '99, pp. 398-411, 1999
10. Fischlin, R. and Schnorr, C.-P.: *Stronger Security Proofs for RSA and Rabin Bits*, in Proc. of EUROCRYPT '97, pp. 267-279, 1997

11. Fischlin, R. and Schnorr, C.-P.: *Stronger Security Proofs for RSA and Rabin Bits*, Journal of Cryptology, Vol. 13, No. 2, pp. 221-244, IACR, 2000
12. Fujisaki, E., Okamoto, T., Pointcheval, D., and Stern, J.: *RSA-OAEP Is Secure under the RSA Assumption*, in Proc. of CRYPTO 2001, pp. 260-274, 2001
13. Gandolfi, K., Mourtel, C., and Olivier, F.: *Electromagnetic Analysis: Concrete Results*, in Proc. of CHES 2001, pp. 251-261, 2001
14. Håstad, J. and Näslund M.: *The Security of Individual RSA Bits*, in Proc. of FOCS '98, pp. 510-521, 1998
15. Kocher, P., Jaffe, J., and Jun, B.: *Differential Power Analysis: Leaking Secrets*, in Proc. of CRYPTO '99, pp. 388-397, 1999
16. Manger, J.: *A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1*, in Proc. of CRYPTO 2001, pp. 230-238, 2001
17. Messegers, T.-S., Dabbish, E.-A., and Sloan, R.-H.: *Investigations of Power Analysis Attacks on Smartcards*, in Proc. of USENIX Workshop on Smartcard Technology, pp. 151-161, 1999
18. PKCS#1 v2.1: *RSA Cryptography Standard*, RSA Labs, DRAFT2, January 5 2001
19. PKCS#5 v2.0: *Password-Based Cryptography Standard*, RSA Labs, March 25, 1999.
20. Pohlig, S.-C. and Hellman, M.-E.: *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Trans. Inform. Theory, 24 (1978), 106-110
21. Rao, J.-R and Rohatgi, P.: *EMpowering Side-Channel Attacks*, preliminary technical report, May 11 2001
22. Rivest, R.-L., Shamir, A., and Adleman L.: *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, pp. 120-126, 1978.
23. *Secure Hash Standard*, FIPS Pub 180-1, 1995 April 17
24. Shoup, V.: *A Proposal for an ISO Standard for Public Key Encryption (version 2.0)*, September 17, 2001
25. Shoup, V.: *OAEP Reconsidered (Extended Abstract)*, in Proc. of CRYPTO 2001, pp. 239-259, 2001
26. Skorobogatov, S.-P. and Anderson, R.-J.: *Optical Fault Induction Attacks*, In proc. of CHES 2002, pp. 2-12, Springer-Verlag, 2003
27. Stinson, D.-R.: *Cryptography – Theory and Practice*, CRC Press, 1995

D. Strengthened Encryption in the CBC Mode*

1. Introduction

Vaudenay's attack [1] on the CBC mode of block ciphers with the PKCS#5 padding [2] uses the information which states whether the deciphered text had the correct padding. One of Vaudenay's proposals consisted of the encryption of the message $M' = M \,||\, padding \,||\, hash(M \,||\, padding)$ instead of M, but he rejected it because of the theoretical weakness. From the practical point of view, the disadvantage of this proposal is, in particular, that during encryption of the last plaintext block, several cipher text blocks arise, instead of 1 or 2 blocks as at present. This noticeably disrupts the semantics of contemporary cryptographic interfaces, for instance CryptoAPI [3].

The present CBC mode (according to the common semantics of programming interfaces) works as follows: If a part of the plaintext is encrypted, a cryptographic devise will return one ciphertext block for each plaintext block. There is the only one exception and this is the encryption of the last block of the plaintext. In this case one or two ciphertext blocks can be returned (depending on the length of the last block). Decryption works in the reverse order: A cryptographic device returns one plaintext block for each ciphertext block, except the decryption of the last block. After decryption of the last plaintext block, its padding is determined, cut off and the valid plaintext is returned. The characteristic of the PKCS#5 padding is that the information which part of the plaintext has to be cut off is determined from and only from the last ciphertext block. Note that this would be disrupted by Vaudenay's proposal. Based on this principal the cryptographic interfaces were built and therefore they will not work if it is violated.†

The goal of this contribution is to design an encryption for the last plaintext block, such that it respects the semantics of the widely used PKCS#5 padding (thereby preserving compatibility with the usual cryptographic interfaces) and at the same time prevents Vaudenay's attack. It is still possible that some systems do not enable the implementation of some proposed variants of encryption because of new requirements for working with key material. However, we have paid the attention to minimizing the number of such systems.

This article does not deal in any way with an alternative definition of padding. We state that our goal was to design countermeasures, which are **practically usable**. That means: *They obviously eliminate Vaudenay's attack, they do not introduce other evidently practically exploitable weaknesses and they clearly do not violate the*

---

* An edited version of the paper published as Klíma, V. and Rosa, T.: *Strengthened Encryption in the CBC Mode*, IACR ePrint archive 2002/061, May 2002.

† There can be situations where the system receives the ciphertext blocks consequently and the fact, that a block is the last block of the whole message, will be recognized just after receiving this block, not before. In Vaudenay's original proposal, after cutting the padding off the system would have to take back several plaintext blocks, which in some cases may not be technically possible.

semantics of contemporary cryptographic interfaces. The analysis of the theoretical characteristics of the proposed variants is an open question and we suggest some promising ways for a further research of the subject.

1.1 Example

According to Vaudenay's proposal, if a 7 bytes long message M is encrypted using 3DES and SHA-1, we have to encrypt 7 bytes of M, 5 bytes of the *padding* and 20 bytes of the hash value, which creates 4 blocks (32 bytes in total). The cryptographic interface would then obtain a request for an encryption of 7 bytes of M (marked as the last block) and it would return four ciphertext blocks to the calling application. However, contemporary interfaces designed according to PKCS#5 expect to receive only one ciphertext block in such a situation. Similar problems arise during decryption. It has been practically verified, that the introduction of this type of padding into the subsystem CryptoAPI makes common applications crash.

2. New Proposals for Strengthened Encryption of the Last Block in the CBC Mode

We propose three variants for strengthened encryption of the last plaintext block in the CBC mode which are compatible with contemporary cryptographic interfaces, including CryptoAPI [3]. We assume variant A as the "middle" one. Its "stronger" version is variant B and the "weaker" one is variant C. The security of these variants is estimated heuristically in a short time after the presentation of the attack, since our main intention was to quickly find a practical solution for already existing applications using the CrytpoAPI. An in depth theoretical analysis of their security remains an open problem. The variants reflect the capabilities of a designer to use different cryptographic tools.

Classical encryption of plaintext by a block cipher with key K_1 is described by the following equations: Let us denote plaintext $x_1, x_2, ..., x_N$, ciphertext $y_1, y_2, ..., y_N$, initialisation value IV, and encryption key K_1. The standard CBC encryption is then described by the following equations:

$$\text{Encryption: } y_1 = E_{K_1}(IV \oplus x_1),\ y_i = E_{K_1}(y_{i-1} \oplus x_i),\ i = 2, ..., N$$
$$\text{Decryption: } x_1 = IV \oplus D_{K_1}(y_1),\ x_i = y_{i-1} \oplus D_{K_1}(y_i),\ i = 2, ..., N$$

Strengthened encryption of the last block uses the present definition of the PKCS#5 padding. It means that x_N is the block, padded according to PKCS#5. All three variants encrypt all plaintext blocks $x_1, x_2, ..., x_{N-1}$ (excluding the last block x_N) in the CBC mode with the key K_1 in the usual way, i.e.

$$\text{Encryption: } y_1 = E_{K_1}(IV \oplus x_1),\ y_i = E_{K_1}(y_{i-1} \oplus x_i),\ i = 2, ..., N\text{-}1 \text{ and}$$
$$\text{Decryption: } x_1 = IV \oplus D_{K_1}(y_1),\ x_i = y_{i-1} \oplus D_{K_1}(y_i),\ i = 2, ..., N\text{-}1.$$

The only difference is in the equation for the encryption and decryption of the last block which we will describe now.

Using the function PBKDF2 from the standard PKCS#5 [2] and three different values of the salt (*SALT*) and three counters (*COUNT*) we derive three different values of keys K_2, K_3, and K_4:

$$K_2 = \text{PBKDF2}(K_1, SALT_1, COUNT_1, dklen),$$
$$K_3 = \text{PBKDF2}(K_1, SALT_2, COUNT_2, dklen),$$
$$K_4 = \text{PBKDF2}(K_1, SALT_3, COUNT_3, dklen),$$

where *dklen* is the length of keys K_2, K_3, and K_4, values $SALT_1$, $SALT_2$, and $SALT_3$ are different constants, and $COUNT_1$, $COUNT_2$, and $COUNT_3$ are other constants, chosen according to the standard PKCS#5.

2.1 Strengthened Encryption - Variant A

We define the equations for the last block in the following way:

$$\text{Encryption: } y_N = E_{K_4}[D_{K_2}(x_N) \oplus E_{K_3}(y_{N-1})]$$
$$\text{Decryption: } x_N = E_{K_2}[E_{K_3}(y_{N-1}) \oplus D_{K_4}(y_N)]$$

The goal of this type of encryption is that the influence of the variables (x_N, y_{N-1}, y_N), affecting encryption and decryption of the last block, is indirect, non-linear and randomized by transformations unknown to the attacker. The keys K_2, K_3, and K_4 must have these properties:

- They are derived from key K_1 by a one-way function.
- It is impossible to determine the individual keys K_2, K_3, or K_4 from the remaining two.

These additional keys are introduced in order to prevent an attacker from deducing any information about E_{K_i} and D_{K_i}, for $i \in \{2, 3, 4\}$, from eventual knowledge of the behaviour of the transformation E_{K_1} on many pairs of plaintext-ciphertext blocks. Note that it is possible to define other kinds of derivations of the keys K_2, K_3, and K_4 which are different from those defined in PKCS#5. It is only necessary to preserve the properties mentioned above.

2.2 Strengthened Encryption - Variants B1 and B2

According to the designer's capability to use the hash function, we propose variants B1 and B2. Both are designed in such a way, that the feedback from the penultimate ciphertext block is a one-way function of the variable y_{N-1}. This property is guaranteed by the value $E_{K_3}(y_{N-1}) \oplus y_{N-1}$ in the B1 variant and by the value $h(E_{K_3}(y_{N-1}))$ in the B2 variant. Derivation of the keys K_2, K_3, and K_4 stays the same as in the variant A. Note that we will use only the n most significant bits from the hash function output, where

n is the length of the block of the block cipher. The equations for the last block are defined in the following way:

Variant B1:

Encryption: $y_N = E_{K_4}[D_{K_2}(x_N) \oplus E_{K_3}(y_{N-1}) \oplus y_{N-1}]$

Decryption: $x_N = E_{K_2}[E_{K_3}(y_{N-1}) \oplus y_{N-1} \oplus D_{K_4}(y_N)]$

Variant B2:

Encryption: $y_N = E_{K_4}[D_{K_2}(x_N) \oplus h(E_{K_3}(y_{N-1}))]$

Decryption: $x_N = E_{K_2}[h(E_{K_3}(y_{N-1})) \oplus D_{K_4}(y_N)]$

2.3 Strengthened Encryption - Variant C

This variant is proposed as the "minimal" variant for the case where the designer does not have the possibility of using other transformations than E_{K_1} and D_{K_1}, i.e. she has no possibility to derive new keys from key K_1 and she has no possibility to use a hash function. The equation for the last block is defined in the following way:

Encryption: $y_N = E_{K_1}[D_{K_1}(x_N) \oplus E_{K_1}(y_{N-1}) \oplus y_{N-1}]$

Decryption: $x_N = E_{K_1}[E_{K_1}(y_{N-1}) \oplus y_{N-1} \oplus D_{K_1}(y_N)]$

3. Heuristic Analysis

From the general attack point of view, we can assume all presented variants as an application of the two modes. Blocks $x_1, x_2, ..., x_{N-1}$ are encrypted using the first mode and the block x_N is encrypted using the second mode. There is no change in the case of encryption of the blocks $x_1, x_2, ..., x_{N-1}$ - it is the original CBC mode. Therefore the proposed variants do not impose new weaknesses here. The encryption of the last block can be assumed also as the CBC mode, the initialisation value of which is derived pseudorandomly from the penultimate ciphertext block y_{N-1}. Note that in this way, the dependency on the original IV is also preserved.

From the point of view of defence against Vaudenay's attack, it is natural to use a generalized notion of the confirmation oracle [4], which is a useful tool in the study of side channels. We have used it for the fault attacks on RSA-KEM in chapter C, §4 in this thesis. In the case of the CBC mode, the confirmation oracle has the form of a decryption engine, which the attacker sends chosen ciphertexts to. The engine accepts or refuses the given ciphertext according to whether the last plaintext block has the correct padding or not. We assume that an attacker has the possibility of obtaining information about the acceptance or refusal of the last block. Therefore she has an access to a confirmation oracle, which allows her to confirm whether the last block of the decrypted plaintext has correct padding or not.

Let us denote ***PAD*** the set of allowed paddings according to PKCS#5. In the case of the classical CBC mode with the PKCS#5 padding, it holds $x_N = D_{K_1}(y_N) \oplus y_{N-1}$, $x_N \in \boldsymbol{PAD}$. Using the confirmation oracle it is possible to confirm the validity of this

relation for an arbitrarily chosen y_N and y_{N-1}. With respect to the definition of the set ***PAD*** and with respect to the way in which the value y_{N-1} enters the expression, the transformation E_{K_1} can be easily inverted using the confirmation oracle. That is exactly what Vaudenay has actually shown in his article [1].

Now, let us recall the relations (note that the statement $x_N \in$ ***PAD*** is their crucial part), which can be confirmed in our variants of a strengthened encryption.

A) $x_N = E_{K_2}[E_{K_3}(y_{N-1}) \oplus D_{K_4}(y_N)]$, $x_N \in$ ***PAD***

B1) $x_N = E_{K_2}[E_{K_3}(y_{N-1}) \oplus y_{N-1} \oplus D_{K_4}(y_N)]$, $x_N \in$ ***PAD***

B2) $x_N = E_{K_2}[h(E_{K_3}(y_{N-1})) \oplus D_{K_4}(y_N)]$, $x_N \in$ ***PAD***

C) $x_N = E_{K_1}[E_{K_1}(y_{N-1}) \oplus y_{N-1} \oplus D_{K_1}(y_N)]$, $x_N \in$ ***PAD***

The influence of y_{N-1} on $x_N = D_{K_1}(y_N) \oplus y_{N-1}$ is in the original CBC mode "direct and non-masked". In particular, we mean that it is fully deterministic for an attacker and linear. These are the essential conditions which make the confirmation oracle useful here. In the proposed variants, the variables y_{N-1} and y_N always act indirectly and via non-linear transformations, unknown to an attacker. Thus, from the confirmation oracle, the attacker could obtain only information about a relation among unknown images of input variables. Moreover, except for variant A, the input variable y_{N-1} goes through a one-way function. This prevents the attacker preparing special values for a test in the case, where she has partial knowledge about the transformation $E_{K_3}(y_{N-1})$ or $E_{K_1}(y_{N-1})$. In this way defence against Vaudenay's attack is practically ensured.

4. Conclusion

Vaudenay has described a practical attack on the CBC mode based on a fault side channel. The correction, which Vaudenay proposed has a general character and doesn't solve practical problems with the real cryptographic interfaces used in contemporary applications. In this contribution, we have presented practical countermeasures which are semantically compatible with current cryptographic interfaces. On the basis of the aforesaid heuristic analysis, we presume that the proposed variants are not vulnerable to attacks of the Vaudenay type. Their theoretical security, however, is an open research problem. We suggest considering and implementing them in the order B2, B1, A, C.

References

1. Vaudenay, S.: *Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS, …* , In Proc. of Eurocrypt 2002, pp. 534 - 545
2. PKCS#5 v2.0: *Password-Based Cryptography Standard*, RSA Laboratories, March 25, 1999
3. Microsoft: *MSDN Library - July 2001*, Platform SDK Documentation, Security, Cryptography, 2001
4. Klíma, V. and Rosa, T.: *Further Results and Considerations on Side Channel Attacks on RSA*, In proc. of CHES 2002, pp. 245-260 ,Springer-Verlag, 2003

E. Side Channel Attacks on CBC Encrypted Messages in the PKCS#7 Format*

1. Introduction

Vaudenay's attack [13] has been further studied in [2], where several extensions of and countermeasures were proposed. The only practically effective padding types defined there were referred to as the ABYT-PAD (arbitrary-tail padding) for byte-oriented messages and the ABIT-PAD for bit-oriented messages. The A**BYT**-PAD (A**BIT**-PAD) was defined in the following way: Let the last byte (bit) of the message be *X*. Pick an arbitrary distinct byte (bit) *Y* and add one or more bytes (bits) *Y* as needed to the end of the message. The receiver reads the last byte (bit) of the plaintext and removes all successive bytes (bits) which are the same as *Y* from the end of the plaintext. The main benefit of this padding is that there is no incorrectly padded plaintext. Therefore, it is no longer possible to use Vaudenay's attack based on a valid-padding oracle, because such an oracle doesn't tell us any new information (its output has a zero entropy overall). However, even when using these methods, there are a lot of other vulnerabilities and possible attacks. In [2], it was underlined that such attacks are pervasive when the integrity of ciphertexts is not guaranteed. As an example, the authors of [2] designed a so-called "cryptographic relay" (a device), which consists of two cryptographic schemes. The first one uses the robust padding scheme described above, while the second one uses a length-preserving scheme (e.g. CTR mode). The device decrypts the ciphertext coming from the first scheme and then re-encrypts it using the second scheme. Because the second scheme does not hide the original plaintext length, it is possible to use this information for an attack on the first scheme. In this paper, we show that it is not necessary to design such an abstract scheme to carry out a successful attack. We simply combine the well-known cryptographic message syntax standard PKCS#7 [3] with the use of the ABYT-PAD padding scheme instead of the PKCS#5 padding.

Let us assume that we have access to a confirmation oracle $PKCS\#7_{CONF}$ which tells us for a given ciphertext (encapsulated in the PKCS#7 structure) whether the decrypted plaintext is correct or not according to the PKCS#7 syntax. This is probably a very natural assumption, because applications usually have to reflect this situation in their behavior. It could be a message for the user, an API error message, an entry in the log file, different timing behavior, etc. We show that by having access to such an oracle an attacker can invert the underlying block cipher for a particular arbitrary key, thereby deciphering the secret encrypted message. This attack requires a single captured ciphertext belonging to the key and approximately 128 oracle calls per one

* An edited version of the paper: Klíma, V. and Rosa, T.: *Side Channel Attacks on CBC Encrypted Messages in the PKCS#7 Format*, In Proc. of 2nd International Scientific Conference: Security and Protection of Information, pp. 75-83, NATO PfP/PWP, Brno, Czech Republic, 28. - 30.4. 2003.

ciphertext byte. This kind of attack can be extended to other padding schemes (i.e. ABIT-PAD, etc.). Surprisingly, our attack is allowed by those PKCS#7 (v1.6) properties that are planned to improve version v1.5. According to the new version (v1.6) there are not only data-octets (bytes) encrypted (as in the previous version), but it also encrypts the length-octets and type-octets. The main idea of the attack is to carefully combine the changes at the beginning and the end of the encrypted message. Then we use the PKCS#7-confirmation oracle which tells us whether the change was correct or not in the sense of the PKCS#7 v1.6 format. This information thwarts the original good property of the ABYT-PAD scheme that all deciphered plaintexts are valid. The "improvement" of version 1.5 of the standard thus brought a new kind of attack. It follows that an improvement that is good under a local estimation may turn out to be a bad choice under a broader context evaluation. On the other hand, we do not express the opinion that our attack is a problem of the step of moving from the PKCS#7-v1.5 to the PKCS#7-v1.6 standard. The conclusion of our paper is that, just like the area of asymmetrical cryptography, we cannot hope to fully solve these problems with side channel attacks just by using a "magic" padding method or an obscure message-encoding format.

The rest of the paper is organized as follows; firstly we introduce the necessary notation and description of the PKCS#7 format (§2) and the confirmation oracle $PKCS\#7_{CONF}$ (§3). An attack is then presented in §4. In §5 we summarize the complexity of the attack and its extensions. Countermeasures are presented in section §6 and a conclusion is made in §7. The notion of confirmation oracle was for the first time successfully applied in the area of side channel attacks on asymmetrical cryptography in [7] (c.f. chapter C, §4 in this thesis). We presumed there that this notion is general enough that it could be extended for other schemes and algorithms as well. Its successful application in the area of symmetrical encryption, demonstrated in this chapter, proves that hypothesis.

2. Preliminaries

2.1 Notation

We will denote *CT* the ciphertext *C* without an initializing value (*IV*), thus $C = (IV, CT)$. We will assume the block cipher, which works over *n*-byte blocks, where *n* is a positive integer. We will denote $E_K(B)$ and $D_K(B)$ enciphering and deciphering of a data block *B* under a secret (symmetric) key *K*. We will denote $ENC\text{-}CBC_K$ and $DEC\text{-}CBC_K$ enciphering and deciphering of the whole plaintext and ciphertext in the CBC mode, respectively. To be consistent with the ASN.1 notation, we will talk about octets with the assumption that the term octet means the same as byte in this study. We will use "BIG ENDIAN" ordering of bytes inside the data block, i.e. b_1 will be the most significant byte in the *n*-byte block $B = (b_1, ..., b_n)$. Hexadecimal numbers are denoted using the prefix “0x”, the exclusive OR operation is denoted ⊕. We will use it for bits, bytes and blocks of bytes. If the blocks are indexed, we use the second

index to pick the byte from it, for instance $CT_{s,n}$ is the last (n-th) byte of the block CT_s. Under such circumstances, indexes start from 1. Note that $C_1 = IV$ and $CT_1 = C_2$ is the first "payload" block corresponding to the plaintext P_1 in our notation.

2.2 PKCS#7 Data Structures

2.2.1 PKCS#7

As stated in [11], the standard PKCS#7 describes the general syntax for cryptographically protected data, e.g. data which is encrypted, digitally signed, etc. Data syntax is described using the abstract syntax notation ASN.1 [4]. It admits recursion, so that one envelope (c.f. [11]) can be nested inside another. The values produced according to this standard are intended to be BER-encoded [5], which means that the values would typically be represented as octet strings. The syntax is general enough to support many different content types. PKCS#7 defines the six following ones: data, signed data, enveloped data, signed-and-enveloped data, digested data, and encrypted data. These content types are defined using the notation ASN.1 and they are used in a number of applications, programs, protocols etc. For instance, we may take the banking protocol SET or the standard for a secure electronic mail S/MIME. We will concentrate on the content type "enveloped data". It contains the data (a binary content) encrypted by the symmetric encryption key, where this key is transmitted using a public-key algorithm. We will assume that the data is as usually encrypted by a block cipher in the CBC mode.

2.2.2 ASN.1 Encoding

Before its encryption, the data being encrypted is ASN.1 encoded first, usually by using the BER/DER encoding [5]. In most cases, encoding consists of adding some type-octets together with some length-octets before the data itself. The type-octets define the type of data (type of data structures) and the length-octets define the length of the data. This length means the length of the original data which follows after the length-octets. The triplet (*type-octets*, *length-octets*, *data-octets*) is then padded and encrypted in the CBC mode.

2.2.3 Data Types

There are a lot of data types that can be used in various applications for data being encrypted. These data types are usually publicly defined and their octet codes are thus well known for a concrete application. Probably, the most often used one is the data type OCTET STRING, encoded as one octet with the value 0x04. Without loss of generality, we may assume that the data type is OCTET STRING for the purpose of our side channel attack description.

2.2.4 Data Length

Similarly, the length of the original data being encrypted is encoded into one or more length-octets. If the data length is less than 128, then there is only one length-octet the value of which states exactly the data length. If the data length is higher than or equal

to 128, then it is encoded as a 2 to 128 octets long string of the length-octets. In this case the first octet has the most significant bit set to 1 and its remaining bits express the number of the following length-octets. The remaining length-octets then express the data length in the integer base 256. For instance, if the length of the original data is less than 64 KBytes, then the first length-octet is 0x82 and the following successive two octets give the particular length in the base 256. The number of length-octets doesn't play any important role in our side channel attack. For the sake of simplicity, we will assume that the data being encrypted has only one length-octet.

2.3 Encryption in the PKCS#7 Version 1.5 and 1.6

Let us have *L* bytes of data. We will assume its ASN.1 encoding as (0x04, *L*, *data*). Version 1.5 of PKCS#7 [11] was designed to enable PEM compatible formats, but this brought some inconveniences for applications. Since certain parts of the encoded image of the data were not encrypted (e.g. type-octets and length-octets remained in an open form), it required applications to "dip under" the ASN.1 and deal directly with the BER/DER encoding of data (signing, encryption). Such BER/DER "hacking" made it difficult for users of ASN.1 compilers to generate encoding/decoding subroutines, because the head of the data and the data itself were processed separately. Having accepted the ascendancy of S/MIME over PEM, and the desirability of avoiding low-level "hacking" of the BER/DER encoding, version 1.6 of PKCS#7 [3] modified the processing rules to operate on the entire BER/DER encoding of the data. In particular, this means that data-octets are encrypted together with the type-octets and length-octets as one binary stream (type-octets, length-octets, data-octets). It is anticipated that version 2 of PKCS#7 will also incorporate this change.

2.4 ABYT-PAD Padding Scheme

Before the encryption, we need to append an appropriate padding. Finally, we then encrypt the quadruple (0x04, *L*, *data*, *padding*) in the CBC mode. Here we assume that the padding scheme used is ABYT-PAD [2], however the attack can be easily extended to some other schemes (for instance ABIT-PAD). According to [2], ABYT-PAD is defined in the following way.

Let the last byte of the plaintext be *X* and pick an arbitrary distinct byte *Y*. We add one or more bytes of *Y* as needed to the end of the plaintext in such a way that the new plaintext length is an integer multiple of *n*. We emphasize that at least one byte of *Y* must be appended. In the case of an empty plaintext, *Y* can be an arbitrary value. If the padding is not more than *n* octets long, we talk about a "short" ABYT-PAD padding. In the case of an unlimited padding length, we will talk about a "long" ABYT-PAD padding. The receiver reads the last byte of the plaintext and removes all the successive identical bytes from the end of the plaintext. In the following text, we will assume that the padding is the "short" one, i.e. the number of padding bytes has to be from 1 to *n*. However, it will become clear that it is easily possible to extend the

attack to the long ABYT-PAD padding. We also assume that the symmetric key and the underlying block cipher are always the same during our attack.

3. Confirmation Oracle $PKCS\#7_{CONF}$

Assume that a sender encrypts messages using the PKCS#7 data type "enveloped data", a block cipher in the CBC mode, and the ABYT-PAD padding scheme. The encrypted data is ASN.1 encoded, padded, and encrypted using a random symmetric key, and then the ciphertext *CT* is put in a specific place in the highly structured data block "enveloped data" [11]. The symmetric key is then encrypted using a public-key scheme and is also put in its specific place in the "enveloped data". The initialization value (*IV*) is saved in this structure, outside the ciphertext *CT* (in data type "Content Encryption Algorithm Identifier"), as well. In particular, we note that it is possible to change *IV* without disturbing any other content of the "enveloped data". Furthermore, we assume that when the attacker changes the length and the content of the ciphertext *CT* later on, she will also eventually change the appropriate length octets of all "higher" structures containing it. Thus, the changed ciphertext *CT* will be correctly encapsulated within the PKCS#7 structure. In the following, we will focus only on the receiver's dealing with the *IV* and *CT* items in the block "enveloped data".

Definition (PKCS#7 confirmation oracle $PKCS\#7_{CONF}(C)$). *Let us have the ciphertext C = (IV, CT). We propose a* $PKCS\#7_{CONF}$ *confirmation oracle,* $PKCS\#7_{CONF}(C)$*: C → (ANSWER = "OK/BAD"), encapsulating the following procedure at the receiver's side:*

1. $P = DEC\text{-}CBC_K(C)$*; the plaintext P is obtained by deciphering the ciphertext C in the CBC mode under the symmetric key K*
2. *Remove the padding from the plaintext P; the resulting message is denoted as M.*
3. *Parse M according to PKCS#7:*
   - *Check the type-octets of M; according to the assumptions in §2, we expect one concrete value to be here - 0x04. If it is not here, an error has occurred.*
   - *Check the length-octets of M; we expect one length octet to be here (L), furthermore, L must be equal to the length of M, obtained in step 2. If it is not, an error has occurred.*
4. *If the two previous checks in step 3 are successful, the answer of* $PKCS\#7_{CONF}(C)$ *is "OK"; otherwise it is "BAD".*

We note that no error messages are expected to occur in steps 1 and 2. For the sake of convenience, we will use the symbols *Orac* and $PKCS\#7_{CONF}$ when referring to the $PKCS\#7_{CONF}$ confirmation oracle interchangeably.

4. Attack Description

Let the attacker intercept a valid ciphertext $C = (IV, CT_1, CT_2, ..., CT_s)$, $s \geq 1$, and let $(P_1, P_2, ..., P_s)$ denote the corresponding plaintext. We will show that using a $PKCS\#7_{CONF}$ she can then compute $X = D_K(Y)$ for any arbitrary chosen Y, obviously implying that she can decipher the whole intercepted ciphertext C.

Recall that we are working with short messages (one length-octet) with the short ABYT-PAD padding. However, we will show in §5 how to modify the attack for longer messages and longer ABYT-PAD padding.

The attack has several steps. The first step is to be carried out only once, in the preparation phase. For simplicity, we assume that we have only one intercepted ciphertext C. If we had more ciphertexts, the preparation phase could be easier.

4.1 Preparation Phase: Finding the Length *L* of the Message

Since C is a valid ciphertext, the corresponding message M (the plaintext without padding) conforms to PKCS#7. Thus, we have $P_{1,1} = 0x04$ and $P_{1,2} = L$, where L is the length of M. In this step, we determine the value of L using a $PKCS\#7_{CONF}$ oracle.

Let $s \geq 3$ (see the remarks at the end of this paragraph for managing the attack for $s < 3$). This condition guarantees that changes made in CT_{s-1} will not affect the first plaintext block containing the length octet. Let us denote *LPAD* the length of padding and *LDATA* the length of remaining data bytes in the last plaintext block, i.e. $LDATA + LPAD = n$. At first, we will successively test every byte from the end of the last plaintext block whether it is a padding byte or a data byte. When we find the first data byte from the end, we have the value *LDATA* and we stop testing. According to our assumptions, $0 \leq LDATA \leq n - 1$, because at least one byte ($P_{s,n}$) is a padding byte according to ABYT-PAD padding. Therefore we begin our test with $P_{s,n-1}$ following the pseudocode written below. We will denote as $C' = (IV', CT_1', CT_2', ..., CT_s')$ our changes in the original ciphertext C. We assume that, during initialization, the following is set: $IV' = IV$ and $CT_i' = CT_i$.

```
Set LDATA = 0

For j = (n - 1) downto 1

{

   Set CT'_{s-1,j} = CT_{s-1,j} ⊕ 1

   If Orac(C') = "OK" then set LDATA = j, break

   /* The change will result in the corruption of the
   whole block P_{s-1} and of the byte P_{s,j}.

   If Orac(C') returns "OK", then the change of the
   original plaintext byte P_{s,j} to the value P_{s,j} ⊕ 1
```

```
    didn't affect the length L. Therefore P_{s,j} is already
    the last data byte and we have LDATA = j.

    If Orac(C') returns "BAD", the length L doesn't
    conform to the padding. There are two possibilities:

            i) P_{s,j} was the last data byte, but it has
               been unwittingly changed to the padding
               byte

            ii) P_{s,j} was the padding byte, which has been
               changed to a non-padding byte

    We can, however, decide between these two
    possibilities by setting CT'_{s-1,j} = CT_{s-1,j} ⊕ 2 and
    calling the oracle again.*/

    Set CT'_{s-1,j} = CT_{s-1,j} ⊕ 2 and call Orac(C')

    If Orac(C') = "OK" then set LDATA = j, break

    /* If the oracle returns "OK", then P_{s,j} was a data
    byte. If it returns "BAD", P_{s,j} was a padding byte.
    In this case we continue to test the next bytes on
    the left.*/

}

Set L = (n - 2) + (s - 2)*n + LDATA

/* Note that (n-2) bytes are counted from the first
block and LDATA bytes come from the last block. The
resting (s-2) blocks have the full length n.*/
```

Remarks

- There is the possibility of further optimising this process by various methods, for instance by the interval halving method, similarly as in [2]. In this case, we would need $\boldsymbol{O}(\log_2 n)$ oracle calls. On the other hand, it is only a marginal improvement, because this step is carried out only once in the preparation phase.
- When the intercepted ciphertext has only one block CT_1, i.e. $s = 1$, we can use the same process as in case $s \geq 3$, because we will have full control over changes in the first plaintext block. We only have to operate with the field IV instead of CT_{s-1}.
- In the case $s = 2$, we can artificially lengthen the original ciphertext (IV, CT_1, CT_2), for instance, as (IV, CT_1, CT_2, IV, IV, CT_1, CT_2). It is only necessary to change the second byte of the primary IV appropriately, because we added $4*n$ bytes to the message M. For instance if $n = 8$, we artificially added 32 bytes to it and we can fix

the length easily by xoring the byte $P_{1,2}$ with 0x20, because the length of the original message M was less than 14, therefore $P_{1,2} \oplus 0x20 = P_{1,2} + 0x20$. Note that, since the mode of CBC is used, changes on P_1 are done easily by manipulating with the value of IV. Now, we can follow the process above for $s \geq 3$.

Now we have the plaintext byte $P_{1,2} = L$ of the ciphertext C. The complexity of this step is maximally 2*(n - 1) oracle calls.

4.2 Computing $X = D_K(Y)$, Leaving One Byte of Uncertainty

Now, we use the first two blocks (IV, CT_1) of the intercepted ciphertext and create a new one $C = (IV, CT_1, S, T, Y)$, where S and T are arbitrarily chosen and Y is the block which is to be decrypted. Let us denote $P = (P_1, P_2, P_3, P_4)$ as the plaintext corresponding to the ciphertext C. From the foregoing steps, we have $P_{1,1} = 0x04$ and $P_{1,2} = L$, where L is known from the previous step. Using the following pseudocode, we determine $X = D_K(Y)$, leaving one byte of uncertainty. As above (in §4.1), changes on C will be denoted as C'.

```
Let A = X_n ⊕ T_n = X_n ⊕ T'_n /* Note that T_n doesn't
change during the computation, so T_n = T'_n.*/

For i = (n - 1) downto 1

{

   /* In this loop, we derive the i-th byte P_{4,i}, where
   P_{4,i+1} =...= P_{4,n} are padding bytes, all equal to A.*/

   Set N = (n - 2) + n + n + (i - 1)

   Set IV' = IV ⊕ P_{1,2} ⊕ N

   /* After deciphering C', the oracle gets the number
   N in the place of P'_{1,2}. Thus it will expect i - 1
   data bytes and n - (i - 1) padding bytes in the last
   plaintext block P_4.*/

   (#) For j = 0 to 255 do

   {

      Set T'_i = T_i ⊕ j

      If Orac(C') = "OK" go to (##)
```

```
      /* If Orac(C') = "OK", the plaintext is PKCS#7
      conforming and X_i ⊕ T_i equals to the padding byte
      A. Thus, we have X_i ⊕ T_i = X_{i+1} ⊕ T_{i+1} =...= X_n ⊕ T_n
      = A and we can continue to derive the next byte.*/

   }

   If (i > 1) then set T'_{i-1} = T_{i-1} ⊕ 1
              else set S'_n = S_n ⊕ 1

   Go to (#)

   /* If the oracle has always responded "BAD" in the
   preceding cycle, it means that when the correct
   value (A) occurred on the i-th byte, it accidentally
   also occurred on its left side. Therefore, we change
   the left byte and go back to (#). Now, the oracle
   must once respond "OK".*/

(##) /* Continue to the next loop.*/

}
```

Let us set $T = T'$ at the end of the procedure. We have

$$X_1 \oplus T_1 = X_2 \oplus T_2 = \ldots = X_n \oplus T_n = A, \quad (4.2)$$

where the values of T_i (and eventually also S_n) have been adjusted above. Note that for $n > 32$ we will need more length-octets, so it would be necessary to slightly modify this procedure. However, for the most of contemporary block ciphers, we have $n \leq 32$, therefore this technical modification will be omitted here.

In this step, we need circa 128*(n - 1) oracle calls on average. According to the procedure written above, the maximum number of oracle calls is clearly limited by the number 512*(n - 1).

4.3 Determining the Remaining Byte of Uncertainty

Now, we use the blocks T and Y created in the previous step. The ciphertext $C = (T, Y)$ gives the one plaintext block consisting of n bytes having the same value A. We will now change T_1, T_2 and T_n to obtain a PKCS#7 conforming message. We construct the message in such a way to have the length of n - 3 octets and one padding byte. We then determine the value of A in the following way. Recall that changes on C will be denoted as C'.

```
For j = 0 to 255 do

{

   Set T'1 = T1 ⊕  0x04 ⊕ j

   Set T'2 = T2 ⊕ (n - 3) ⊕ j

   Set T'n = Tn ⊕ 1

   C' = (T', Y)

   /* Note that C' decrypts to the plaintext bytes (A ⊕
   0x04 ⊕ j, A ⊕ (n-3) ⊕ j, A,..., A, A ⊕ 1). */

   If Orac(C') = "OK" then A = j, break

   /* If Orac(C') = "OK", the plaintext is PKCS#7
   conforming. Thus, we have A ⊕ 0x04 ⊕ j = 0x04. We
   then easily obtain the unknown A as A = j. */

}
```

Now, we use the value T and substitute it with A into the system of equations (4.2), thereby deriving the value of X. This step requires circa 128 oracle calls on average. It takes maximally 256 oracle calls.

5. Complexity of the Attack and Its Extensions

Recall that the complexity of the attack is at most $2*(n - 1)$ calls in the preparation phase. This phase is carried out only once for a particular symmetric key. To decipher each ciphertext block, we then need circa $128*(n - 1) + 128 = 128*n$ oracle calls on average. The maximum number of oracle calls per ciphertext block is bounded above by $512*(n - 1) + 256$ and obviously, the whole attack has a linear complexity $\boldsymbol{O}(n)$, which can be regarded as trivially feasible attack from the cryptanalytic viewpoint.

In the following text, we summarize our remarks on possible extensions and modifications of the attack.

- In most cases of longer messages with more than one length-octet, we can easily derive the plaintext byte $P_{1,2} = L$ in the preparation phase. In these cases $P_{1,2}$ is the first length-octet and thus $P_{1,2}$ is equal to the number of remaining length-octets + 0x80 (c.f. §2.2.4). We can estimate the number of remaining length-octets directly from the length of the ciphertext.
- Generally speaking, when longer messages (with any kind of ABYT-PAD padding) with more than one length-octet are used, we can successively change the third, fourth, etc. byte of the *IV* and send the ciphertext to the

oracle. If we change the length octet, the oracle returns "BAD". If it returns "OK", we hit the first data octet. Now, we have the number of length octets (W) and we get $P_{1,2} = 0x80 + W$ immediately.

- A variant of this attack can be also derived for the ABIT-PAD padding. Since the only difference between ABYT-PAD and ABIT-PAD is the size of the elementary block unit, the derivation is a matter of changing the byte-oriented approach for a bit-oriented one. Note that the PKCS#7 format discussed here is byte-oriented in its nature, therefore even when using ABIT-PAD, we pad bytes. However, we may expect that the PKCS#7$_{CONF}$ based on ABIT-PAD would, in a certain way, allow an attacker to do the inversion of $E_K(B)$ bit-by-bit instead of byte-by-byte. Such an approach generally helps the attacker to improve the effectiveness of her attack. Such an improvement would be useful in case of extremely long padding string.

6. Countermeasures

The attack presented here is based on the behaviour of a typical transport-layer application, which routinely receives a ciphertext, deciphers it, and decodes its data payload, which is then passed to the upper layers. If an error occurs during this processing (e.g. ASN.1 parser fails, the padding is incorrect, etc.), it is natural that the application informs its communicating peer. Ignoring these errors is theoretically possible, but in practice it wouldn't make a lot of sense. Moreover, such a failure would be probably detectable from the behaviour of the upper-layer application.

We emphasize that the attack discussed here is not only a particular problem for padding methods. Generally speaking, such an attack can be expected whenever the following conditions are fulfilled:

(i) there are some formatting rules set for plaintexts which must be checked,

(ii) an attacker can freely modify captured ciphertexts and re-send them to the communicating application,

(iii) the changes made in (ii) induce predictable changes of the corresponding plaintext.

The combination of the CBC mode with the PKCS#5 padding scheme was perhaps the most obvious way in which the conditions given above were fulfilled. The only thing that is still surprising is the amount of time it took for the cryptanalysts to disclose this weakness. To avoid other possible “surprises”, we should constantly verify these conditions when designing new encryption schemes. In this study, we addressed the situation where the first condition seems to be thwarted, since the padding method ABYT-PAD does not impose any checkable rules. However, the condition is easily restored if we incorporate the formatting rules given by the PKCS#7 standard. It clearly follows that despite being very tempting, we cannot hope to solve problems of attacks addressed here and in [2] and [13] by thwarting only the first condition written above.

A better way seems to be to focus on conditions (ii) and (iii). We can use the paper of Krawczyk [8] to conjecture that the best way is generally to thwart the second condition by using the authentication of ciphertexts. This countermeasure was also generally recommended in [2]. However, in some older applications it might not always be easy to introduce such a modification. Therefore, we looked at thwarting the last conditions. In [6] (c.f. chapter D of the thesis), we designed a method that effectively prevents an attacker from making predictable changes of the plaintext by changing the ciphertext. Our method then enables any padding method that is limited to the last block to be used. The main idea of our approach is encrypting the last block in a different way and under a different key. This so-called strengthened encryption changes the definition of the encryption and decryption process, but it is still compatible with the original CBC encryption from the point of view of data structures and their length. From a practical viewpoint, we conjecture that for existing applications and protocols it is better to change the program codes or data semantic rather than the data structure itself. We must emphasize that our method works well, unless there are other structural checks of plaintexts. Therefore, generally we strongly recommend adding a cryptographic check of ciphertexts in the sense of [8].

7. Conclusions

In this chapter of the thesis, we have shown that we cannot hope to fully solve problems with side channel attacks on the CBC encryption mode by using a "magic" padding method or an obscure message-encoding format. Vaudenay showed in [13] that the CBC encryption mode ([9], [1]) combined with the PKCS#5 padding scheme [10] allows an attacker to invert the underlying block cipher, provided she has an access to an oracle which for each input ciphertext states whether the corresponding plaintext has a valid padding or not. Countermeasures against this attack using different padding schemes were studied in [2] and the best method was referred to as the ABYT-PAD.

In this paper, we combine the well-known cryptographic message syntax standard PKCS#7 [3] with the use of ABYT-PAD in the place of the PKCS#5 padding scheme. We assume that the attacker has access to an oracle $\text{PKCS\#7}_{\text{CONF}}$ which tells her for a given ciphertext (encapsulated in the PKCS#7 structure) whether the deciphered plaintext is correct or not according to the PKCS#7 (v1.6) syntax. This is a very natural and straightforward assumption, because applications usually have to reflect this situation in their behaviour. It could be a message for the user, an API error message, an entry in the log file, different timing behaviour, etc. We have shown that having an access to such an oracle enables the attacker to invert the underlying block cipher and decipher the encrypted message. It requires a single ciphertext for a particular key and approximately 128 oracle calls per ciphertext byte.

Surprisingly, the attack is allowed by those PKCS#7 (v1.6) properties that are designed to improve version v1.5. They are also planned for version 2. However, the improvement of the standard brought a new kind of attack. It follows that an improvement beneficial according to a local estimation may turn out to be a bad choice under a broader context evaluation. On the other hand, we do not express the

opinion that our attack is a problem of the step of moving from the PKCS#7 v1.5 to the PKCS#7 v1.6 standard.

The attack described here can be also easily extended on other TLV-like schemes. The TLV stands for tag-length-value, which is a common nickname of many data protocols and formats used nowadays. Since TLV involves also many standards used in the banking sector, it indicates that existing systems in such areas deserve certain amount of attention according to the attack presented here.

The discussed problems with side channel attacks on the CBC encryption mode should be solved using strong cryptographic integrity checks of ciphertexts. Our contribution should be regarded as further evidence that these checks must be included in the new cryptographic standards and protocols.

References

1. Baldwin, R. and Rivest, R.: *RFC 2268 - The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms*, October 1996
2. Black, J. and Urtubia, H.: *Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption*, In Proc. of 11th USENIX Security Symposium, San Francisco 2002, pp. 327-338
3. *Extensions and Revisions to PKCS #7 (Draft PKCS #7 v1.6)*, An RSA Laboratories Technical Note, May 13, 1997
4. *ITU-T Recommendation X.680 (1997)*, ISO/IEC 8824-1:1998, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation
5. *ITU-T Recommendation X.690 (1997)*, ISO/IEC 8825-1:1998, Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
6. Klíma, V. and Rosa, T.: *Strengthened encryption in the CBC mode*, Cryptology ePrint Archive: Report 2002/061, http://eprint.iacr.org/2002/061.pdf
7. Klíma, V. and Rosa, T.: *Further Results and Considerations on Side Channel Attacks on RSA*, In proc. of CHES 2002, San Francisco Bay, USA, August 2002, pp. 245-260, Springer-Verlag, 2003
8. Krawczyk, H.: *The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)*, CRYPTO' 01, pp. 310 - 331, Springer-Verlag, 2001
9. NIST Special Publication: *SP 800-38A 2001 ED - Recommendation for Block Cipher Modes of Operation*, December 2001
10. *PKCS#5 v2.0: Password-Based Cryptography Standard*, RSA Laboratories, March 25, 1999
11. *PKCS #7 v1.5: Cryptographic Message Syntax Standard*, RSA Laboratories, November 1, 1993
12. Rosa, T.: *Future Cryptography: Standards are not Enough*, in Proc. of Security and Protection of Information, NATO-IDET, 2001, Military Academy in Brno, pp. 237 - 245, Brno, 2001
13. Vaudenay, S.: *Security Flaws Induced By CBC Padding - Application to SSL, IPSEC, WTLS...*, EUROCRYPT '02, pp. 534-545, Springer-Verlag, 2002

F. Attacking RSA-based Sessions in SSL/TLS*

1. Introduction

In contemporary cryptography, it is widely agreed that one of the most important issues of all asymmetric schemes is the way in which the scheme encodes the data to be processed. In the case of RSA [14], the most widely used encoding methods are described in PKCS#1 [9]. This standard also underlies RSA-based sessions in the family of SSL/TLS protocols. These protocols became de facto the standard platform for secure communication in the Internet environment. In this part of the thesis, we assume certain familiarity with their architecture (c.f. §5). Since its complete description is far beyond the scope of this article, we refer interested readers to the excellent book [10] for further details. In 1998 Bleichenbacher showed that the concrete encoding method called EME-PKCS1-v1_5, which is also employed in the SSL/TLS protocols, is highly vulnerable to chosen ciphertext attacks [1]. The attack assumes that information about the course of the decoding process is leaking to an attacker. We refer to such attacks as *side channel attacks*, since they rely on *side information* that unintentionally leaks out from a cryptographic module during its common activity (c.f. chapter A of this thesis for introduction to the theory of side channels).

Bleichenbacher showed that it is highly probable that side information exists allowing the attacker to break the particular realization of the RSA scheme in many systems based on EME-PKCS1-v1_5. He has also shown how to use such information to decrypt any captured ciphertext or to sign any arbitrary message by using a common interaction with the attacked cryptographic module. As a countermeasure to his attack it was recommended to either use the EME-OAEP method (also defined in PKCS#1) or to steer attackers away from knowing details about the course of the decoding process. In the case of the SSL/TLS protocols it seemed to be possible to incorporate the second type of countermeasures. The story of the attack ended here by incorporating appropriate warnings in appropriate standards [9], [10], [12], and [15]. Security architects were especially instructed not to allow an attacker to know whether the plaintext P being decoded has the prescribed mandatory structure marks or not.

Besides being warned to carry out the above-mentioned countermeasure, architects were also instructed to carefully verify all possible marks of P that are specific for the SSL/TLS protocols. In particular, they were told to check the correctness of a version number (c.f. §5.2 and [12]), which is stored in the two left-most bytes of the *premaster-secret*. Unfortunately, it has not been properly specified how such a test may be combined with the countermeasure mentioned above and what to do if the

* An extended version of the paper: Klíma, V., Pokorný, O., and Rosa, T.: *Attacking RSA-based Sessions in SSL/TLS*, in Proc. of CHES '03, Cologne, Germany, September 7-11, pp. 426-440, Springer-Verlag, 2003.

"version number test" fails. Designers may be very tempted to simply issue an error message. In reality, however, such a message opened up a Pandora's box bringing a new variant of side channel attack. We present this attack and discuss its implementation details. It turns out that the version number, which was initially believed to rule out the original attack [1], even allows a relatively optimized variant of the attack if the version number check is badly implemented. Our practical tests showed that among hundreds of SSL/TLS servers randomly chosen from the Internet, two thirds of them were vulnerable to our attack (for details see §4.3).

We note that the TLS protocol may be historically viewed as an SSL bearing the version number 3.1 [12], while the SSL with the version number 3.0 is often referred to as a "plain" SSL. There are some minor changes between SSL and TLS, but these changes are unimportant for the purpose of this study, since we rely on the general properties, which are common to both SSL v. 3.0 and TLS. Therefore, we will talk about them as about the SSL/TLS protocols. We note that SSL protocols with version numbers less than 3.0 will not be considered here, since they have already been proven to have several serious weaknesses [10], [16].

The rest of this chapter is organized as follows: in §2 we introduce a *bad-version oracle* (BVO), which is a construction that mathematically encapsulates side information leaking from the decoding process. The BVO is then used for mounting our attack in §3. The attack is based on an extended variant of Bleichanbacher's algorithm from [1]. The complexity of the attack together with the statistics of the vulnerable servers found on the Internet are given in §4. In §5 we discuss some technical details behind the practical realization of the attack. Countermeasures are then proposed and discussed in §6. The conclusions are made in §7. In the appendix we recall a slightly generalized version of the original Bleichenbacher's algorithm [1].

Proposition 1 (Connection and session). *Unless stated otherwise, the term connection means the communication carried out between a client and a server. It lasts from when the client opened up a networked pipe with the server, until the pipe is closed. The term session is used to refer to a particular part of this connection which is protected under the same value of symmetrical encryption keys.*

Proposition 2 (RSA-based session). *We say that the session is RSA-based if it uses the RSA scheme to establish its symmetrical keys.*

2. Bad-Version Oracle

We start by recalling the definition of PKCS-conforming plaintext [1]. Unless stated otherwise, the term plaintext means an *RSA* plaintext. Furthermore, we denote RSA instance parameters as (N, e, d), where N is a public modulus, e is a public exponent, and d is a private exponent, such that for all x, $x \in \langle 0, N - 1 \rangle$ it holds that $x = (x^e \bmod N)^d \bmod N$. We denote as k the length of the modulus N in bytes, i.e. $k = \lceil (\log_2 N)/8 \rceil$, and the boundary B as $B = 256^{k-2}$.

Definition 1 (PKCS-conforming plaintext). *Let us denote the plaintext as P, $P = \sum_{i=1}^{k}(P_i * 256^{k-i})$, $0 \leq P_i \leq 255$, where P_1 is the most significant byte of the plaintext. We say that P is PKCS-conforming if the following conditions hold:*

i) $P_1 = 0$
ii) $P_2 = 2$
iii) $P_j \neq 0$ *for all* $j \in \langle 3, 10 \rangle$
iv) $\exists j, j \in \langle 11, k \rangle, P_j = 0$; *the string* $P_{j+1}||...||P_k$ *is then called as a message M or a data payload*

The definition describes the set of all valid plaintexts for the given modulus of the length k bytes. In the case of SSL/TLS protocols, however, only the subset of this set is allowed, since these protocols introduce several extensions to the basic PKCS#1 (v. 1.5) format. Therefore, we define the term S-PKCS-conforming plaintext as follows.

Definition 2 (S-PKCS-conforming plaintext). *We say that P is S-PKCS-conforming if it is PKCS-conforming and the following conditions hold:*

i) $P_j \neq 0$ *for all* $j \in \langle 3, k - 49 \rangle$
ii) $P_{k-48} = 0$

The main restriction introduced here is the constant number of data bytes (which is equal to 48). The number of padding bytes equals k - 51. Furthermore, SSL/TLS protocols introduce a special interpretation for the first two data bytes P_{k-47} and P_{k-46}, which are respectively regarded as `major` and `minor` version numbers. This extension was introduced to thwart so-called version rollback attacks. The *data payload*, which is the concatenation of $P_{k-47} \parallel P_{k-46} \parallel P_{k-45} \parallel ... \parallel P_k$, is called a *premaster-secret* here. It is the only secret used in the key derivation process that produces the session keys used by the client and the server in the given session. An attacker, who is able to discover the *premaster-secret*, can decrypt the whole communication between the client and server which has been carried out in the session. The value of $P_{k-45} \parallel ... \parallel P_k$ is generated randomly by the client who then adds the version number P_{k-47} and P_{k-46}, encrypts the whole value of the *premaster-secret* by the server's public RSA key, and sends the resulting ciphertext C to the server. The server decrypts it and creates its own copy of the *premaster-secret*. All these steps are carried out during the Handshake sub-protocol (c.f. §5).

It is widely known that the server shall not report whether the plaintext P, $P = C^d \bmod N$, is PKCS-conforming or not. In practice, a server is recommended to continue with a randomly chosen value of the *premaster-secret* if the value of P is not S-PKCS-conforming. Obviously, the communication breaks down soon after sending a `Finished` message (c.f. §5), since the client and the server will both use different values for the session keys. However, the client (attacker) does not know whether the communication has broken down due to an invalid format of P or due to incorrect value of the *premaster-secret*. So, the attack is effectively defeated in this way. Of course, the attacker still gains some information from such an interaction with the server. She may at least try to confirm her guesses of the correct value of the *premaster-secret*. However, it has been shown by Jonsson and Kaliski [4] that it is infeasible to exploit this information for an attack.

Let us suppose that the server incorporates the above-mentioned countermeasure, the primary aim of which is to thwart Bleichenbacher's attack [1]. Furthermore, let all S-PKCS-conforming plaintexts be processed by the server to check the validity of proprietary SSL/TLS extensions according to the following proposition.

Proposition 3 (Conjectured server's behavior).

i) The server checks if the deciphered plaintext P is S-PKCS-conforming. If the plaintext is not S-PKCS conforming, the server generates a new premaster-secret randomly, thereby breaking down the communication soon, after receiving the client's `Finished` *message.*

ii) The server checks each S-PKCS-conforming plaintext P to see whether P_{k-47} = `major` *and* P_{k-46} = `minor`*, where* `major.minor` *is the expected version number which is known to the attacker (c.f. §5 for details). For instance, the most usual version numbers at the time of writing this analysis were 3.0 and 3.1. If the test fails, the server issues a distinguishable error message. The test is never done for plaintexts that are not S-PKCS-conforming.*

Practical tests showed that it is reasonable to assume Proposition 3 is fulfilled in many practical realizations of SSL/TLS servers.

Definition 3 (Bad-Version Oracle - BVO). *BVO is a mapping BVO:* $Z_N \rightarrow \{0, 1\}$*. BVO(C) = 1 iff* $C = P^e \bmod N$*, where e is the server's public exponent, N is the server's modulus, and P is an S-PKCS conforming plaintext, such that either* $P_{k-47} \neq$ `major` *or* $P_{k-46} \neq$ `minor`*, where* `major.minor` *is the expected version number. BVO(C) = 0 otherwise.*

BVO can be easily constructed for any SSL/TLS server that acts according to Proposition 3. We send the ciphertext C to the server and if we receive the distinguished message from (ii), we set BVO(C) = 1. Otherwise, we set BVO(C) = 0.

Theorem 1 (Usage of BVO). *Let us have a BVO for given (e, N) and* `major.minor` *and let C be an RSA ciphertext. Then BVO(C) = 1 implies that* $C = P^e \bmod N$*, where P is an S-PKCS-conforming plaintext.*

Proof. Follows directly from Definition 3.

■

Because S-PKCS-conforming plaintext is also PKCS-conforming, it follows from Theorem 1 that we can use BVO to mount Bleichenbacher's attack. We discuss the details in §3. Now we introduce several definitions that will be useful in the rest of this part. We use a similar notation to the one used in [1].

Definition 4 (Probabilities concerning BVO). *Let Pr(A) = B/N be the probability of the event A that the conditions (i-ii) of Definition 1 hold for randomly chosen plaintext P. Let Pr(S-PKCS|A) be the conditional probability that the plaintext P is S-PKCS-conforming assuming that A occurred for P. Let Pr(BVO|S-PKCS) be the conditional probability that BVO(*$P^e \bmod N$*) = 1 assuming that P is S-PKCS-conforming.*

For Pr(A) we have $256^{-2} < \Pr(A) < 256^{-1}$ as stated in [1]. The probability Pr(*S-PKCS*|A) can be expressed as $\Pr(S\text{-}PKCS|A) = (255/256)^{(k-51)} * 256^{-1}$, since the length of the non-zero padding bytes must be equal to k-51. There is usually one value of the version number that is expected by BVO (see §5). Therefore, Pr(*BVO*|*S-PKCS*) = $1-256^{-2}$. Note that the value of Pr(*BVO*|*S-PKCS*)*Pr(*S-PKCS*|A)*Pr(A) is the probability that for a randomly chosen ciphertext C we get BVO(C) = 1.

3. Attacking *Premaster-secret*

3.1 Mounting and Extending Bleichenbacher's Attack

This attack allows us to compute the value $x = y^d \bmod N$ for any given integer y, where d is an unknown RSA private exponent and N is an RSA modulus. This attack works under the condition that an attacker has an oracle that for any ciphertext C tells her whether the corresponding RSA plaintext $P = C^d \bmod N$ is PKCS-conforming or not. Theorem 1 shows that BVO introduced in the previous part can be used as such an oracle. In the case of the SSL/TLS protocols this means that we can mount this attack to either disclose a *premaster-secret* for an arbitrary captured session or to forge a server's signature. In the following text, we mainly focus on the *premaster-secret* disclosure. Forging of signatures is discussed briefly in §3.4.

The main idea here is to employ Bleichenbacher's attack with several changes related to the specific properties of S-PKCS and BVO (§3.2). Furthermore, we employed particular optimizations, which we have tested in our sample programs, and which generally help an attacker (§3.3).

3.2 S-PKCS and BVO Properties

We show how to modify Bleichenbacher's original RSA inversion algorithm for use with the BVO and to increase its efficiency. For the sake of completeness we repeat the necessary facts from [1] in the appendix together with a brief generalization of it.

Recall that PKCS-conforming plaintext P satisfies the following system of inequalities

$$E \le P \le F,$$

where $E = 2B$, $F = 3B–1$, and $B = 256^{k-2}$. The boundaries E, F are extensively used through the whole RSA inversion algorithm. Since BVO as well as the SSL/TLS protocols deal only with S-PKCS-conforming plaintexts, we may refine the boundaries as

$$E' \le P \le F',$$

where the value of E' is obtained by incorporating the minimum value of the padding and the value of F' is computed with respect to the fixed position of the zero delimiter in the plaintext P:

$E' = 2B + 1*256^{k-3} + 1*256^{k-4} + ... + 1*256^{49} = 2B + 256^{49}(256^{k-51} - 1)/255$ and

$F' = 2B + 255*(256^{k-3} + 256^{k-4} + ... +256^{49}) + 0 + 255*(256^{47} + 256^{46} + ... + 256^{0}) = 3B – 255*256^{48} – 1.$

Substituting E', F' in place of E, F in the original algorithm (see the appendix) increases its effectiveness.

It follows from the technical details (c.f. §5) that the attacker knows the expected value of the version number, which is checked by BVO. Therefore, when attacking the ciphertext C_0, such that BVO(C_0) = 0, carrying the *premaster-secret*, the attacker knows exactly the two bytes $P_{0,k-47}$ and $P_{0,k-46}$ of the S-PKCS-conforming plaintext $P_0 = C_0^d \bmod N$. She also knows that $P_{0,k-48} = 0$. We used this knowledge in our program to further trim the interval boundaries $<a, b>$ computed in step 3 of the algorithm (see the appendix).

3.3 Basic General Optimizations

Besides the optimizations that follow directly from §3.2, we also used the generally applicable methods described in the following subparagraphs.

Definition 5 (Suitable multiplier). *Let us have an integer C. The integer s is said to be a suitable multiplier for C if it holds that $C' = s^e C \bmod N = (P')^e \bmod N$, where P' is a S-PKCS-conforming plaintext.*

3.3.1 Beta Method

The following method (β-method) follows from a generalization of the remark mentioned in [1], pp.7 - 8.

Lemma 1 (On linear combination). *Let us have two ciphertexts C_i and C_j, such that $C_i = (s_i)^e C_0 \bmod N$, $C_j = (s_j)^e C_0 \bmod N$, where s_i and s_j are suitable multipliers for C_0. I.e. $P_i = C_i^d \bmod N = 2B + 256^{49} PS_i + D_i$ and $P_j = C_j^d \bmod N = 2B + 256^{49} PS_j + D_j$, where $0 < PS_{i,j}$ and $0 \leq D_{i,j} < 256^{48}$. Then for C, $C = s^e C_0 \bmod N$ and $\beta \in \mathbf{Z}$, where $s = [(1-\beta)s_i + \beta s_j] \bmod N$, it holds that $C^d \bmod N = P$, such that $P = [2B + 256^{49}((1-\beta)PS_i + \beta PS_j) + (1-\beta)D_i + \beta D_j] \bmod N$.*

Proof. It suffices to observe that $P = [(1-\beta)s_i + \beta s_j]P_0 \bmod N = [(1-\beta)P_i + \beta P_j] \bmod N$, where $P_0 = C_0^d \bmod N$.

■

It follows from the lemma written above that once we have suitable multipliers $s_{i,j}$ for a ciphertext C, we can try to search for the next suitable multiplier s as for a linear combination of s_i and s_j. In practice, we can try small positive and negative values of β and test whether the particular linear combination s gives S-PKCS-conforming plaintext or not. Working in this way, we may hope to accelerate the algorithm in step 2b (c.f. the appendix). Since we can reasonably assume that $\gcd(s_j - s_i, N) = 1$, there is a particular value of β for every triplet of suitable multipliers (s_i, s_j, s). However, experiments have shown that there are also differences in how much information can be obtained from such s depending on the size of β. For small values of β, it has been observed that the obtained values of s do not reduce the size of M_i as fast as the values of s obtained for β close to $N/2$. The reason is perhaps a linear dependency on $\mathbf{Z}$, which is stronger for small β. On the other hand, β close to $N/2$ clearly cannot be directly found by "brute force" searching. More precisely, we may find such β directly, but we cannot assure that obtained s will be of moderate size for further processing by the RSA inversion algorithm. Therefore, it remains to extract as much information as possible from reasonably small values of β and then to either continue

with incremental searching used in the original version of the algorithm [1] or to use the Parallel-Threads (PT) method described in §3.3.2. In advance of the following discussion, we note that the source for the next incremental searching or for the PT-method is the maximum suitable multiplier s_j found, such that $s_j < N/2$.

When using the above-mentioned method with negative values of β, we may get a multiplier s that is close to N (it can be regarded as a small negative value modulo N). Such an s cannot be directly processed, since it induces a very large interval for r in the original algorithm (see step 3 in the appendix). We will show how the algorithm can be adjusted to process small positive values of s as well as small negative values of s modulo N.

Theorem 2 (On symmetry). *Let us have integers s, P, and N satisfying*

$$E_1 \le sP \bmod N \le F_1, \text{ where } E_1, F_1 \in \mathbf{Z}.$$

Then there is the integer v, v = N - s, satisfying

$$E_2 \le vP \bmod N \le F_2, \text{ where } E_2 = N - F_1, F_2 = N - E_1.$$

Proof. We have that $vP \bmod N = (N - s)P \bmod N = (-sP) \bmod N = N - (sP \bmod N)$. The upper boundary of $(sP \bmod N)$ is F_1, therefore, the lower boundary E_2 of $(vP \bmod N)$ is $E_2 = N - F_1$. Analogically, the upper boundary F_2 of $(vP \bmod N)$ is given by the lower boundary E_1 as $F_2 = N - E_1$.

■

We use the theorem as follows: if we get a high value of s using the β-method described above, then we convert it to the corresponding symmetric value $v = N - s$ which is then processed in a modified version of step 3 of the algorithm (see the appendix). The core of the modification is using boundaries E_2, F_2 instead of the original boundaries $E_1 = E'$, $F_1 = F'$ (c.f. §3.2).

3.3.2 Parallel-Threads (PT) Method

Recall that the complexity of step 2 of the algorithm (see the appendix) for $i > 1$ depends on the size of M_{i-1}. Generally, the step is expected to be much faster if $|M_{i-1}| = 1$ than if $|M_{i-1}| > 1$. The reason is that $|M_{i-1}| = 1$ means there is only one interval approximating the value of P_0 left and therefore certain rules can be used when searching for the next suitable multiplier s_i. Experimenting with our test program, we observed that even if $|M_{i-1}| > 1$, the number of intervals was usually small enough that it was better to start a parallel thread T for each $I \in M_{i-1}$ as if it was the only interval left, i.e. it starts its own thread in step 2c of the algorithm. These threads T_1, ..., T_w, where $w = |M_{i-1}|$, were precisely multitasked on a per BVO call basis. They were arranged in the cycle $T_1 \to T_2 \ldots \to T_w \to T_1$ and stepping was done in the cycle after each one BVO call. The results obtained when thread T_j found a suitable multiplier were projected on the whole current set of intervals for all threads. After that, the threads belonging to the intervals that disappeared were discarded. We observed that the PT-method increased the effectiveness of the original algorithm.

Using a certain amount of heuristics we set the condition that directs whether we should use the PT-method or not. The PT-method is started in step i if the following inequality holds

$$| M_{i-1}| < (2\varepsilon \Pr(A))^{-1} + 1.$$

The value of ε estimates the number of passes it takes from the start of the PT-method until there is only one interval left, i.e. $|M_{i+\varepsilon-1}| = 1$, where the PT-method started in pass i. In our programs, we used $\varepsilon = 2$ which was the ceiling of the mean value observed for ε.

3.4 Note on Forging a Server's Signature

The BVO construction allows us to mount Bleichenbacher's attack without any restrictions on its functionality. As noted above, we can compute the RSA inverse for any integer y, thereby obtaining the value $x = y^d \bmod N$ for the particular server's private exponent d and the modulus N. Discussing the so-called semantics of the attack, there are only two cases in which it would be reasonable to compute this inversion.

In the first case we compute the RSA inverse for a captured ciphertext carrying an encrypted value of the *premaster-secret*. This approach allows us to decrypt the whole communication that was carried out in a given session between a client and the server. This is the main approach of our method, which we have practically tested and optimized.

In the second case we compute an RSA signature of a message m on behalf of the server. The whole attack runs in a similar way, which means that the main activity between an attacker and the server is still concerned on the phase of passing the *premaster-secret* value during the handshake procedure of the SSL/TLS protocols. However, this is only because we need to build up a BVO (c.f. §2) for computing the RSA inversion. The source of this inversion (the ciphertext C) will no longer be an encrypted *premaster-secret* itself, but the formatted value of $h(m)$, where h is an appropriate hash function. Currently, the SSL/TLS protocols sets $h(m) =^{\text{def}} \text{MD5}(m) \,||\, \text{SHA-1}(m)$ and the value of $h(m)$ is further formatted according to the EMSA-PKCS1-v1_5 method from PKCS#1 ([9], [10], [12], [13], [17]). At the end of the attack we obtain $C^d \bmod N$ which is the signature of our input C. It further depends on the `keyUsage` property [18] of the certificate of the server's RSA key, whether such a signature can be used for further attacks or not. At first the server's RSA key must be attributed for signing purposes. Secondly, it depends on the specific system as to how far the faked signature is important, directly implying how dangerous the attack is. From the basic properties of SSL/TLS ([10], [12]) it follows that such a signature may be abused to certify an ephemeral RSA or D-H [11] public key of a faked server. The faked server can then be palmed on an ordinary user to elicit some secret information from her. Generally speaking, this would be an attack on the authentication of a server. The necessary condition here is that the user is willing to use either the so-called export RSA key or the ephemeral Diffie-Hellman key agreement [11]. The practical situation is that some clients will - some clients will not. It strongly depends on the attention paid to the configuration of such a client. Unfortunately, these "minor" details are very often neglected in a huge amount of applications. Moreover, we emphasize that the attack described here may not be the only one possible, since the particular importance of a server's signature depends on the role that the server plays in a particular information system. The best way to avoid all these attacks is to

not attribute the server's RSA key for signing purposes, unless it is absolutely necessary.

From the effectiveness viewpoint, we can estimate that using the RSA inversion based on BVO for signature forging will require more BVO calls, since we need to insert an extra masking zero-step (see appendix, step 1 of the algorithm). The number of additional BVO calls may be calculated as $[\Pr(BVO|S\text{-}PKCS)^* \Pr(S\text{-}PKCS|A)^* \Pr(A)]^{-1}$, which is given by the probability that for a randomly chosen ciphertext C we get $\mathrm{BVO}(C) = 1$. Adding this value to the number of BVO calls in the former attack on *premaster-secret* (c.f. §4) gives an estimate of the overall complexity of signature forging.

4. Complexity Measurements

Basing on the elaboration from [1], we can estimate the number of BVO calls for decrypting a plaintext C_0 belonging to a S-PKCS-conforming plaintext P_0 as

$$2^*\Pr(P)^{-1} + (16k - 32)^*\Pr(P|A)^{-1}, \text{ where } \Pr(P|A) = \Pr(BVO|S\text{-}PKCS)^*\Pr(S\text{-}PKCS|A),$$

$$\Pr(P) = \Pr(P, A) = \Pr(P|A)^*\Pr(A),$$

where $\Pr(P)$ is the probability that for a randomly chosen ciphertext C we get $\mathrm{BVO}(C) = 1$.

This estimation does not cover the optimization described in §3.2 and §3.3. Therefore we treat it as the worst-case estimation for a situation when these optimizations are not notably helping an attacker. Experiments show that the optimized algorithm is practically almost two times faster than this estimation (c.f. §4.1) for the most widely used RSA key lengths. Let us comment on the expression of the estimation now.

The first additive factor corresponds with our assumption that the attacker wants to decipher C_0 belonging to a properly formatted plaintext carrying a value of the *premaster-secret*. In such a situation, she does not have to carry out initial blinding (c.f. the appendix, step 1). According to [1], we can estimate that she needs to find two suitable multipliers $s_{1,2}$ for C_0, until she can proceed with the generally faster step 2c. This gives the first factor as $2^*\Pr(P)^{-1}$. Note that, heuristically speaking, the optimizations (§3) mainly reduce the necessity of finding s_2 in the "hard" way, thereby decreasing the first factor closely to the value $\Pr(P)^{-1}$. This hypothesis corresponds well with the results of our measurements.

The second factor is a slightly modified expression presented in [1]. It corresponds to the number of expected BVO calls for the whole number of passes through step 2c. Recall that $C_0 = (P_0)^e \bmod N$, where $2B \le P_0 \le 3B - 1$, so P_0 lays in the interval of the length B, $B = 256^{k-2}$. Conjecturing that each pass through step 3 roughly halves the length of the interval for P_0, we may estimate that we need $8(k - 2)$ passes. Furthermore, it is conjectured [1] that each pass through step 2c takes $2^*\Pr(P|A)^{-1}$ BVO calls. From here follows the estimation of BVO calls as $(16k - 32)^*\Pr(P|A)^{-1}$.

Finally, we note that the complexity of the attack is mainly determined by the amount of necessary BVO calls. This amount actually limits the attack in the three

ways. The first one is that an attacked server must bear such a number of corrupted Handshakes (i.e. not collapse due to a log overflow, etc.). The second limitation comes from a total network delay that increases linearly with the number of BVO calls. The third limit is determined by the computational power of the server itself, which mainly means how fast it can carry out the RSA operation with a private key. Other computations during the attack are essentially faster and therefore we do not discuss them here.

4.1. Simulated Local BVO

In this paragraph, we present the measured complexity of the attack with respect to the total amount of BVO calls. The data of our experiment was obtained for the four particular randomly generated RSA moduli of 1024, 1025, 2048 and 2049 bits in length. For every such modulus we implemented a local simulation of BVO that we linked together with the optimized algorithm discussed in this study. We then measured the number of BVO calls for 1200 ciphertexts of the randomly generated and encrypted values of the *premaster-secret*.

Due to the strong dependence of the number of BVO calls on Pr(A) we see that the complexity of the attack is not strictly increasing with respect to the length of the modulus N. This discrepancy was already mentioned in [1]. It follows that one should use moduli with a bit length in the form $8r$, where r is an integer, mainly avoiding the moduli with the length $8r + 1$.

Table 1. Basic statistics of a measured attack complexity in BVO calls

Modulus length (bits)	**BVO calls**				
	Originally estimated without optimization	**Practically measured** (with optimizations from §3)			
		Min	Max	Median	Mean
1024	36 591 001	815 835	278 903 416	13 331 256	20 835 297
1025	979 488	630 589	105 122 011	1 197 380	1 422 176
2048	48 054 328	2 824 986	354 420 492	19 908 079	28 728 801
2049	2 794 937	1 413 005	475 298 397	3 462 557	3 896 432

Analyzing the measured data, we observed that the distribution of the amount of BVO calls can be approximated by a log-normal Gaussian distribution, i.e. the logarithm of the amount of BVO calls roughly follows a normal Gaussian distribution. Heuristically speaking, this means that the most basic random events governing the complexity of the attack primarily combine together in a multiplicative manner. The values of median, mean, and variance are presented in Table 1. These values were obtained using the log-normal approximation of the data samples measured. These approximations are plotted in Fig. 1 and Fig. 2. We can see that all the distributions skew to the right. Therefore, the most interesting values are perhaps given by the medians. For example, in the case of a 1024 bits long modulus, we can expect that the one half of all attacks succeed in less then 13.34 million BVO calls. Furthermore, the data in Table 1 supports our conjecture that the optimizations

proposed in §3 mainly speed up the first “hard” part of the algorithm. Therefore, this speeding up is clearly notable for moduli of 1024 and 2048 bits, while there is no observable effect for the moduli of 1025 and 2049 bits.

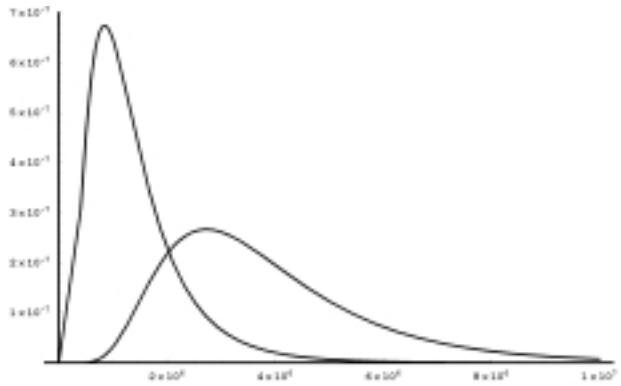

Fig. 1. Log-normal approximation of BVO calls density functions for 1024 (higher peak) and 2048 bits long moduli

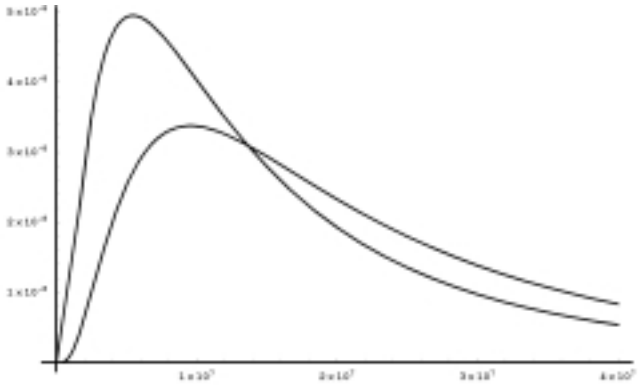

Fig. 2. Log-normal approximation of BVO calls density functions for 1025 (higher peak) and 2049 bits long moduli

4.2. Real Attack

We successfully tested the attack on a real SSL server (AMD Athlon/1 GHz, 256MB RAM) using 1024 bits long RSA key. The total number of BVO calls for decryption

of a randomly selected *premaster-secret* was 2 727 042 and the whole attack took 14 hours 22 minutes and 45 seconds. It gives an estimated speed of 52.68 BVO invocations per second. The server and the attacking client were locally connected via a 100 Mb/s Ethernet network without any other notable traffic. With respect to the whole conditions of this experiment, we can conclude that this is probably one of the best practically achievable results. Therefore, we can expect that there would be few practical attacks succeeding in less then 14 hours of sustained high effort (for a 1024 bits long RSA key). Using the value of the median for 1024 bits modulus from Table 1, we can roughly expect one half of all attacks in our setup to succeed in less than 70 hours and 18 minutes. For 2048 bits long RSA key in the same setup we get an estimated speed of 11.47 BVO calls per second. Therefore, one half of all attacks should then succeed in less than 21 days.

The experiment setup described above could be slightly improved by using a more powerful server. Plugging in such a server (2x Pentium III/1.4 GHz, 1 GB RAM, 100 Mb/s Ethernet, OS RedHat 7.2, Apache 1.3.27), it was possible to achieve a speed of 67.7 BVO calls per second for a 1024 bits RSA key. The median time for a whole attack on the *premaster-secret* could be then estimated as 54 hours and 42 minutes. Note that all these estimates assume achieving and sustaining high communication and computation throughput on the server's side.

4.3. Real Vulnerability

To assess the practical impacts of the attack presented here, we had randomly generated a list of 611 public Internet SSL/TLS servers (we accepted servers providing SSL v. 3.0 or TLS v. 1.0) and then tested these servers to see whether it was possible to construct a BVO for them or not. We found that two thirds of these servers were vulnerable to our attack. We emphasize that it does not necessarily mean that the attack would always succeed on every such server. Despite the fact that all these servers can be regarded as broken from a pure cryptanalytic viewpoint, the complexity of the attack may still render it impractical in a large amount of cases. We expect that a properly administrated server (e.g. log messages are often inspected, suspicious clients are added to black-lists, etc.) should withstand the attack. Under such an administration, the attack should be recognized and the attacking client would soon be blocked. Of course, the cryptographic strength of all these SSL/TLS implementations should definitely be improved. We strongly recommend applying appropriate patches as soon as possible.

We observed an interesting anomaly for 110 out of 611 tested servers. All of them provided both SSL v. 3.0 and TLS. 26 of them were *primarily* vulnerable only through the SLL v3.0 protocol, while the remaining 84 servers were *primarily* vulnerable only through the TLS protocol. We advisedly used the word "primarily", since if these servers share the same RSA key for both protocols, which is a very common practice, then an attacker can easily assault one protocol through an interaction with the other one. Moreover, the format of the ciphertext carrying the *premaster-secret* is the same for both protocols, so this cross-attacking actually does not increase the complexity of the whole attack.

5. Technical Details

In the SSL/TLS protocols, there are several sub-protocols that are used during various stages of a connection ([10], [12]). The most interesting ones for our attack are the Handshake and the Alert sub-protocols. The Handshake is used at the beginning of each session and its aim is to establish symmetrical encryption keys for that session. The SSL/TLS protocols allow various combinations of symmetrical and asymmetrical cryptographic schemes which are called *CipherSuites*. Our attack focuses on those *CipherSuites* that use the RSA algorithm for establishing symmetrical session keys. These suites are used by a huge amount of contemporary applications.

The messages exchanged between a client and a server during the Handshake and their mandatory order are given on Fig. 3.

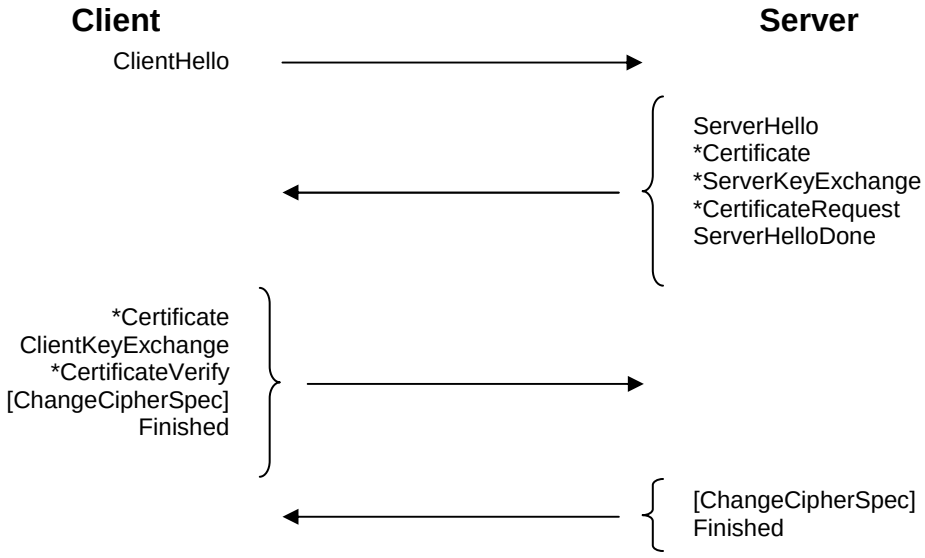

Fig. 3. SSL/TLS Handshake

The messages marked with * are optional, however, they must be in their given places if they are issued. The main purposes of the `ClientHello` message are announcing supported *CipherSuites* and protocol versions together with presenting a client's random value used for the derivation of session keys. The server replies with the `ServerHello` message in which it selects the *CipherSuite* and the protocol version giving the highest possible security level for this session. In RSA-based sessions, the `ClientKeyExchange` message carries an encrypted value of the *premaster-secret* that was formatted to be S-PKCS-conforming before its encryption (c.f. §2). The sequence of messages `[ChangeCipherSpec]` and `Finished` denotes the end of the Handshake. Note that the `ChangeCipherSpec` is not properly part of the Handshake sub-protocol [10], but it is an unimportant technical

detail here. The `Finished` messages are the first messages protected by the newly negotiated *CipherSuite* and session keys. They also include a cryptographic checksum of all the exchanged data that has been part of the Handshake sub-protocol. The checksum is computed using the PRF method [10] under the new session keys. Under normal operation, the Handshake sub-protocol passes to the Application Data sub-protocol, which serves as a transparently secured data pipe between the client and the server. The connection may return to the Handshake phase later on whenever the client decides to start a new session (the current session is then closed). We note that our attack is entirely focused on analyzing the first session in a captured connection. The second and next sessions will have their Handshakes encrypted under the keys belonging to their "parent" session. It is still possible to decrypt these sessions however, the attacker must do this on a session-by-session basis starting from the first one.

Possible fatal errors and warnings arising during SSL/TLS communication are reported using the Alert sub-protocol. Its messages transparently flow between packets of other sub-protocols. Once the session is well established, the Alert messages are also cryptographically protected.

5.1 Constructing BVO

From the technical viewpoint, each BVO(C) call means starting a new RSA-based session with the Handshake described above. The attacker does mainly the following: she asks for an appropriate RSA-based session, sets the version number in the `ClientHello` (c.f. §5.2), and incorporates the challenge C into the `ClientKeyExchange` message. She completes the Handshake using the session keys derived for a randomly guessed value of the *premaster-secret'*. Then she waits for the server's reaction, which will be sent using the Alert sub-protocol. The core of her approach is based on the Alert message she gets. The attack relies on the possibility of identifying the case in which the decrypted plaintext P, $P = C^d \bmod N$, is S-PKCS-conforming with a bad version number. For example, the OpenSSL implementation [7] sends the "handshake failure" (for SSL v. 3.0) or "decode error" (for TLS v1.0) Alert message in such a situation. This message is unencrypted, since the session has not been set up yet, and it precisely distinguishes the situation described above from other possible erroneous states at the end of the Handshake. We note that the Alert message may not be the one and only way in which the BVO can be constructed. There may be other side channels that allow a BVO construction for instance, a timing side channel. The use of this side channel in connection with a SSL/TLS server was already demonstrated in a different attack [2].

5.2 Version Number

According to the version number included in the *premaster-secret*, the documents [10], [12] say the following: "*Upon receiving the premaster-secret, the server should check that this value matches the value transmitted by the client in the*

`ClientHello` *message*" (c.f. [12], p. 44). However, Rescorla noted (c.f. [10], p. 79) "*...but in practice many clients use the negotiated version instead...*". So, there may be two correct values of the version number for the given Handshake. As may be expected, from a pure cryptological point of view, the situation becomes a bit messy when moving from theory to a practice. It solely depends on the particular server whether it requires the version number to be set to the value offered in `ClientHello` (type-I server) or if it allows this value to be also set to the negotiated version from `ServerHello` (type-II server). However, such a situation may be checked before an attack and the particular behavior of the server may be then well estimated. With this knowledge, we can construct a BVO so that there is only one correct value of the version number expected. If the server is of type-I then there is no problem with this. If it is of type-II then all we have to do is to find the setting under which the version number negotiated in `ServerHello` becomes the same as the number offered in `ClientHello`. Then the server behaves exactly like a server of type-I.

We can also use the version number that is known from the `ClientHello` and `ServerHello` to narrow the size of intervals searched for in the attack (c.f. §3.2).

First, we compare the version numbers from `ClientHello` (ver_1) and `ServerHello` (ver_2) belonging to the Handshake of the analyzed session. If they are the same, then this common value must also have been included in the *premaster-secret*. If these values are different then we change the value in `ClientHello` so that the value chosen by the server would not change. We start a new session using this `ClientHello` message. In place of the `ClientKeyExchange` we use directly the `ClientKeyExchange` from the analyzed session. If the server terminates the connection due to a bad version number, then we know that there is ver_1 in the attacked *premaster-secret*. Otherwise, it is ver_2.

6. Countermeasures

Due to the compatibility demands, it does not seem possible to simply leave the EME-PKCS1-v1_5 method and use its successor EME-OAEP. Note that even the EME-OAEP method must be implemented carefully (c.f. [5], [6]). On the other hand, it has been recently shown by Jonsson and Kaliski in [4] that the EME-PKCS1-v1_5 can offer reasonable security (the proof was carried out for the TLS protocol) assuming that it is implemented properly – i.e. mainly that side channels are avoided. What remains is to show what a proper implementation should look like. The current guidelines in [12] together with [15] are obviously insufficient and should be updated to avoid weaknesses like the one discussed in this analysis. Moreover, it seems that the edge between secure and insecure implementation of EME-PKCS1-v1_5 is very sharp. This implies that the standards regarding its implementation must really be very precise.

6.1 Promising Countermeasures which Are Cryptographically Odd

The following countermeasures can hinder our attack, but each one exposes at least a theoretical cryptographic weakness.

6.1.1 Testing Randomly Generated Payload

It may be a tempting idea to introduce the following general countermeasure: at first, the plaintext P being decoded is checked to see if it is S-PKCS-conforming. If it is not, then the data payload $P_{k-47} \parallel ... \parallel P_k$ is randomly generated from scratch. Therefore, at the end of the first phase, we should have a data payload, no matter how it was obtained - whether by extracting or by random generating. The data payload would then be the subject of all successive checks, especially the version number test. Note that with a little effort, this countermeasure can be read between the lines on page 44 of RFC 2246 [12].

We will show that such a countermeasure is not cryptographically good, since the successive checks work on small parts of the data payload. By examining a sequence of Handshake invocations, we can distinguish whether the server generates the payload randomly or if it uses the result of decoding a properly formatted plaintext P. Thereby, we know if the original plaintext P is S-PKCS-conforming or not.

In our attack, such a countermeasure would mean that an attacker has to change her strategy. Now, there is a high probability that a randomly chosen ciphertext C gives a bad version number. The attacker will generate random C and wait until the server responds that the version number is correct. With this result she still does not know whether $P = C^d \bmod N$ is a correct S-PKCS-conforming plaintext (with a correct version number) or if the version number was accidentally correct in the random payload. However, she can decide between these two variants by sending the same C again. If it again gives the correct version number, then it is highly probable that P is S-PKCS-conforming, since the probability that a consecutive randomly generated payload again gives the correct version number is close to 256^{-2} here. The probability that even a third consecutive invocation gives the correct version number, given that P is not S-PKCS-conforming, is close to 256^{-4}, etc. Let us say that she would always carry out these two checking invocations for each ciphertext of a possibly S-PKCS-conforming plaintext. If X denotes the number of necessary oracle calls in the BVO-attack, then she needs approximately $X*(2^{16}+2)$ oracle calls now. Using only one control invocation gives the estimate as $X*(2^{16}+1)$ oracle calls. We note that even this generally low complexity may successfully thwart the attack in many practical implementations. On the other hand, such a protocol still cannot be regarded as cryptographically secure.

6.1.2 Treating the Version Number as a PKCS Mark

Another seductive countermeasure is to regard a failure of the version number check as a failure of EME-PKCS1-v1_5 decoding. Therefore, seeing an incorrect PKCS#1 format or an incorrect version number, the server would randomly generate new data payload $P_{k-47} \parallel P_{k-46} \parallel ... \parallel P_k$ and continue with it through the rest of the Handshake (of course omitting the version check). Such a countermeasure would effectively thwart the attack described here.

This countermeasure is nearly perfect but we have to note that there are some weaknesses, even if they are highly theoretical and impractical yet. The first disadvantage is that an attacker can change the formatting rules at her will by changing the expected version number (c.f. the role of `ClientHello` in §5.2). Let C be a ciphertext corresponding to an S-PKCS-conforming plaintext P, i.e. $C = P^e \bmod N$. Manipulating the expected version number, the attacker can force a server to either accept or reject C, where "to reject C" means to generate a random payload instead of using the data payload from P. The attacker can do that even when P is unknown. Now, let us consider that the server is implemented as a small electronic device (e.g. as a chipcard, embedded module, etc.) allowing the attacker to listen to power or electromagnetic side channels. Then the attacker may, for example, try to set up a distinguisher Δ, Δ: $(C, v_1, v_2) \rightarrow \delta$, where $\delta \in \{0, 1, 2\}$, C is a ciphertext and v_1, v_2 are different version numbers. If $\delta \in \{1, 2\}$, then it means that the ciphertext C is accepted under the expected version number v_i, $i = \delta$, while it is rejected under v_j, $j = 3 - i$. Otherwise $\delta = 0$. A possible way of building Δ is to seek for correlations between the samples of a side channel signal captured at the time of the RSA decryption and the *premaster-secret* processing. For illustration, let us assume that the correct answer is $\delta \in \{1, 2\}$. The signals set $\{S_i\}$ obtained for C sent under the version number v_i, $i = \delta$ should then be mutually correlated at some points due to the same value of the *premaster-secret* processed (since C is valid under v_i, the data payload from P will always be used for the *premaster-secret*). On the other hand, the signals set $\{S_j\}$ obtained for C under v_j, $j = 3 - i$ should be significantly less correlated for a particular subset of these points, since the *premaster-secret* is generated randomly for each invocation. However, since the ciphertext C and the corresponding plaintext P stay always the same, there should be points where the signals $\{S_j\}$ are still correlated. Therefore, the attacker can compare the correlations among signals $\{S_i\}$ with the correlations among $\{S_j\}$ and estimate a probable correct value of δ. If the behavior of the signals analyzed does not correspond with this model, the attacker places $\delta = 0$ (i.e. distinguishing was impossible). Such an analysis is possible mainly due to the free control of acceptance rules for C and this is the reason why we conjecture that the countermeasure discussed here may be considerably susceptible to side channel attacks.

Once the attacker has Δ, she can perform an attack, which is in fact similar to the attack described in §6.1.1. Let us assume that v_1, v_2 are two different constants which are carefully selected and remain the same during the attack. To confirm that a given ciphertext C corresponds to an S-PKCS-conforming plaintext P, the attacker invokes Δ as $\delta = \Delta(C, v_1, v_2)$. If $\delta \in \{1, 2\}$, then the attacker knows that P is S-PKCS-conforming (and, furthermore, that it was accepted under either v_1 or v_2). Otherwise, the attacker continues searching for next C. Let X denote the number of necessary oracle calls in the BVO-attack and let T_Δ denote the number of server calls for one invocation of Δ. Then we may estimate that the attacker would need approximately $Z = X*T_\Delta*2^{15}$ server calls in total. The factor 2^{15} corresponds with the probability that a randomly chosen S-PKCS-conforming plaintext gives the version number v_1 or v_2. If we assume that $1 < T_\Delta < 256$, then we have $X*2^{15} < Z < X*2^{23}$. Although such a complexity would probably thwart the attack, it should not be totally neglected from a general point of view.

Second theoretical threat is in the following: Suppose that the attacker has a ciphertext C, such that $C = P^e \bmod N$ for an S-PKCS-conforming plaintext P, and at the same time she already knows $\mu = P_{k-45} \| ... \| P_k$. Then she can discover the values of P_{k-47} and P_{k-46}.

She does so by sending the `ClientHello` (*ver*) messages for different versions *ver* according to her choice together with the ciphertext C in the `ClientKeyExchange` message. She sends these messages until the server uses the *premaster-secret* from P. Since she knows the rest of the *premaster-secret*, she can compute `Finished` message correctly. When the server accepts her `Finished` message, she hit the correct version *ver*, originally unknown to her. From a purely cryptographic viewpoint, such a property should be avoided.

The approach described above is recommended by Rescorla ([10], p. 170), who attributes it to RFC 2246 [12]. However, from a closer cryptographic examination, it may seem that RFC 2246 rather recommends the measure from §6.1.1. Evidently, this is another reason why this standard deserves a certain amount of additional explanation.

6.2 Countermeasure which Is Both Practically and Cryptographically Bearable

Demands: Recall that a good countermeasure should mainly:

i) ensure that tampering with the version number is detected,
ii) hide partial information about the RSA plaintext being decoded as much as possible,
iii) not allow new attacks.

Proposal: Under these demands, we propose processing the ciphertext C from the `ClientKeyExchange` message to get the *premaster-secret* as follows:

i) decrypt C on P as $P = C^d \bmod N$
ii) check if P is S-PKCS-conforming, if it is not, replace the values $P_{k-45} ... P_k$ by 46 bytes of random data
iii) in any case securely discard $P_1 ... P_{k-48}$
iv) in any case replace P_{k-47} and P_{k-46} by the expected `major` and `minor` version numbers, respectively (c.f. §5.2)
v) set *premaster-secret* $= P_{k-47} \| ... \| P_k$

The server may optionally check the version number from the original plaintext P against the expected values and log the result of this test. It may also log the check result from step (ii). However, all these logs should then be regarded as sensitive values. Note that this countermeasure may also be attacked when the information on the checks leaks out (directly or indirectly) through side channels. Despite of giving attackers perhaps less prominent chances than the countermeasure §6.1.2, the threat of side channels must not be underestimated, since they may still allow devastating attacks here.

There is a minor problem relating the tolerant type-II servers (c.f. §5.2) in that such a server will do the substitution in step (iv) twice, since two values are expected for the version number. It also means that the server must compute two *premaster-secrets* as well as sets of session keys and hold them until seeing the client's `Finished`.

After then the server decides which one should be used and which one can be discarded.

We conjecture that the countermeasure presented above meets our demands. To support this hypothesis, we present the following arguments.

Let us consider tampering with a version number so that the offered number from a client is not the one received by the server. Such tampering will be detected after exchanging `Finished` messages, since the client and the server will both use different values of the *premaster-secret*. We may reasonably assume that it would be infeasible for an attacker to also tamper with these `Finished` messages, which do not only carry a cryptographic checksum, but are also protected using session keys derived from the *premaster-secret*.

According to the server's behaviour proposed above and the possible chosen ciphertext attack, an attacker is only able to try to distinguish whether the server uses a random or the original value of the *premaster-secret* in step (ii). Assume the attacker has a ciphertext C, where $C = P^e \bmod N$. Let us denote $\mu = P_{k-45} || \ldots || P_k$.

- If she does not know the value of μ, she cannot get any new notable partial information about P, since she is unable to distinguish whether the server uses a random or the original (*version number* $||$ μ) value of the *premaster-secret*.
- If she knows the value of μ, she has an oracle O_μ: $\mathbf{Z}_N \rightarrow \{0, 1\}$, so that $O_\mu(C)$ tells her whether C decrypts to a S-PKCS-conforming P or not. In this way she gets certain partial information about the plaintext of this specific ciphertext C. Such information does not seem to be of notable merit provided she can only get it for some singular ciphertexts. For instance, she cannot learn the exact values of P_{k-47} and P_{k-46} as in §6.1.2, since the server does not use the values P_{k-47} and P_{k-46} in any way. Let us assume that the attacker is able to use $O_\mu(C)$ for any ciphertext C. This means that she can know the appropriate value of μ for every such ciphertext. From here and the theorem of RSA individual bits [3], it follows that she can invert the RSA permutation $x \rightarrow (y = x^e \bmod N)$ for any integer y. Proof of the reduction from a partial-RSA problem to a gap-RSA-P1 problem in [4] even shows an optimized algorithm for such an inversion (see [4], appendix A, proof of Lemma 1). It follows that getting the possibility of routine O_μ usage (i.e. "un-keying" it for any C) is as hard as inverting the whole RSA. Therefore, we conjecture that leaking partial information about P is minimized.

From step (iv) it follows that an active attacker can use messages from a captured session to tamper with the server using various version numbers for the *premaster-secret*. However, all she can do is to make the server set P_{k-47} and P_{k-46} at arbitrary values of her choice (using `ClientHello`, see §5.2) and then wait to see if the server accepts a tampered content of `Finished` belonging to the captured session. With regard to how the computation of `Finished` together with the derivation of session keys are carried out ([10], [12]), one can hardly expect that successful attacks would be constructed in this way.

7. Conclusions

We have presented a new practically feasible side channel attack against the SSL/TLS protocols. When Bleichenbacher presented his attack on PKCS#1 (v. 1.5) in 1998 [1], it was generally assumed that the attack was impractical for the SSL/TLS protocols, since these protocols add several proprietary restrictions on the plaintext format, which increase the complexity of the attack. Of course, the protocols could not be called secure from a pure cryptographical viewpoint. Therefore, a special countermeasure was introduced and generally adopted [10], [12]. However in this chapter, we have shown that problems with Bleichenbacher's-like attacks on the SSL/TLS protocols are still not properly solved. We have identified a new possibility of a substantial side channel occurring during an SSL/TLS Handshake. The side channel originates when a receiver checks a version number value stored in the two left-most bytes of the *premaster-secret*. Based on the receiver's behavior during this check, we have defined its mathematical encapsulation as a *bad-version oracle* (BVO, c.f. §2). Such a check is widely recommended for SSL/TLS servers, but unfortunately it is not properly specified how it should be performed. Practical tests showed that two thirds of randomly chosen Internet servers carried out the test wrongly, thereby allowing the construction of BVO resulting in a new attack on RSA-based sessions. The attack itself may be viewed as an optimized and generalized variant of the original Bleichenbacher's attack [1]. The most obvious target of our attack would probably be discovering the *premaster-secret*, thereby decrypting a captured RSA-based session. It is also possible (with an additionally increased complexity, c.f. §3.4) to compute the signature of any arbitrary message on behalf of the server.

The attack was carried out in practice and its efficiency was measured (§4). The amount of time the attack takes in practice is mainly determined by the amount of BVO calls. Each BVO call corresponds to one attempt to establish a SSL/TLS connection with an attacked server. If the server uses a typical 1024 bits long RSA key, then we can expect that roughly 50% of attacks succeed in less than 13.34 million BVO calls. For a practical estimation, the particular server speed must be known. For instance, in one of our testing setups we achieved a speed that allowed us to expect that 50% of attacks succeeded in less then 54 hours and 42 minutes. This load may be further spread as 2 hours of these interactions per day, thereby spreading the whole attack over roughly one month, etc. The attack is not limited to running on a single computer, so it can be distributed. The main aim would not be speeding up the attack, but making its localization and blocking harder. Although the complexity presented here is definitely very low from a pure cryptographic viewpoint, there may still be technical measures that can thwart the attack in a practice. For instance, each BVO call should produce at least one log record on the server's side. If these logs are well maintained and appropriately inspected, then the attack should be recognized in time. Unfortunately, there also seem to be poorly administrated servers where SSL/TLS audit messages are almost ignored. These servers remain protected solely by their network and computational throughput, which is obviously alarming.

Finally, we conclude that even those well-administrated servers should be patched to thwart the attack in a primarily cryptographic rather than a pure technical way. For this case, we have discussed various possible countermeasures in §6. There are three countermeasures presented the strength of which can be commented on as follows.

The measure §6.1.1 is obviously weak and should be avoided. The measure §6.1.2 thwarts our attack effectively, however, it still leaves a weakness (even though it is purely theoretical). The third countermeasure presented in §6.2 seems to be both cryptographically and practically optimal. However, for those implementations that are already using the measure from §6.1.2, we do not think it is necessary for them to immediately follow §6.2. This should be used mainly in new implementations of the SSL/TLS protocols.

References

1. Bleichenbacher, D.: *Chosen Ciphertexts Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1*, in Proc. of CRYPTO '98, pp. 1 - 12, 1998
2. Canvel, B., Hiltgen, A., Vaudenay, S., and Vaugnoux, M.: *Password Interception in a SSL/TLS Channel*, In proc. of CRYPTO '03, pp. 583-599, 2003
3. Håstad, J. and Näslund, M.: *The Security of Individual RSA Bits*, in Proc. of FOCS '98, pp. 510 - 521, 1998
4. Jonsson, J. and Kaliski, B.-S., Jr.: *On the Security of RSA Encryption in TLS*, in Proc. of CRYPTO '02, pp. 127 -142, 2002
5. Klíma, V., Rosa, T.: *Further Results and Considerations on Side Channel Attacks on RSA*, in Proc. of CHES '02, August 13 - 15, 2002
6. Manger, J.: *A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1*, in Proc. of CRYPTO'01, pp. 230-238, 2001
7. OpenSSL: *OpenSSL ver. 0.9.7*, http://www.openssl.org/, December 31, 2002
8. *PKCS#5 v2.0: Password-Based Cryptography Standard*, RSA Laboratories, March 25, 1999
9. *PKCS #1: RSA Encryption Standard*, An RSA Laboratories Technical Note, Version 1.5, Revised November 1, 1993
10. Rescorla, E.: *SSL and TLS: Designing and Building Secure Systems*, Addison-Wesley, New York, 2000
11. RFC 2631: Rescorla, E.: *Diffie-Hellman Key Agreement Method*, June 1999
12. RFC 2246: Allen, C. and Dierks, T.: *The TLS Protocol*, Version 1.0, January 1999
13. RFC 1321: Rivest, R.: *The MD5 Message-Digest Algorithm*, April 1992
14. Rivest, R., L., Shamir, A., and Adleman, L.: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, 21, pp. 120-126, 1978
15. RSA Labs: *Prescriptions for Applications that are Vulnerable to the Adaptive Chosen Ciphertext Attack on PKCS #1 v1.5*, RSA Laboratories, http://www.rsasecurity.com/rsalabs/pkcs1/prescriptions.html
16. Schneier, B. and Wagner, D.: *Analysis of the SSL 3.0 Protocol*, The Second USENIX Workshop on Electronic Commerce Proceedings, USENIX Press, November 1996, pp. 29 - 40
17. *Secure Hash Standard*, FIPS Pub 180-1, 1995 April 17
18. *X509: ITU-T Recommendation X.509 (06/97)* - Information Technology - Open System Interconnection - The Directory: Authenticantion Framework, ITU, 1997

Appendix

For the sake of completeness we enclose here the algorithm from [1]. For our purposes we define directly a slight generalization and modification of it. Recall that in the original text $E = 2B$, $F = 3B–1$, where $B = 256^{k-2}$. In our variant, we will use the refined values E' and F' (c.f. §3). According to Definition 1 and the original notation used bellow, we note that a ciphertext C is said to be PKCS conforming iff $C = P^e \bmod N$, where P is PKCS-conforming plaintext. The modified algorithm is as follows.

Step 1: Blinding. Given an integer c, choose different random integers s_0; then check, by accessing the oracle, whether $c(s_0)^e \bmod N$ is PKCS conforming.

For the first successful s_0, set

$$c_0 \leftarrow c(s_0)^e \bmod N$$
$$M_0 \leftarrow \{[E, F]\}$$
$$i \leftarrow 1.$$

Step 2: Searching for PKCS conforming messages.

Step 2.a: Starting the search. If $i = 1$, then search for the smallest positive integer $s_1 \geq N/(F+1)$, such that the ciphertext $c(s_1)^e \bmod N$ is PKCS conforming.

Step 2.b: Searching with more than one interval left. Otherwise, if $i > 1$ and the number of intervals in M_{i-1} is at least 2, then search for the smallest integer $s_i > s_{i-1}$, such that the ciphertext $c(s_i)^e \bmod N$ is PKCS conforming.

Step 2.c: Searching with one interval left. Otherwise, if M_{i-1} contains exactly one interval (i.e. $M_{i-1} = \{[a, b]\}$), then choose small integer values r_i, s_i such that

$$r_i \geq \lceil 2(bs_{i-1} - E)/N \rceil$$

and

$$(E + r_iN)/b \leq s_i < (F + r_iN)/a$$

until the ciphertext $c(s_i)^e \bmod N$ is PKCS conforming.

Step 3: Narrowing the set of solution. After s_i has been found, the set M_i is computed as

$$M_i \leftarrow \cup_{(a,b,r)} \{[\max(a, \lceil (E + rN)/s_i \rceil), \min(b, \lfloor (F + rN)/s_i \rfloor)]\}$$

for all $[a, b] \in M_{i-1}$ and $(as_i - F)/N \leq r \leq (bs_i - E)/N$.

Step 4: Computing the solution. If M_i contains only one interval of length 1 (i.e., $M_i = \{[a, a]\}$), then set $m \leftarrow a(s_0)^{-1} \bmod N$, and return m as solution of $m \equiv c^d \pmod N$. Otherwise, set $i \leftarrow i + 1$ and go to step 2.

G. Key-collisions in (EC)DSA: Attacking Non-repudiation*

1. Introduction

The service of non-repudiation is one of the most basic cryptographic goals [12]. The commonly agreed definition of this service says that: *The non-repudiation property of a given action allows an independent third party to make sure that a particular event did (or did not) occur* [10]. Note that the independent party is typically a judge. Such a property is of a great importance for applications where cryptographic mechanisms enter an area of law. A good example of such a service is the introduction of *electronic signature standards and laws* ([3], [4]), which is an ongoing activity through the whole world. There are also other areas where the non-repudiation plays an important role, for instance we may refer to asymmetric traitor tracing schemes [9], which achieve their non-repudiation by using signature schemes having this property. All these examples show that contemporary cryptographic mechanisms must not only protect data, but also be judiciously sound.

In this chapter, we show an approach which can be used to break the non-repudiation service in systems which are based on the (EC)DSA [6] signature schemes. The main idea behind our attack may be called an *"alternative explanation"*. This is a familiar and effective way in which a signature may be denied. An attacker constructs the alternative explanation that argues that there is the (mathematically) valid signature, while she claims that she has never signed the document presented at the court. Such an argument is then presented to the judge. We should note, that the provable mathematical connection between document signing and signature verification is that proper signing implies verifying the signature as a valid one. Proving this implication in the opposite direction, i.e. that verifying the signature as a valid one implies that the document was properly signed before, is somewhat tricky. In most schemes, this "proof" is just based on that there is no sound alternative explanation of why the verification procedure says that the signature is valid. Then, presenting such an alternative explanation means that the trial must be tried basing on other evidences, what may at least be very inconvenient. Therefore, a possibility of the alternative explanation existence should be minimized, especially on the elementary mathematical level. A well-known way to get the alternative explanation is to exploit collisions of a hash function used in the signature scheme. The attacker then claims that she has signed a *different message*, instead of that message presented at the court. Such a threat is well recognized and an adequate attention is usually paid to collision resistance of hash functions in signature schemes.

* An extended version of the research note presented as Rosa, T.: *On Key-collisions in (EC)DSA Schemes*, CRYPTO 2002 Rump Session, IACR ePrint archive 2002/129, Santa Barbara, USA, August 2002. Preliminary version of the paper won the best presentation award on the Czech cryptologic workshop VKB 2002 [14].

Apparently, some attention is also paid to the inner collision resistance of the signature transformations ([17], [18]). In this chapter, we show that it is also possible to get a collision of public keys (more precisely of public signature scheme instances), which we refer to as a key-collision (*k*-collision, c.f. §2 for the definition). Then, the alternative explanation says that it was *someone else* who has signed that message. A straightforward decision that users have *both* signed this message would not be proper, likewise we do not accept the decision that the user has signed *both* messages in the case of the message-collision (*m*-collision). Moreover, there may be cases where it is logically impossible for two users to have signed the given message. For instance, the signature scheme may be employed in an authentication service which disallows more than one user to be authenticated at the same time.

It is easy to see that *m*-collisions and *k*-collisions both weaken the non-repudiation property according to the definition stated above. It is worth to note again (c.f. [17]) that attacks on the non-repudiation may expose surprising weaknesses, since it is often a private key owner who plays the role of an adversary here. However, we usually tend to view such a person as the honest user who is a prime target of malicious attackers, and therefore she must be protected using all accessible means.

In the following text, we show a formal definition of the term *k*-collision (§2), then we introduce the notion of GDSA (§3), which is a general construction that encapsulates elementary common algebraic properties of DSA and ECDSA. Besides allowing us to study our subject for both schemes at the same time, it also demonstrates the generality of properties which allow us to search for *k*-collisions effectively. A concrete algorithm for *k*-collision searching is presented in §4 then. Section §5 shows basic limits of *k*-collision computation for (EC)DSA. Use of this observation is made in §6, where general countermeasures are introduced. Some heuristic remarks that did not fit elsewhere are summarized in §7 as possible inspirations for further research. Finally, we conclude in §8.

Proposition 1.1. *Unless stated otherwise, the term "breaking a signature scheme (instance)" means an ability to compute a private key only from publicly known values (i.e. a public key, message signatures, etc.).*

Proposition 1.2. *Unless stated otherwise, the elements of* $\boldsymbol{Z}_n$ *will be regarded as the lowest positive integers representing particular equivalence classes in* $\boldsymbol{Z}_n$.

Proposition 1.3. *In case it is clear from a context, we will shorten a notation of indexed variables according to the following example:* $x_A, x_B \rightarrow_{written\ as}\ x_{A,B}$.

2. *K*-collisions – Definitions

Definition 2.1. *We write as* $(Pub, m, S)_\rho$ *the relation* ρ *saying that S is a valid signature of the message m under the public signature scheme instance Pub.*

The term *public instance* and the structure of *S* are defined in §3. It suffices to understand them heuristically through a general meaning here.

Definition 2.2 (*k*-collision). *The five-tuple* $(Pub_A, m_A, Pub_B, m_B, S)$*, where* Pub_A *and* Pub_B *are public instances of a signature scheme,* m_A *and* m_B *are messages and S is a*

signature, form a k-collision if the following holds: $Pub_A \neq Pub_B$, $(Pub_A, m_A, S)_\rho$ *and* $(Pub_B, m_B, S)_\rho$. *Furthermore the public instances* $Pub_{A,B}$, *messages* $m_{A,B}$, *and their signature S are referred to as k-colliding public instances, k-colliding messages and k-colliding signature respectively.*

Definition 2.3 (1st and 2nd order *k*-collision). *We use the term 1st order k-collision to refer to the k-collision, where* $m_A = m_B$. *We use the term 2nd order k-collision to refer to the k-collision, where* $m_A \neq m_B$.

The sets of all 1st order and 2nd order *k*-collisions decompose the set of all *k*-collisions for a given signature scheme. In this contribution, we study algorithms which can be used for the purpose of *k*-collision searching. The following definition introduces the basic properties of such algorithms.

Definition 2.4 (*k*-collision searching algorithms properties).

i) *We say that an algorithm is non-cooperative if it finds a k-collision for a given public signature scheme instance* Pub_A, *a k-colliding message* m_A, *and a signature S, such that* $(Pub_A, m_A, S)_\rho$, *without needing any further information which is not publicly accessible. We say that the algorithm is cooperative, otherwise.*

ii) *An algorithm for k-collision searching is message-independent if it allows the k-colliding messages* $m_{A,B}$ *to be pre-set to arbitrary strings.*

iii) *An algorithm is proper if it does not disallow respective owners of k-colliding public instances* $Pub_{A,B}$ *to know their relevant private keys. I.e., we may assume that the owner of* Pub_A *knows* $Priv_A$ *and the owner of* Pub_B *knows* $Priv_B$.

Non-cooperative algorithms have a special importance here, since they allow an attacker to do so-called *signature stealing*. Using such an algorithm, the attacker may pretend to have signed an important document which has actually been signed by someone else. The real author of the signature does not have to provide any secret information to the attacker, so the attacker "steals" her signature. There are practical cases in which such an attack brings some benefits to the attacker, for instance, bypassing an authorship proving service.

A message-independent algorithm for *k*-collision searching can be used to search for both 1st order and 2nd order *k*-collisions. Furthermore, it is reasonable to require such an algorithm to be proper in case the judge requests the respective owners of $Pub_{A,B}$ to prove that they know their respective private keys. Therefore, we focus on the design of a proper message-independent non-cooperative algorithm for *k*-collision searching. Such an algorithm is described in §4.

2.1 Illustrative example of a practical attack

The notion of *k*-collisions directly supports the discussion given in [11]. Is has been observed there, that a timestamp of a document signature cannot be, generally speaking, considered as a timestamp of the document itself. To show that, authors

actually demonstrated that 2nd order *k*-collision searching is feasible for RSA. Their method was based on the idea of solving an equation $m^d \bmod n = s$ for d, provided that s and m are given and an attacker can arbitrarily choose the value of n (representing RSA modulus). Authors suggested constructing n in such a way that the discrete logarithm problem is easily solvable. However, this selection would be probably easily visible and would render the whole RSA key useless. In the case of (EC)DSA, we will show that we do not need to make such special constructions, so the colliding keys do not impose obvious marks of an attack and can be deployed for practical signing as well. Therefore, we may regard schemes based on DSA to be somehow more vulnerable to the attack presented in [11] than RSA (c.f. also §7 bellow).

Furthermore, let us imagine the following scenario: There is a scientific conference whose potential participants are requested to submit anonymous papers. To thwart cheating, each researcher must also append a string of her digital signature of the paper. Let us assume that only the signature is appended, not a public key. I.e. using the above-mentioned notation, researcher *A* sends $paper_A$ and *S* from a triplet $(Pub_A, paper_A, S)_\rho$. When the paper is accepted and or a disputation occurs, the researcher must prove that she owns the proper verification/signing keys, i.e. that *S* is her proper signature. This should prevent changing the author and or the content of the paper later on. However, it would not work, when it is possible to construct *k*-collisions. For instance, having a *k*-collision $(Pub_A, paper_A, Pub_B, paper_B, S)$, such that $paper_A = paper_B$, user *B* may pretend to be the author of the paper. Moreover, if it is possible to compute the *k*-collision non-cooperatively, then user *B* can do so without an agreement with user *A*.

What follows is that we cannot take the signature string *S* in itself as a fingerprint of the document content ($paper_A$) and the identity of the signatory (Pub_A). Only the full triplet $(Pub_A, paper_A, S)_\rho$ can be used for such a purpose. In the aforesaid example, we actually did a demonstration of a possible protocol weakness which is analogical to the vulnerability discussed in [11]. Therefore, we may consider the whole notion of *k*-collisions as a platform for generalization of these attacks.

2.2 Another example

In the European Electronic Signature Standardization Initiative [3], there is a term *advanced electronic signature* which is defined in the following way: *electronic signature which meets the following requirements:* ***a) it is uniquely linked to the signatory****; b) it is capable of identifying the signatory; c) it is created using means that the signatory can maintain under his sole control; and d) it is linked to the data to which it relates in such a manner that any subsequent change of the data is detectable (see Directive 1999/93/EC).* It is the point (a) which is very important for us here. Let us have a cryptographic digital signature scheme which is planned to be used for a construction of a particular advanced electronic signature scheme. Note that this is the kind of electronic signature which is commonly understood as a "safe" one through the whole European Union, and therefore almost all applications tend to achieve at least this "level" of an electronic signature scheme. It is also accepted that a general cryptographic digital signature scheme is automatically also an advanced

electronic signature scheme. However, if the digital signature scheme used allows an attacker to construct *k*-collisions, then the electronic signature scheme based on it clearly cannot be called advanced, since the condition in (a) would not be fulfilled. Such a discrepancy can probably lead to tough judicial consequences, and therefore *k*-collisions should not be underestimated here.

3. Generalized DSA

The main purpose of Generalized DSA (GDSA) is to generalize common algebraic properties of DSA [6] and ECDSA [6], [8]. It is introduced here solely for the purpose of developing a general model for dealing with *k*-collisions, which have to give valid results for both DSA and ECDSA. Therefore, the particular security requirements for GDSA based signature schemes, other than those being connected with *k*-collision attacks, are not discussed here. Moreover, we make an effort to keep the GDSA definition algebraically close to the way in which the (EC)DSA schemes are practically realized (here we may differ from the approaches generalizing (EC)DSA from other viewpoints, e.g. [1], [17]).

Definition 3.1 (Generalized DSA – GDSA). *A GDSA instance consists of public parameters, a private key, a public key, and public transformations.*

- *The public parameters are represented as the three-tuple (**P**, n, g), where **P** is a cyclic group, g is a generator of a prime-order subgroup **G** of **P** and n is the order of **G**. Unless stated otherwise, the group operation on **P** will be written in multiplicative notation and the identity element of **P** will be denoted as **id**.*
- *The private key (x) is an integer satisfying $0 < x < n$. The public key (y) is an element of **P**, $y = g^x$.*
- *The public transformations consist of two publicly known mappings denoted as H and φ. H is a hash function, H: $\mathbf{M} \rightarrow H(\mathbf{M})$, where **M** is a set of input messages to be processed. We assume that H implicitly incorporates a string-to-integer conversion, i.e. $H(\mathbf{M}) \subset \mathbf{Z}$.*
- *The purpose of φ is to define a conversion function φ: $\mathbf{P} \rightarrow \mathbf{Z}_n$. Note that since n is a prime, $\mathbf{Z}_n$ is isomorphic to **GF**(n). Our reasoning doesn't depend on the concrete definition of φ here.*
- *We denote the GDSA instance as Inst, Inst = (**P**, n, g, x, y, H, φ). The GDSA instance without the private key x will be referred to as the public GDSA instance (or the public part of GDSA instance) Pub, Pub = (**P**, n, g, y, H, φ).*

Definition 3.2 (Proper GDSA instance). *Let (**P**, n, g, x, y, H, φ) be a GDSA instance. This instance is said to be proper if the following conditions hold:*

i) ***P** has a proper structure – the meaning of this condition depends on the particular kind of GDSA (e.g. **P** must not be built over a weak elliptic curve, n is large enough, etc.)*

ii) *n is a prime, n divides the order of* $\boldsymbol{P}$
iii) $ord(g) = n$
iv) $0 < x < n$
v) $g^x = y$

The purpose of this definition is to summarize general requirements to which a GDSA instance must conform. We will use this definition when showing that a particular generated instance is sound enough to be regarded as a properly working scheme without any obvious marks of an attack.

Definition 3.3 (GDSA signing algorithm). *Let* $(\boldsymbol{P}, n, g, x, y, H, \varphi)$ *be a GDSA instance and let m be a message to be signed. The signing operation then proceeds with the following steps:*

i) *compute the integer h,* $h = H(m)$
ii) *generate a secret random integer k,* $0 < k < n$*; note that k will be referred to as a nonce*
iii) *compute the integer r,* $r = \varphi(g^k)$
iv) *compute the integer s,* $s = (h + rx)k^{-1} \bmod n$*, where* $kk^{-1} \equiv 1 \pmod n$
v) *if either* $r = 0$ *or* $s = 0$*, repeat the whole computation from (ii)*
vi) *the pair (r, s) is the signature of m*

Definition 3.4 (GDSA verifying algorithm). *Let Pub,* $Pub = (\boldsymbol{P}, n, g, y, H, \varphi)$*, be a public GDSA instance and let m be a message, whose signature (r, s) has to be verified. The verifying operation then proceeds with the following steps:*

i) *if either* $r = 0$ *or* $s = 0$*, then the signature is rejected as an invalid one*
ii) *compute the integer h,* $h = H(m)$
iii) *compute the integer* u_1, $u_1 = h*s^{-1} \bmod n$*, where* $ss^{-1} \equiv 1 \pmod n$
iv) *compute the integer* u_2, $u_2 = r*s^{-1} \bmod n$
v) *compute the integer r',* $r' = \varphi(g^{u1}y^{u2})$
vi) *the signature is valid iff* $r' = r$*, i.e.* $(Pub, m, (r,s))_\rho$ *iff* $r' = r$

3.1 DSA

The DSA scheme [6] will be viewed as the GDSA scheme where $\boldsymbol{P}$ is a multiplicative group $\boldsymbol{Z}_p^*$, where p is a prime and $n|(p-1)$. The identity element is $\boldsymbol{id} = [1]_p$. The conversion function φ is defined as the mapping $\varphi: \boldsymbol{Z}_p^* \rightarrow \boldsymbol{Z}_n: a \rightarrow ((a \bmod p) \bmod n)$. The standard [6] currently prescribes the use of the SHA-1 [5] as H. Note that the order of the working prime order subgroup is often ([6]) denoted as q, therefore we adopt this notation in Appendix A.

3.2 ECDSA

The ECDSA scheme [6], [8] will be viewed as the GDSA scheme where $\boldsymbol{P} = E(\boldsymbol{F}_q)$, $\boldsymbol{F}_q$ is a finite field isomorphic to $\boldsymbol{GF}(q)$ and E is a suitable planar elliptic curve over $\boldsymbol{F}_q$. $E(\boldsymbol{F}_q)$ is an abelian group of points on the curve E (together with the special *point*

at infinity denoted as O). The order of $E(\boldsymbol{F}_q)$ is denoted $\#E(\boldsymbol{F}_q)$, so $n|\#E(\boldsymbol{F}_q)$. The identity element is $\boldsymbol{id} = O$. The group operation on $E(\boldsymbol{F}_q)$ is written in additive notation, where a v-times iterated addition of a point A, $A \in E(\boldsymbol{F}_q)$, is denoted as $B = [v]A$, $B \in E(\boldsymbol{F}_q)$. The conversion function φ is the mapping φ: $E(\boldsymbol{F}_q) \rightarrow \boldsymbol{Z}_n$: $A = (x, y) \rightarrow \text{int}(x) \bmod n$, where int($x$) is an integer representation for the x-coordinate of A, $x \in \boldsymbol{F}_q$. The concrete definition of int(.) depends on the way in which the field $\boldsymbol{F}_q$ is constructed. The standard [6] currently prescribes the use of the SHA-1 [5] as H.

4. *K*-collisions for GDSA

In this part we show how to effectively compute k-collisions for the GDSA scheme. The algorithm introduced here is message-independent, non-cooperative, and proper. Therefore, it can be also used for the purpose of signature stealing (c.f. §2).

Theoretically, it might be possible for k-colliding instances to belong to different kinds of GDSA. However, our algorithm presented here is based on that the instances belong to the same kind of GDSA, i.e. they are both either DSA instances or ECDSA instances. This assumption allows us to design the algorithm in an effective way, while it does not bring any practical restriction.

Algorithm 4.1 (Computing a *k*-collision for GDSA).

Input:

- Public GDSA instance $Pub_A = (\boldsymbol{P}_A, n_A, g_A, y_A, H_A, \varphi_A)$, which is a public part of a proper GDSA instance $Inst_A$.
- Message m_A and its signature (r, s), such that $(Pub_A, m_A, (r, s))_{\rho}$.
- Message m_B, which the k-collision is computed for.

Output:

- GDSA instance $Inst_B = (\boldsymbol{P}_B, n_B, g_B, x_B, y_B, H_B, \varphi_B)$.
- k-collision $(Pub_A, m_A, Pub_B, m_B, (r, s))$.

Computation:

i) place $\boldsymbol{P} = \boldsymbol{P}_B = \boldsymbol{P}_A$, $n = n_B = n_A$, $H = H_B = H_A$ and $\varphi = \varphi_B = \varphi_A$
ii) compute the integer h_A, $h_A = H(m_A)$
iii) compute the integer u_1, $u_1 = h_A * s^{-1} \bmod n$, where $ss^{-1} \equiv 1 \pmod n$
iv) compute the integer u_2, $u_2 = r * s^{-1} \bmod n$
v) compute α, $\alpha \in \boldsymbol{P}$, $\alpha = g_A^{u1} y_A^{u2}$
vi) generate a secret random integer z, $0 < z < n$
vii) compute the integer k_B, such that $zk_B \equiv 1 \pmod n$, i.e. $k_B \equiv z^{-1} \pmod n$
viii) compute the integer h_B, $h_B = H(m_B)$
ix) if $k_B s - h_B \equiv 0 \pmod n$ goto (vi)
x) set $x_B = (k_B s - h_B) r^{-1} \bmod n$, where $rr^{-1} \equiv 1 \pmod n$
xi) set $g_B = \alpha^z$
xii) set $y_B = g_B^{xB}$
xiii) if $y_B = y_A$ goto (vi)
xiv) set $Inst_B = (\boldsymbol{P}_B, n_B, g_B, x_B, y_B, H_B, \varphi_B)$, $Pub_B = (\boldsymbol{P}_B, n_B, g_B, y_B, H_B, \varphi_B)$
xv) return $Inst_B$, $(Pub_A, m_A, Pub_B, m_B, (r, s))$

■

Note that setting $\boldsymbol{P}_B = \boldsymbol{P}_A$ (and $n_B = n_A$) should not look suspicious later on (e.g. at the court), because DSA and ECDSA schemes were both developed with the possibility of sharing the group $\boldsymbol{P}$ and its prime order subgroup among many independent users. Sometimes, it is even recommended in the case of ECDSA to use prescribed elliptic curves rather than generating new ones (c.f. curves in [6]).

The value of z generated in step (vi) should be discarded after finishing the computation, since it may allow other attackers to discover the private key x_B and it would serve as an easy proof of a "cooked" generation of $Inst_B$.

Theorems on general properties of the GDSA instance $Inst_B$ produced by algorithm 4.1, together with the main theorem stating that the algorithm produces a k-collision, follow. Unless stated otherwise, these theorems and lemmas presume the validity of the input assumptions of 4.1, and their symbols refer to its input, temporary and output variables. Owners of the instances $Inst_A$ and $Inst_B$ are referred to as the users A and B respectively.

Lemma 4.2 (Tractability of algorithm 4.1). *Algorithm 4.1 is tractable if the signing and verifying algorithms of the particular GDSA are tractable.*

Proof. This algorithm uses the same kind of operations as the signing and verifying algorithms (c.f. definitions 3.3 and 3.4).

■

Lemma 4.3 (On the generator g_B). *For g_B it holds that $g_B = (g_A^{kA})^z$, where k_A is the nonce used by the user A for the (r, s) computation. Furthermore, we have $g_B^{kB} = g_A^{kA}$ and $\varphi(g_B^{kB}) = \varphi(g_A^{kA}) = r$.*

Proof. The equation $g_B = (g_A^{kA})^z$ follows immediately from step (xi) of algorithm 4.1 and the assumption of $(Pub_A, m_A, (r, s))_\rho$. Also following are $g_B^{kB} = g_A^{kA}$ and $r = \varphi(g_B^{kB})$, since $zk_B \equiv 1 \pmod n$ and n is the order of g_A.

■

Lemma 4.4 (On the values of $x_{A,B}$ and $y_{A,B}$). *It holds that:*

i) $x_B = [(k_A^{-1}k_Bh_A - h_B)r^{-1} + k_A^{-1}k_Bx_A] \bmod n$

ii) $x_A = [(k_Ak_B^{-1}h_B - h_A)r^{-1} + k_Ak_B^{-1}x_B] \bmod n$

iii) $y_B = g_A^{\gamma}$, *for* $\gamma \equiv (h_A - k_Ak_B^{-1}h_B)r^{-1} + x_A \pmod n$

iv) $y_A = g_B^{\lambda}$, *for* $\lambda \equiv (h_B - k_A^{-1}k_Bh_A)r^{-1} + x_B \pmod n$

Proof.

i) According to step (x) of algorithm 4.1, we have $x_B = (k_Bs - h_B)r^{-1} \bmod n$. We assume that (r, s) is a valid signature of m_A computed according to definition 3.3. From here we get $s = (h_A + rx_A)k_A^{-1} \bmod n$, where $k_Ak_A^{-1} \equiv 1 \pmod n$ and k_A is a random integer, $0 < k_A < n$. Substituting this expression into the equation for x_B we have $x_B = [(k_A^{-1}k_Bh_A - h_B)r^{-1} + k_A^{-1}k_Bx_A] \bmod n$.

ii) Follows directly from (i).

iii) Using (i), lemma 4.3 and the equation from step (xii) of 4.1 we get the equation for y_B.

iv) Analogically as (iii).

■

Theorem 4.5 (On termination). *Algorithm 4.1 terminates.*

Proof. There are only two loops in 4.1 and these are in steps (ix) and (xiii). The loop in step (ix) acts if and only if $k_B s - h_B \equiv 0 \pmod{n}$. Since $\gcd(s, n) = 1$, there is exactly one value of k_B, $0 < k_B < n$, satisfying this congruence. Therefore, an infinite loop does not occur here for randomly chosen values of k_B. The second loop is in step (xiii) and it acts if and only if $y_B = y_A$. Using lemma 4.4 we can rewrite this condition as $y_A = g_A^{\gamma}$, for $\gamma \equiv (h_A - k_A k_B^{-1} h_B) r^{-1} + x_A \pmod{n}$, where k_A is a random number, $0 < k_A < n$, which is fixed for all loops through algorithm 4.1. This condition holds if and only if $(h_A - k_A k_B^{-1} h_B) r^{-1} \equiv 0 \pmod{n}$. Again, there is only one value of k_B which satisfies this congruence. Therefore, this loop is finite for randomly chosen values of k_B.

■

Theorem 4.6 (Properness of $Inst_B$). *The GDSA instance $Inst_B$ computed by algorithm 4.1 is proper.*

Proof. Let us check the conditions (i - v) required by definition 3.2:

- i-ii) These conditions are fulfilled according to step (i) of algorithm 4.1. We rely on the assumption that $Inst_A$ is proper, and therefore $\boldsymbol{P}_A$ must have a proper structure, and n must divide the order of $\boldsymbol{P}_A$.
- iii) Recall that $n = n_B = n_A$ and lemma 4.3 above. Since we assume that the instance $Inst_A$ is proper, it holds that $\mathrm{ord}(g_A) = n$, where n is a prime. Because of n being a prime and $k_A, z < n$ it follows that $\gcd(n, k_A z) = 1$. Therefore $\mathrm{ord}(g_B) = \mathrm{ord}(g_A) = n$.
- iv) Since x_B is a result of operation modulo n it trivially holds that $0 \le x_B < n$. It remains to check that $x_B \neq 0$. Let us suppose that $x_B = 0$. Since $\gcd(r^{-1}, n) = 1$ and $x_B = (k_B s - h_B) r^{-1} \bmod n$, this equation holds if and only if $k_B s - h_B \equiv 0 \pmod{n}$. However, this is prevented by step (ix). Therefore $0 < x_B < n$.
- v) It follows directly from step (xii) of algorithm 4.1.

■

Theorem 4.7 (Algorithm 4.1 produces a *k*-collision). *The five-tuple* $(Pub_A, m_A, Pub_B, m_B, (r, s))$ *computed by algorithm 4.1, is a k-collision. Furthermore, the algorithm is a message-independent non-cooperative and proper one.*

Proof. We assume that $(Pub_A, m_A, (r, s))_{\rho}$. It remains to show that also $(Pub_B, m_B, (r, s))_{\rho}$ and $Pub_A \neq Pub_B$. We use verifying algorithm 3.4 for m_B and Pub_B at first. In the steps (i - iii) of 3.4 we obtain $u_1 = h_B s^{-1} \bmod n$, $u_2 = r s^{-1} \bmod n$, where $s s^{-1} \equiv 1 \pmod{n}$. In step (v) of 3.4 we get $r' = \varphi(g_B^{u1} y_B^{u2}) = \varphi(g_B^{\omega})$, where $\omega \equiv h_B s^{-1} + x_B r s^{-1} \pmod{n}$. From step (x) of 4.1 we have $x_B = (k_B s - h_B) r^{-1} \bmod n$. Substituting this value to the congruence for ω we get $\omega \equiv h_B s^{-1} + k_B - h_B s^{-1} \equiv k_B \pmod{n}$, so $r' = \varphi(g_B^{kB})$. According to lemma 4.3 we get $r' = r$. Therefore, the signature (r, s) is valid. Furthermore, according to the condition in step (xiii) of 4.1, the public instances Pub_A and Pub_B differ at least in the values of public keys y_A and y_B, therefore, $(Pub_A, m_A, Pub_B, m_B, (r, s))$ is a k-collision. Moreover, there is neither a restriction for messages $m_{A,B}$ nor a need for a cooperation with the user A. Therefore, this algorithm is message-independent and non-cooperative. Since the validity of the A's private key is left

intact and the *B*'s private key is computed in step (x) of 4.1, this algorithm is also proper.

■

Theorem 4.8 (On attacker's private key secrecy). *Unless the particular realization of GDSA can be broken (c.f. proposition 1.1), there is a negligible probability that the user A is able to break B's instance* $Inst_B$.

Proof. If the particular GDSA realization (e.g. DSA, or ECDSA) cannot be broken, then there is no way for an ordinary user to break someone else's instance. However, we shall check if the special construction of $Inst_B$ used in algorithm 4.1 helps the users *A* and *B* to break each other's instances or not. At first, we observe that the users *A* and *B* play symmetric roles in our scheme: Both of them may regard her GDSA instance as the first one for which the second user has computed her *k*-colliding instance. This symmetry can be seen from lemma 4.4. Therefore, if there is a way for the user *A* to discover the *B*'s private key, then there is also a way for the user *B* to discover *A*'s private key. Because there is no need for a cooperation between *A* and *B*, it follows that the user *B* could break the *A*'s GDSA instance simply by computing an appropriate *k*-collision. If we assume that breaking the particular GDSA scheme is hard, then there must be a negligible probability that the appropriate *k*-collision would be found using algorithm 4.1. Therefore, the construction used in 4.1 cannot practically allow the user *B* to break the *A*'s instance. From the symmetry observed above, it follows that the construction does not practically allow the user *A* to break *B*'s instance.

■

5. Basic Limits for General *K*-collision Searching Algorithms

In §4 we have seen an effective algorithm for *k*-collision searching. However, there are other approaches to this problem. The aim of this paragraph is to show basic limits for general *k*-collision searching algorithms in GDSA schemes. It will help us when designing appropriate general countermeasures in §6.

Definition 5.1. *Let us denote* $\varphi^{|\mathbf{G}}$ *the GDSA (c.f. def. 3.1) conversion function* φ *restricted on* $\mathbf{G}$*, where* $\mathbf{G}$ *is the prime-order subgroup of* $\mathbf{P}$ *generated by g.*

It has been shown (c.f. [1], [13]), that $\varphi^{|\mathbf{G}}$ is an almost-bijective mapping. Moreover, we conjecture the following hypothesis for secure GDSA instances.

Hypothesis 5.2 (Inner collision resistance). *For randomly chosen pairs* (g_1, g_2)*, such that* $g_{1,2} \in \mathbf{P}$*,* $ord(g_1) = ord(g_2) = n$*, there is no tractable algorithm producing integers v, w, such that* $\varphi^{|\mathbf{G}}(g_1^{\,v}) = \varphi^{|\mathbf{G}}(g_2^{\,w})$*, while* $g_1^{\,v} \neq g_2^{\,w}$.

Theorem 5.3 (Limiting theorem). *Unless GDSA schemes can be broken, there is no feasible proper k-collision searching algorithm producing k-collision* $(Pub_A, m_A, Pub_B, m_B, (r, s))$ *for* $Pub_{A,B}$*, such that* $Pub_A = (\mathbf{P}, n, g_A, y_A, H, \varphi)$ *and* $Pub_B = (\mathbf{P}, n, g_B, y_B, H, \varphi)$*, where* $g_{A,B}$ *are* <u>*arbitrary given generators*</u>.

Proof (assuming validity of 5.2). Obviously, the GDSA instance $Inst = (\boldsymbol{P}, n, g, x, y, H, \varphi)$ can be broken if we can solve the discrete logarithm problem (DLP) according to the generator g effectively. We show how a general proper k-collision searching algorithm can be used for solving the DLP. Let us assume that we want to solve the DLP on $\boldsymbol{P}$ with the base g for the public key y to get the private key x. In the first step, we choose temporary helping GDSA instances $Inst_A$ and $Inst_B$, such that $Inst_A = (\boldsymbol{P}, n, g, x_A, y_A, H, \varphi)$, $Inst_B = (\boldsymbol{P}, n, y, x_B, y_B, H, \varphi)$, where $x_{A,B}$ are arbitrarily chosen private keys and $y_{A,B}$ are the appropriate public keys. Note that $Inst_B$ uses the public key y in the place of its generator g_B. Also, note that $x_{A,B}$ may be fixed later on, just during the k-collision searching process. We only assume that these values are known then (i.e. the algorithm is proper). Now we start searching to find two k-colliding messages $m_{A,B}$, and a k-colliding signature (r, s). Since we know $x_{A,B}$ then, we can compute particular nonces $k_{A,B}$ for $m_{A,B}$ and (r, s) easily. It holds that $r = \varphi^{|\boldsymbol{G}}(g^{kA}) = \varphi^{|\boldsymbol{G}}(y^{kB})$. Using the properties of $\varphi^{|\boldsymbol{G}}$ stated above, we can rewrite this as $g^{kA} = y^{kB}$ with a high probability. Since n is the order of g and $y = g^x$, we have $k_A \equiv xk_B \pmod{n}$. From here we can compute the value of x easily, which means that our attack on x is finished.

Hypothesis 5.2 tells us that the general algorithm used above cannot rely on inner collisions of $\varphi^{|\boldsymbol{G}}$. Although there may still be singular problems with these collisions, they should appear as often as if $\varphi^{|\boldsymbol{G}}$ was a pseudorandom function. Such rare exceptions may then be easily overcome by repeating the whole process several times, using different values of $g_{A,j}$ and $g_{B,j}$ in each pass j. For instance, we can set $g_{A,j} = g^j$, $g_{B,j} = y^j$ for $j \in \langle 1, n)$. This construction leads to $jk_{A,j} \equiv jxk_{B,j} \pmod{n}$ then, so still $k_{A,j} \equiv xk_{B,j} \pmod{n}$.

■

6. Countermeasures

6.1 Basic Reasoning

Theorem 5.3 tells us that even if we place no restrictions on the public and private keys (except that we want to know these respective keys) and set no rules for the k-colliding messages and the signature, the k-colliding instances $Pub_{A,B}$ still cannot have their remaining public parameters set to arbitrary values. Their choices must in some way be dependent to allow an attacker to compute a k-collision. It requires further research to say how far this dependency goes, but we can conclude heuristically that the dependency is probably stronger than the necessity to omit the setup used in the premise of theorem 5.3.

Corollary 6.1 (Hypothesis on dependency). *If the particular GDSA scheme cannot be broken, then there is no proper k-collision searching algorithm that allows the public parameters of k-colliding instances to be chosen independently.*

6.2 Online Protocol

Basing on corollary 6.1, we propose the idea of a simple but strong countermeasure: All public parameters for GDSA schemes must be *independently* generated by a trusted third party *on a per-instance basis*. The authority shall also issue a certificate of proper GDSA instance generation. For example, the scenario under which users generate their GDSA instances and requests for a public key certificate should be like this one:

i. **user -> certification authority:** *certification_request_start*
ii. **certification authority -> user:** *public_ parameters_(**P**, n, g)*, *token*
iii. user chooses a private key and computes the public key
iv. **user -> certification authority:** *public_key*, *possession_proof*, *token*

We assume the conversion function φ to be implicitly set according to the particular kind of GDSA. The token introduced in steps (ii) and (iv) helps the certification authority ensure that the particular public parameters generated are used only by one user. The possession_proof introduced in step (iv) serves as a proof of private key possession. It has to ensure that whatever k-collision searching algorithm should be used, it must be a proper one (c.f. definition 2.4 and corollary 6.1). The public key certificate issued according to the protocol written above also serves as the certificate of a proper GDSA instance generation.

Note that in the case of ECDSA, this setup may be attacked by the inner m-collisions presented in [17, §4.2 – Duplicate Signatures]. However, disclosing such an m-collision leads to the private key disclosure and therefore it is not considered a security weakness ([17]). If we assume that the certification authority is honest, then the inner m-collisions of DSA described in [18] are successfully defeated by this setup.

The proposed protocol may also be used in a situation when users need to share a common group $\boldsymbol{P}$ and its prime-order subgroup (e.g. they all want to use the same standardized elliptic curve). In such a case, the authority independently generates for each request a new value of the generator g only, while the remaining public parameters ($\boldsymbol{P}$, n) stay constant. The impossibility of k-collision computing then follows directly from theorem 5.3.

6.3 Non-invertible GDSA Setup

It might be useful to substitute the above-mentioned protocol by a non-invertible verifiable GDSA setup. Such a setup may be regarded as an extension of the DSA and ECDSA setups currently defined in [6], [8]. One idea could be to extend the concept of "seeded" prime number (DSA case) searching or elliptic curve (ECDSA case) construction to also cover the subgroup generator g. This item remains unprotected by those mechanisms currently and it has already been shown ([18]) that it is no good. The value of SEED used to initialize the setup would then serve as the certificate of proper instance generation. This improving step would help to reduce the risk of k-

collisions prominently. However, there still remains a possibility of generating a 2nd order *k*-collision cooperatively (the main problem here is that users are not forced to choose their public parameters independently). Let $Inst_A = (\boldsymbol{P}, n, g, x_A, y_A, H, \varphi)$ be a GDSA instance generated properly (e.g. certified by the SEED) and let (m_A, m_B) be two different messages for which users *A* and *B* want to find a *k*-collision. Now, the user *A* computes the signature (r, s) of m_A in $Inst_A$ normally, while she keeps the value of the nonce *k*. This value is passed to the user *B* then, together with the signature (r, s) and the public part of $Inst_A$. The user *B* then constructs $Inst_B = (\boldsymbol{P}, n, g, x_B, y_B, H, \varphi)$, where the private key x_B is computed from the congruence $s \equiv (H(m_B) + x_B r)k^{-1} \pmod{n}$. It is easy to see that $(Pub_A, m_A, Pub_B, m_B, (r, s))$ is the 2nd order *k*-collision. The drawback of this process is that it requires the cooperation between the users who then know each other's private keys. Moreover, it allows a third party seeing this *k*-collision to discover the linear relation between $x_{A,B}$ as $x_A - x_B \equiv r^{-1}(H(m_B) - H(m_A)) \pmod{n}$. On the other hand, it clearly breaks the countermeasure based on the simple setup extension. It follows that there is a need for a broader extension of the current standard – it shall also cover the generation of the private key. Moreover, the certificate of the proper private key generation shall be verifiable, without disclosing any secret information about the key. It remains an open research question on how to do such an extension securely.

6.4 Notary Services and-or Authentication of Public Instances

Another kind of countermeasure may be deployed in systems which use some kind of notary services (for overview on notary services see [12]). It may be feasible in such a situation to require every signed document to be over-signed by a trusted third party. The signature should cover: the message, its (primary) signature, and the public part of the GDSA instance which shall be used for the (primary) signature verification. Similar techniques may also be based on a time stamping service.

It might be also tempting to propose the following countermeasure: request users to sign not only the message, but also the public part of GDSA instance. The signature should then cover the string $m||public\_instance$ instead of plain *m*. However, this countermeasure seems to have several weaknesses, at least from a theoretical point of view. It still leaves a possibility for an attacker to claim that, due to an error, the *public_instance* was appended badly or even it was not appended at all. Furthermore, there is a threat of cooperative 2nd order *k*-collisions which should be investigated to devise an acceptable proof of security. So far, it is known that it is not enough to hash *m* together with the public key only. The whole set of public parameters must be added, too. There is still an open question of how far this countermeasure is affected by the properties of the conversion function φ, especially by its invertibility. Moreover, this countermeasure affects data formats and the behavior of all client applications, in contrast with the protocol proposed in §6.2 above, which only needs the extension of the certificate request process.

7. Closing Remarks

We stress that *k*-collisions are not only a problem for DSA and ECDSA. The same problem may be studied in the RSA [14], ElGamal [2] and Schnorr [16] signature schemes (as well as in the others). Of course, there is a difference in how feasible and conspicuous these attacks are. Feasibility says whether it is possible to do such an attack, while the second criterion tells us if it is easy to recognize marks of the attack later. The current state of research shows that the DSA and ECDSA schemes belong to the group where *k*-collision attacks are feasible and do not leave special marks on *k*-colliding instances. ElGamal and Schnorr schemes probably belong to the same group, since they share those general algebraic properties which were used for our attack. However, both of them introduce certain properties which induce limitations. For example, the algorithm presented here can produce 1st order *k*-collisions only, when applied (and adjusted) on Schnorr scheme (due to binding of the message m being signed and the value of r, $r = g^k \bmod p$, as $e = H(m\|r)$, where the value of e becomes a part of the signature, c.f. [16, p. 168]). Therefore, we may conclude that these schemes come with some (maybe planned or unplanned) built-in countermeasures which are (however) not strong enough to defeat all these attacks. On the other hand, RSA seems to belong to the group where these attacks are still feasible [15], but they *do* produce special marks which could be used by a third party to even break one of the *k*-colliding instances.

8. Conclusion

We have introduced the notion of key-collisions in signature schemes, which may be regarded as a kind of attack parallel to well known message-collisions. Both of them share the common idea of an *alternative explanation* saying why the judge has a message and its (mathematically) valid signature when the user swears that she did not sign it. Instead of claiming that it was a *different message* from what has been signed in reality, an attack based on key-collisions leads to claiming that it was a *different user* who has signed the message or even who has signed a different message. Next, we presented a (trivially) feasible algorithm for key-collision searching in DSA and ECDSA, which is message-independent and does not require any cooperation between the owners of colliding instances. Therefore, an attacker may use this algorithm to steal a signature of another user. The effectiveness of the algorithm comes mainly from the ability of the attacker to generate (EC)DSA instances with a relatively high degree of freedom. We showed that even the legitimate owner of the key should not have the ability to generate her key completely at her will without having to be able to present a proof of its honest creation.

It was shown in §5, that unless DSA and ECDSA schemes can be broken, possibilities for key-collision searching in these respective schemes can be prevented. We have developed a general countermeasure based on the simple online procedure for a key generation (§6.2). When deployed in an existing information system, it only changes a certificate request process. Other processes and data structures remain

unchanged. Several other possible countermeasures were proposed and discussed, too (§6.3, §6.4).

References

1. Brown, D.-R.-L.: *Generic Groups, Collision Resistance, and ECDSA*, IEEE 1363, February 2002 (c.f. [7])
2. ElGamal, T.: *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory, Vol. 31, pp. 469–472, IEEE, 1985
3. *EESSI - The European Electronic Signature Standardization Initiative*, c.f. the homepage at http://www.ict.etsi.org/eessi/EESSI-homepage.htm
4. *E-SIGN - The Electronic Signatures in Global and National Commerce Act*, enacted on June 30, 2000, c.f. http://www.cybercrime.gov/esign.htm
5. FIPS PUB 180-1: *Secure Hash Standard (SHA-1)*, National Institute of Standards and Technology, January 2001
6. FIPS PUB 186-2: *Digital Signature Standard (DSS)*, National Institute of Standards and Technology, January 27, 2000, update: October 5, 2001
7. IEEE P1363: *Standard Specifications for Public Key Cryptography*, August 1998. c.f. http://grouper.ieee.org/groups/1363
8. Johnson, D., Menezes, A.-J., and Vanstone, S.-A.: *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, International Journal of Information Security, Vol 1, Issue 1, pp. 36-63, Springer-Verlag, 2001
9. Kiayias, A. and Yung, M.: *Breaking and Repairing Asymmetric Public-Key Traitor Tracing*, in Proc. of the 2002 ACM Workshop on Digital Rights Management, 2002
10. Landwehr, C.-E.: *Computer Security*, International Journal of Information Security, Vol 1, Issue 1, pp. 3-13, Springer-Verlag, 2001
11. Massias, H., Serret Avila, X., and Quisquater, J.-J.: *Timestamps: Main issues on their use and implementation*, In Proc. of IEEE 8th International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises-Fourth International Workshop on Enterprise Security, pp. 178-183, June 1999
12. Menezes, A.-J., van Oorschot, P.-C., and Vanstone, S.-A.: *Handbook of Applied Cryptography*, CRC Press, 1996
13. Nguyen, P.-Q. and Shparlinski, I.-E.: *The Insecurity of the Digital Signature Algorithm with Partially Known Nonces*, Journal of Cryptology, Vol. 15, 3/2002, pp. 151-176, Springer-Verlag, 2002
14. Rivest, R.-L., Shamir, A., and Adleman, L.: *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, pp. 120-126, 1978
15. Rosa, T.: *On Key-collisions in Signature Schemes*, In proc. of the workshop VKB 2002 (Czech language), pp. 14-26, Brno, April 3.-4., 2002
16. Schnorr, C.-P.: *Efficient Signature Generation by Smart Cards*, Journal of Cryptology, Vol. 4, pp. 161–174, Springer-Verlag, 1991
17. Stern, J., Pointcheval, D., Malone-Lee, J., and Smart, N.-P.: *Flaws in Applying Proof Methodologies to Signature Schemes*, in Proc. of CRYPTO 2002, LNCS 2442, pp. 93-110, Springer-Verlag, 2002
18. Vaudenay, S.: *Hidden Collisions on DSS*, in Proc. of CRYPTO '96, pp. 83-88, Springer-Verlag, 1996

Appendix A: Algorithm 4.1 edited for DSA and ECDSA

The algorithm for effective k-collision searching (c.f. §4) edited for the DSA and ECDSA schemes is presented here. It should be a helpful illustration of how the general reasoning done for the GDSA model (c.f. §3, §4) transforms back to these particular schemes.

Notes: The conversion function φ together with the hash function H (SHA-1) are implicitly defined for the DSA and ECDSA schemes (c.f. §3.1, §3.2). Therefore we omit them from the notation of the (EC)DSA instances here. Furthermore, we use the prime p alone when referring to the multiplicative group $\boldsymbol{P} = \boldsymbol{Z}_p^*$ in the case of DSA.

Algorithm DSA-4.1 (Computing a *k*-collision for DSA).

Input:

- Public DSA instance $Pub_A = (p_A, q_A, g_A, y_A)$.
- Message m_A and its signature (r, s), such that $(Pub_A, m_A, (r, s))_{\rho}$.
- Message m_B, which the k-collision is computed for.

Output:

- DSA instance $Inst_B = (p_B, q_B, g_B, x_B, y_B)$.
- k-collision $(Pub_A, m_A, Pub_B, m_B, (r, s))$.

Computation:

i) place $p = p_B = p_A$, $q = q_B = q_A$
ii) compute the integer h_A, $h_A = \text{SHA-1}(m_A)$
iii) compute the integer u_1, $u_1 = h_A * s^{-1} \bmod q$, where $ss^{-1} \equiv 1 \pmod q$
iv) compute the integer u_2, $u_2 = r * s^{-1} \bmod q$
v) compute α, $\alpha = g_A^{u1} y_A^{u2} \bmod p$
vi) generate a secret random integer z, $0 < z < q$
vii) compute the integer k_B, $zk_B \equiv 1 \pmod q$, i.e. $k_B \equiv z^{-1} \pmod q$
viii) compute the integer h_B, $h_B = \text{SHA-1}(m_B)$
ix) if $(k_B s - h_B) \equiv 0 \pmod q$ goto (vi)
x) set $x_B = (k_B s - h_B) r^{-1} \bmod q$, where $rr^{-1} \equiv 1 \pmod q$
xi) set $g_B = \alpha^z \bmod p$
xii) set $y_B = g_B^{xB} \bmod p$
xiii) if $y_B = y_A$ goto (vi)
xiv) set $Inst_B = (p_B, q_B, g_B, x_B, y_B)$, $Pub_B = (p_B, q_B, g_B, y_B)$
xv) return $Inst_B$, $(Pub_A, m_A, Pub_B, m_B, (r, s))$

■

Algorithm ECDSA-4.1 (Computing a *k*-collision for ECDSA).

Input:

- Public ECDSA instance $Pub_A = (E(\boldsymbol{F}_q)_A, n_A, G_A, Y_A)$.
- Message m_A and its signature (r, s), such that $(Pub_A, m_A, (r, s))_{\rho}$.
- Message m_B, which the *k*-collision is computed for.

Output:

- ECDSA instance $Inst_B = (E(\boldsymbol{F}_q)_B, n_B, G_B, x_B, Y_B)$.
- *k*-collision $(Pub_A, m_A, Pub_B, m_B, (r, s))$.

Computation:

i) place $E(\boldsymbol{F}_q)_B = E(\boldsymbol{F}_q)_A$, $n = n_B = n_A$

ii) compute the integer h_A, $h_A = \text{SHA-1}(m_A)$

iii) compute the integer u_1, $u_1 = h_A * s^{-1} \bmod n$, where $ss^{-1} \equiv 1 \pmod n$

iv) compute the integer u_2, $u_2 = r * s^{-1} \bmod n$

v) compute α, $\alpha = [u_1]G_A + [u_2]Y_A$

vi) generate a secret random integer z, $0 < z < n$

vii) compute the integer k_B, $zk_B \equiv 1 \pmod n$ i.e. $k_B \equiv z^{-1} \pmod n$

viii) compute the integer h_B, $h_B = \text{SHA-1}(m_B)$

ix) if $(k_B s - h_B) \equiv 0 \pmod n$ goto (vi)

x) set $x_B = (k_B s - h_B) r^{-1} \bmod n$, where $rr^{-1} \equiv 1 \pmod n$

xi) set $G_B = [z]\alpha$

xii) set $Y_B = [x_B]G_B$

xiii) if $Y_B = Y_A$ goto (vi)

xiv) set $Inst_B = (E(\boldsymbol{F}_q)_B, n_B, G_B, x_B, Y_B)$, $Pub_B = (E(\boldsymbol{F}_q)_B, n_B, G_B, Y_B)$

xv) return $Inst_B$, $(Pub_A, m_A, Pub_B, m_B, (r, s))$

■