Applying Symmetric Encryption
Technical Report

Version 1.2

Falko Strenzke
cryptosource GmbH, Darmstadt
`fstrenzke@cryptosource.de`
`www.cryptosource.de`

October 28, 2016

This work, which is intended to close a gap left by basically all text books on cryptography, shall serve as guidance for the design and implementation of basic symmetric encryption schemes, the challenges of which are often underestimated as recurring reports of failures in this field show. It points out the main pitfalls of such an undertaking, specifically it explains the importance Authenticated Encryption and presents recently published attacks against long known vulnerabilities in symmetric encryption schemes. Attacks and countermeasures are explained with the aim of giving application and protocol designers the knowledge of identifying and preventing such fundamental vulnerabilities in their constructions and implementations.

Contents

"Expect poison from the standing water"
– William Blake, The Marriage of Heaven and Hell

Availability

This document is available at `www.cryptosource.de/applying_symmetric_encryption.pdf`.

Disclaimer

The information, explanations and advices given in this document were verified and are correct to the best of the author's knowledge. However, the author takes no responsibility for the correctness of the contents of this document. All warranties, expressed or implied, are disclaimed, including without limitation, any and all warranties concerning the accuracy of the contents including its fitness or appropriateness for a particular use. Furthermore, the author shall not be liable for any and all damage, direct or indirect, arising from or relating to any use of the contents contained in this document.

1. Introduction

Today, information processing systems are employed in a vast and growing range of applications. Besides the internet, they have reached systems such as cars and critical infrastructures controlling the essential supply of the technologically high developed civilizations. For the protection of the data that is inevitably stored or exchanged in the context of these applications, cryptography is essential. The main goals of its employment are achieving confidentiality and authenticity of data. To this end, symmetric cryptography still plays the major role compared to public key cryptography. The reason for this is that even where public key cryptography is used, it is usually accompanied by symmetric cryptography, which in these cases forms the lower layers for encryption and authentication. Examples are hybrid encryption schemes, i.e. the public key encryption of a symmetric key which, in turn encrypts the payload data, and symmetric session keys agreed upon through a public key scheme. Errors in the specification or implementation of the symmetric layer threaten the security of the combined scheme.

However, the story of the application of symmetric cryptography is not only a story of great successes, but also of great failures. Besides the failures in primitives, e.g. ciphers and hash functions, that turned out to be too weak and were and are kept in use far beyond the point where replacement was indicated as mandatory by scientific progress, there is a great source of security problems stemming from the incorrect use of the symmetric primitives, leading to information leakage vulnerabilities. But there are, in our point of view, fundamental differences with respect to the reasons for the occurrence of either type of failures in the application of symmetric cryptography: the use of weak and outdated primitives is mainly due to cost considerations. Any company or institution will have to make a cost and risk analysis whenever scientific progress suggests dangers in the application the currently implemented technical solutions and replacement of these solutions is considered. Often, as it seems, the result of these analyses in the field of cryptographic schemes is that it is cheaper to keep the endangered techniques; indeed, often there must occur a concrete malicious attack in the field before the outdated schemes are put out of use. Unsatisfying as this situation may be, especially with the often unclear impact on risks and costs resulting for end consumers – contrary to those of the responsible companies these are not averaged for the single end user – the reasons for these problems and the necessary countermeasures, i.e. the enactment of laws and regulations that enforce the use of secure algorithms, are clear. For the second type of failures however, the picture is very different: In the design of new devices, frameworks and applications, the employment of advanced primitives like AES is often not a problem. Yet still, in the design of protocols and application of cryptographic primitives the same mistakes are made over and over again. Sometimes, even a large audience of potential security reviewers does not prevent fundamental errors from infiltrating the field for considerable amount of time. A good example for this is the error with unauthenticated or improperly authenticated encryption (this is one of the topics that will occupy a large part of this work) in the XML encryption standard [1]. To a much larger fraction, this type of failures does not result from consciously holding on to a legacy solution for the reasons discussed above, but from plain lack of knowledge. In our opinion, one of the main reasons for this is the text book situation on application of cryptography. While basically any text book on cryptography generously explains the inner workings of ciphers, security of RSA, and many other basic elements of cryptographic theory, they generally do not mention important risks in the application of cryptographic schemes such as those arising from missing or incorrectly implemented Authenticated Encryption. Our critique is not that these text books are faulty. The main point that we criticise is the missing distinction between the theory of cryptographic primitives on the one side and their correct application on the other side. As a matter of fact, only very few people need to understand the inner workings of the ciphers or public key schemes that they are using, because fortunately a large number of software implementations for basically any conceivable hardware platform is freely available on the internet. What they need to know, is how to build a secure encryption scheme from primitives. Concerning public key cryptography, the same class of people needs to know what CCA2 conversion to employ and what key parameters to use with RSA, but they need not have to know much about the underlying reasons. This calls for a text book on best practices in cryptography, where the term "best practices" has to be used with care: for the cases we have in mind, disregarding these practices often results drastic security problems. But unfortunately, to the best of our knowledge, such a book does not yet exist.

The following text is not the book the lack of which we claim above. However, it addresses the main points in this context for the case of symmetric cryptography, and only for achieving confidentiality and authenticity of messages, which is the most fundamental task of cryptography that suffices for a large number of applications. It is not very convenient in the sense that it presumes some basic knowledge in basics of cryptography and its terminology, for the concrete specifications of the schemes it refers the reader to the respective standards, and also does not cover them in full breadth. However, it addresses the most prevalent sources of errors in protocols using symmetric encryption. As it clearly aims to give the reader the understanding of how to achieve secure cryptographic protocols and implementations, it does not care for the concrete exploitability of the respective failures it mentions in specific application contexts, though these might be very limited in the works which devised them. It would be a false understanding of IT security to think that this latter circumstance means any kind of relaxation for the design of new protocols. If a cryptographic scheme is specified incorrectly, and the security impact of this failure are not thoroughly analysed, then no statement can be made about the security of the system. Designing correct schemes and protocols is in general much more cost effective than a security analysis of the impact of their faults in a concrete application, which, furthermore, would have to be repeated upon any change in that application.

The main focus of the document clearly lays on need of correctly implemented authenticated encryption for purposes of confidentiality. Simple as this topic may seem, its omission from basically all text books and its continuing appearance as a source of dramatic security failures make it outstanding from the other failures this work intends to help to prevent.

2. The most important Issues with the Application of Symmetric Encrytion

This section summarizes the most important security aspects of symmetric encryption schemes, that when ignored in many cases will introduce severe security flaws into a system.

2.1. Limitations of Cryptography

The first thing one has to consider is that encryption of messages conceals the actual content, but not that a message is sent and not the length of that message. While the former aspect is trivial and plays a role basically only for privacy issues, e.g. anonymous surfing on the internet, the problems that might result from the latter might be overlooked in some cases, especially when transformations like compression precede the encryption, this is addressed in Section 6.2.

A simple example can corroborate how the message length can give valid information to an adversary: assume that a central server sends messages encrypted with a symmetric encryption scheme that preserves the byte length of the message (a realistic assumption) to a large number of recipients. Assume that a six digit secret is encoded as a number with a leading length byte followed by the numeric value without leading zeros. This would, especially if the other message parts are constant in length, reveal to the adversary the number of leading zeros in the secret just from observing the lengths of the ciphertexts. Given hat these secrets are chosen randomly, he might identify those users with secrets of the form 0000XY, 00000Y, or 000000, clearly giving him a great advantage when performing on-line attacks which incur the guessing of these secrets as authentication tokens.

This example seems contrived and this type of vulnerability trivial to avoid. Note, however, that often in complex software systems different layers of information processing work together, and that without a thorough security review no single person might have an overview of their synergetic effects.

Thus in frameworks where for the designer of the cryptographic protocol it is not clear what kind of data in what format will be send over the encrypted channel, as a general countermeasure, it could be considered to add a string of few bytes of random length to each message.

2.2. Encryption is (almost) never enough

Merely symmetrically encrypting a message is not sufficient for two reasons. The first one is that using a block cipher in any mode like CBC, ECB, CTR, CFB, OFB (Block Cipher Modes of Operation, which we abbreviate as BCM, are explained shortly) or any Stream Cipher does not provide any kind of integrity protection. This means that an attacker is able to perform manipulations on the ciphertext which lead to modified plaintext. With most modes, specifically whith all of the above mentioned except for ECB, it is even possible to provoke the flipping of plaintext bits in chosen positions. Allowing an attacker to alter plaintexts is in general an undesirable feature, so it is obvious that some sort of integrity protection is needed as well.

But there is a second reason for the need for integrity protection in symmetric encryption schemes: The ability of the attacker to introduce modifications in the plaintext can also be used for attacks in which information about the plaintext is obtained. This is a threat when the attacker can feed manipulated ciphertexts into the decryption device and get some kind of information, for instance error messages, about the decryption process. Then he can potentially gain information about the plaintext. This type of attack is usually referred to as decryption oracle attack, i.e. the decryption device acts as an oracle. It is also important to know that it is possible to implement a means of authentication or integrity protection that suffices for the first reason, but not for the second.

A generic solution which removes these problems is given through the so called authenticated modes or a generic composition of an Encryption Scheme (block cipher in a specific mode of operation or a stream cipher) with a so called message authentication code (explained in Section 5). They provide what is labeled “Authenticated Encryption”. But there are still some possible caveats here, so even if one uses Authenticated Encryption, for instance by using the respective functions of a cryptographic libraries, this can still result in severe vulnerabilities in the application. Specifically, this is highly probable when parts of the plaintext are processed prior to the complete receival of the ciphertext and the integrity verification. The famous padding oracles are one such type of vulnerability, but the are more. This is problem domain is explained in Section 8.

2.3. Use a secure Standard Mode of Operation

The modes of Operation ECB, CBC, CFB, OFB and CTR are defined in [3] and are still considered the standard Block Cipher Modes of Operation. However, there is clear flaw in the ECB mode as pertaining to the leakage of repetitive plaintext patterns through the ciphertext. Thus ECB should not be used. The other modes, except for a subtle problem in CBC, meet all security requirements except that they lack any means of providing integrity protection as addressed in Section 2.2, making it necessary to combine them with a Message Authentication Code, the concept of which will be introduced later in Section 5. Otherwise it is possible to choose an Authenticated Mode like EAX[4], CCM[5] or GCM [6].

2.4. Use the chosen Mode of Operation correctly

Once a mode of operation is chosen, it is important to fully adhere to the requirements are there pertaining to the creation of the initialization vectors and nonces. For instance, be aware that there is a difference in the requirements between a Nonce for CTR and the Initialization Vector for CBC [3]. These are explained in Section 4.3. Do not stray from any of the recommendations given in the respective standard without having fully understood the implications.

3. A Note on Storage Encryption

It is important to understand that storage encryption, e.g. hard disk encryption, is a very different setting than message encryption, the same certainly applies to the aspect of integrity or authenticity. This results in different efficiency and security constraints. For example, in disk encryption the use of Authenticated Encryption is often avoided because the creation of the MAC over a single sector after write access to a single byte incurs a large computational cost. Furthermore, the space occupied by the MAC tags saved on the disk would not be available for data storage. Also, the attack scenarios are different. In the setting of message encryption one generally has to assume that an adversary can intercept and manipulate ciphertexts. Applied to the case of disk encryption, this would mean repeated access to the encrypted storage volume with legitimate use in between. Since there are also attacks where an attacker succeeds by installing malware in the boot sector, which only need a single access to the device, it is justified to leave the more complex attacks uncovered.

For these reasons this document ignores the specifics of storage encryption, it only addresses issues arising in the scenario where messages are exchanged.

4. Properties of Symmetric Encryption Schemes

In this section, the important security and efficiency properties of the standard BCMs are presented. An overview of the standard BCMs with respect to these features in the form of a table is given in Section 4.7.

4.1. On-line Operations

The on-line property of a data processing algorithm means that the algorithm can produce output while the input data is being fed into it part by part. The counterexample is an algorithm that needs the whole input data before it outputs anything. The on-line property is regarded an important efficiency feature not only in symmetric encryption, but in many fields of data processing, for instance in data compression. In symmetric encryption, the on-line property is usually also understood to include the demand that the ciphertext is only larger than the plaintext by a constant (usually a header and/or the MAC).

From a technical point of view, all standard cryptographic modes posses the on-line property for both encryption and decryption. However, from a security perspective, there are important restrictions: Concerning encryption, there is a subtle problem when using CBC encryption in an on-line manner. Details are given in Section 6. The problem with on-line decryption for any standard mode becomes apparent when employing Authenticated Encryption, this will be explained in Section 8.3.

4.2. Random Access

Random read access into the ciphertext is understood as the possibility to decrypt an arbitrary part (for instance a block of the underlying cipher) in the ciphertext without having to decrypt all preceding blocks. In the setting, where messages are exchanged between sender and receiver, this property usually is of no importance. But in some settings it might be relevant.

Random write access, i.e. the selective modification of parts of a ciphertext as parts of the message are changed, plays a role mainly only in storage encryption and will be ignored in this work.

4.3. Initialization Vectors and Nonces

Some BCMs demand an initialization vector (IV), some demand a nonce. Either is needed to randomize the initial state of the encryption scheme, which is necessary to prevent certain attacks. An IV must be unpredictable to any third party which may influence the contents of the plaintext. Specifically, this implies that IVs are not reused, i.e., for any new encryption under the same key, a new IV must be used. Furthermore, an IV must be pseudorandom, i.e. chosen at random and equally distributed from the space of all possible blocks.

The word “nonce” is constructed from the words “number” and “once”. It has much weaker requirements: It just has to be ensured that each nonce is used only once for encryptions under the same key, just as the construction of the word “nonce” suggests. A nonce can be implemented by a counter.

Both IV and nonce creation are addressed in [3]. Note that IV and nonce reusage have different security impacts for the different BCMs.

4.4. Parallelization

For certain applications, it might be relevant whether it is possible to perform encryption or decryption of the individual data blocks in parallel in a BCM. The precondition for this clearly is that the encryption or decryption of any block must be independent of the encryption or decryption results of all the other blocks. Parallelization of block operations allows lower latencies if the respective hardware platform supports some type of parallel computation or the algorithm is implemented in a manner inherently supporting, or demanding, parallel processing of blocks. One technique with this property is referred to as bit slicing [7].

4.5. Precomputation

There are applications, where it can be desirable to have the ability to precompute, before the plaintext or ciphertext is available, intermediate values that allow for faster encryption or decryption after that data is available.

4.6. Malleability

All BCMs exhibit so called malleability properties [8]. This means, that an adversary is able to perform modifications on ciphertexts that, when decrypted, produce plaintexts that are related to the original message. For the different standard BCMs, the malleability properties are different combinations of

- block reordering: the adversary can change the order of blocks in the ciphertext, resulting in a corresponding change of plaintext blocks,
- bit flips: the adversary can introduce bit flips into the ciphertext, which result in predictable bit flips in the plaintext,
- corruption of blocks: as a by-product, the above manipulations can also result in certain plaintext blocks becoming completely pseudorandom and unpredictable.

		CBC	ECB	CFB	OFB	CTR
encryption	on-line	limited, see text	yes	yes	yes	yes
	parall.	no	fully	no	only XOR	fully
	precomp.	no	no	no	yes, after fixing IV	yes, after fixing nonce
decryption	on-line	limited with AE	limited with AE	limited with AE	limited with AE	limited with AE
	parall.	fully	fully	fully	only XOR	fully
	precomp.	no	no	no	yes, after knowing IV	yes, after knowing nonce
	random read access	yes	yes	yes	no	yes
-	IV requirement	pseudo-random, unpredictable	N/A	pseudo-random, unpredictable	nonce	nonce
-	secure with respect to watermarking attack / repeated content leakage	yes	no	yes	yes	yes

Table 1: Overview of the most important properties of the standard BCM. Orange cells indicate a security threat in case the property is ignored.

4.7. Overview of the Standard Modes of Operation

Table 1 shows an overview of the security and efficiency issues of the BCMs discussed in the preceding sections. The last row addressing the repeated content leakage resp. vulnerability against watermarking attacks has not been discussed above. It only affects ECB and is the reason why ECB should not be used – unlike the indicated security threats, this problem of ECB is not a result of erroneous use of the BCM, but an inherent shortcoming of ECB.

Note that the modes CTR and OFB are actually stream ciphers build from a block cipher. The distinguishing feature of stream cipher is that it generates a pseudorandom key stream, which is XORed to the message. CFB is basically also a stream cipher, but there the key stream depends on the previous ciphertext, which is not the case for usual stream ciphers.

5. Message Authentication Codes

A Message Authentication Code (MAC) is an algorithm that produces a cryptographic checksum of a message based on the knowledge of a secret key. Examples are CMAC [9] and HMAC [10, 11].

The usual procedure for combining a MAC with a symmetric encryption scheme, e.g. a block cipher in certain mode of operation, is to append the tag generated by the MAC algorithm to the ciphertext. However there remains the choice as to whether to apply the MAC algorithm to the plaintext or the ciphertext. Considerations of involved security notions are given in [12]. From a formal point of view, the Encrypt-then-MAC composition method is achieving maximal security, also from more practical consideration given in Section 8.1, this method usually is the preferred one. But even with this method there remains a potential pitfall, this is explained in Section 8.2.

6. Chosen Plaintext Attacks

In chosen plaintext attacks the adversary is able to control a part of the plaintext that is being encrypted. All standard modes, except for ECB are secure against this type of attack, provided they are used correctly. However, incorrect use of the CBC mode is even present in established protocols such as SSL as we will see in the following subsection. The second subsection will deal with the general problem of encrypting compressed data.

6.1. CBC's Vulnerability against Blockwise Adaptive Adversaries

An example for incorrect use of a standard mode is the SSL protocol. In a single SSL-session, a single CBC ciphertext is transmitted (which is also protected by a MAC, but this is irrelevant for the problem discussed here). However, attackers that interact with the user or the SSL-client platform during this SSL session, can, in certain scenarios, determine a low entropy string, i.e. a string for which only a small number of values is possible (for instance a PIN number), if they can control the message contents of the first block of the following messages within the same SSL session [13, 14, 15]. An attacker with these properties is a blockwise-adaptive chosen-plaintext adversary (BACPA), a notion introduced in [16, 17]. The reason is that SSL uses the CBC mode erroneously under consideration of BACPA: the protocol treats the whole data exchanged in one session as a single plaintext. However, instead, each packet would have to be treated as a new message and a new unpredictable initialization vector would have to be used. Figure 1 depicts how the corresponding attack against CBC works in general.

This limits the on-line property of CBC to a certain extend: if ciphertext becomes known publicly (which is compliant to the idea of on-line encryption), then there must be no way for a hostile party to influence the further plaintext of the current encryption. The straightforward countermeasure would be to use a new initialization vector whenever an "outside" influence on the plaintext is possible. But then the ciphertext is larger than the plaintext by more than a constant value, thus this aspect of the on-line property is lost. Another countermeasure is to use a stream-cipher (mode), for instance CTR, or CFB[18], which do not suffer from the vulnerability against BACPA. Further secure alternatives involving modified versions of CBC are presented in [18, 19].

6.2. Dangers of Encrypting Compressed Data

The problem resulting from the endangerment of confidentiality of data which was compressed prior to being encrypted is very simple to understand: As stated in Section 2.1, encryption does not cover the problem of hiding the length of the data that is being encrypted. This can be a problem by itself, as corroborated in Section 2.1, but when applying compression before encryption, also messages of equal length may lead to ciphertexts of different length, depending on the compressibility of their content[20][1]. If the attacker knows parts of the plaintext or can influence them, he may be able to deduce whether other parts of the plaintext (partially) match the known part. Putting it very simply: Encryption does not

[1]In the reference, this type of vulnerability is classified as a side-channel vulnerability. We strongly object to this classification, because "side-channel" refers to the channel through which information is gained, i.e. it indicates that an attacker needs information other than what is publicly transmitted. The length of the transmitted data is clearly available in the "main-channel".

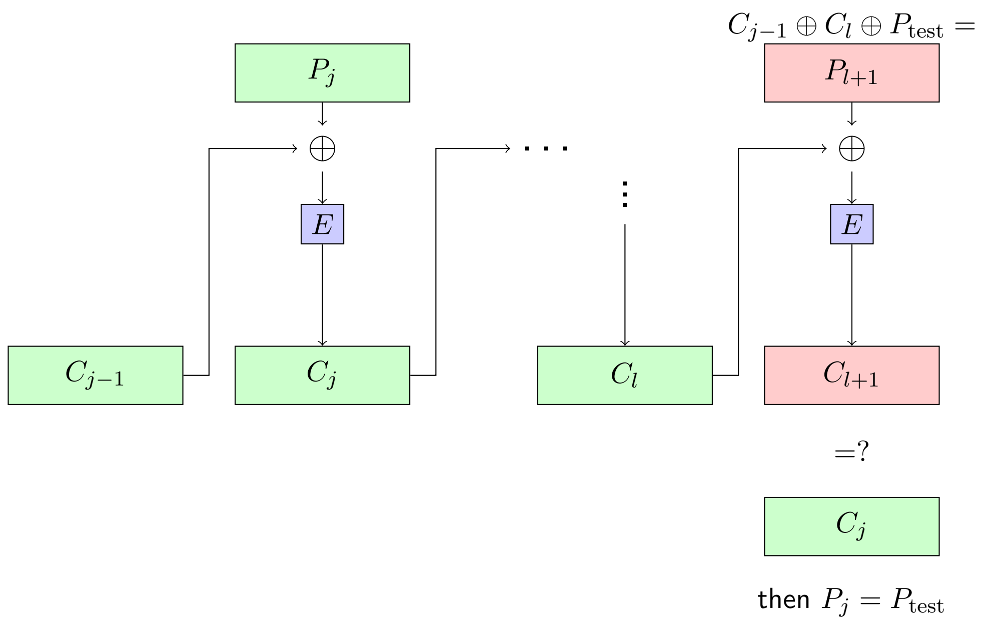
$C_{j-1} \oplus C_l \oplus P_{\text{test}} =$
P_j
P_{l+1}
E
E
C_{j-1}
C_j
C_l
C_{l+1}
$=?$
C_j
then $P_j = P_{\text{test}}$

Figure 1: Depiction of BACPA attack against a CBC encryption as described in [14]. After having seen ciphertexts up to C_l, the attacker makes use of his knowledge that the subsequent plaintext block will be encrypted with C_l as IV. His aim is to decrypt the previous plaintext block P_j encrypted under the same key (in the case of SSL, it has to be from the same SSL session). He must know a great portion of the actual plaintext of P_j, since on average he has to run $n/2$ attempts of the following attack steps, where n is the number of remaining possible values for P_j from his point of view. Furthermore, he must be able to control the contents of the first block of subsequent messages. He iterates through these possible values as P_{test} and provokes the plaintext $P_{l+1} = C_{j-1} \oplus C_l \oplus P_{\text{test}}$ to be encrypted for each such P_{test}. Simple algebra shows that if $C_{l+1} = C_j$, then $P_j = P_{\text{test}}$ and the attacker has determined the missing information from P_j.

hide the length of messages, compression translates content relations to output size, thus taken together, content relations are leaked. A concrete attack that exploits this problem in SSL is CRIME [21].

In general, like the attacks from the preceding section, only low entropy strings can be attacked. However, determining whether this is a risk or not is difficult to assess. Thus, as a consequence, compression should not be applied to data that is encrypted.

7. Known Plaintext Attacks

Basically, any chosen plaintext attack also works as a known plaintext attack, if the variations of the plaintext the attacker would introduce can be achieved by other means. Assume for instance the problem with compression and encryption described in Section 6.2. Assume that an otherwise secure protocol compresses messages sent to large number of users before encrypting them in CTR mode[2]. Let us assume that these messages contain the current date and a secret four digit PIN numbers and no or little other information. Clearly, the current date is known plaintext and on 2012-12-12 the attacker might be able to identify those messages that contain PIN numbers such as 2012, 1212, etc., since a compression algorithm will compress the repeated numbers better than others. If the messages are short enough, with detailed knowledge of the employed compression algorithm, the attacker will end up with a lot of information about certain messages, i.e PIN numbers.

8. (Adaptive) Chosen Ciphertext Attacks

In this section it is discussed why the authenticity of plaintexts is not only necessary to prevent the acceptance of forged messages by the receiver, but also for confidentiality in general. We only discuss the generic construction of Authenticated Encryption by combining encryption in a standard BCM with a MAC, but basically all the discussed aspects apply to the dedicated Authenticated Encryption modes as well.

Adaptive chosen ciphertext attacks are attacks that are, at least in the case of symmetric encryption, aimed at recovering the message to a ciphertext the adversary has gotten hold of. The attacker then makes use of a decryption oracle – in most cases a server, which is the legitimate addressee of the original message. The adversary modifies the original ciphertext and feeds it to the server which decrypts the ciphertext. From explicit error messages and/or timing delays of the server's answers to repeated differently modified versions of the ciphertext, the attacker can get information about the contents of the modified plaintext. In many scenarios, full decryption of the ciphertext is possible. One such scenario is given in the following subsection.

8.1. Avoiding Padding Oracles

When using a block cipher to encrypt a message, in the CBC mode of operation the message has to be padded to a multiple of the block cipher block size before the encryption procedure can be carried out. An exception is the application of ciphertext stealing [22].

To this end, a so called padding scheme has to be applied to the message. There exist a number of such schemes, one prominent example is PKCS#7 padding [23]. In this padding, if the last message block has n unused bytes, these bytes are all set to the value n. If $n = 0$, then a whole message block is appended, each byte having the numeric value of the number of bytes per block for the employed block cipher.

After the decryption, the receiver removes the padding in a straightforward way. However, padding scheme specifications demand an error to be indicated in case of invalid padding. For the above example, this would for instance be the case if in the plaintext the last block's last byte has value n, but at least one

[2]Assuming a stream cipher mode makes the attack easier, as a "real" block cipher mode such as CBC only reveals the number of blocks, not the number of bytes of a plaintext.

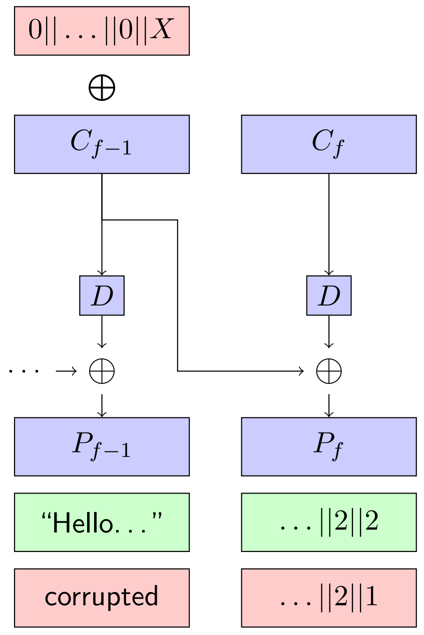

Figure 2: Depiction of a padding oracle attack against CBC with PKCS#7 padding assuming a block with block size of eight bytes, the individual bytes of which are indicated by subscripts; e.g., the first byte of block P is $(P)_0$. Original ciphertext blocks are shown in blue, original plaintext blocks in blue and green. At the top, the modifications introduced by the attacker are shown in red, at the bottom, the resulting modified plaintext is shown in the same colour. The attacker takes an original ciphertext and creates modified versions of it by xoring a block with a varying non-zero value in the last byte X (shown in red) to the next to last block C_{f-1}. The result is a completely corrupted plaintext block P_{f-1}, i.e. it takes on pseudorandom values, and a value of P'_f wich differs from the original block P_f only in the last byte. Let us first assume that the original length of the padding is different from one. In the depicted example, for non-zero values of X, the padding oracle indicates correct padding when $X \oplus (C_{f-1})_7 = 0x01$ as the only correct solution when trying $X = 0x03$ and thus $(C_{f-1})_7 = 0x02$ is the original padding length. To test whether the original padding length is one, prior to the above tests, the attacker can for instance toggle the byte before $(C_{f-1})_7$, if this does not cause padding errors, he knows the original padding length is one.

of the last n bytes does not have value n. Another error condition would be the last byte having a value larger than the number of bytes per block.

In case of unauthenticated encryption, an attacker can build an attack allowing him to decrypt whole ciphertexts if he has access to a padding oracle, i.e. a device that decrypts the ciphertexts and returns an eventual padding error. Such attacks were first described by Vaudenay [24]. Figures 2 and 3 explain how these attacks can be conducted. They potentially lead to the decryption of the whole ciphertext.

Padding oracle attacks building on the evaluation of error messages that reveal whether a padding error occurred, belong to the class of fault attacks. If the attack relies on the timing delay of an unspecific error message (which in itself does not allow the distinction between padding and MAC errors), which will be shorter for a padding error than for MAC failure, then it belongs to the class of timing attacks, which is a type of side channel attack.

It turns out that vulnerabilities of this type are still found in field, despite the fact that they are known for such a long period of time, allowing practical attacks [25].

Authenticated Encryption solves this problem, but certainly only if the authenticity of the message is

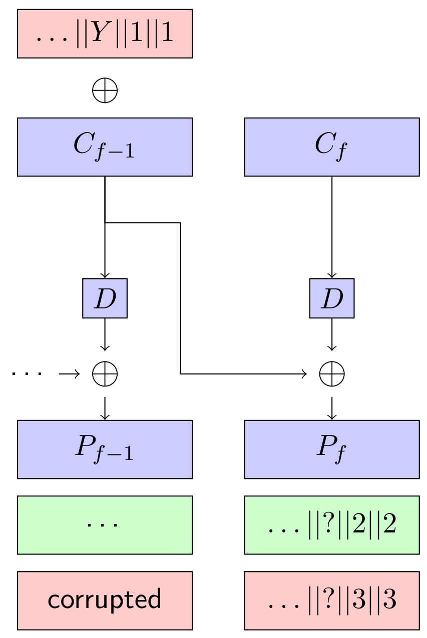

Figure 3: Continuation of the padding oracle attack: After having determined the last byte of the last block, the attacker determines the values of the remaining unknown bytes of the last block (he knows already all the padding bytes). In the example, the original padding length was 2, thus the next unknown byte from the end is $(C_{f-1})_5$. The attacker chooses modification bytes $0x01$ which take the two padding bytes from value $0x02$ to $0x03$. This results in the last byte before the padding to cause padding errors if has a value different from $0x03$. The value Y is varied until no padding error happens when the message is decrypted. In this manner, all the bytes of last block can be determined. Furthermore, due to the block reordering properties of CBC, any other block from the original ciphertext (i.e. encrypted under the same key in the CBC mode) can be placed as the last block; also moving its preceding block with it, i.e. to C_{f-1}, ensures that it is decrypted correctly in the CBC mode. This method even works when the attacker cannot control the expected length of the message (in which case he could simply cut off the ciphertext at the desired block), for instance because it is fixed from the point of view of the receiving side. In this way, the complete ciphertext can be decrypted.

verified prior to the removal of the padding, this is addressed in the following subsection.

8.2. The Order of Decryption and MAC Verification

In Section 5 it was stated that the Encrypt-then-MAC variant is the preferred choice over MAC-then-Encrypt when it comes to build an Authenticated Encryption scheme from a BCM and a MAC algorithm. But note that it is possible to have implementations of either method that are secure and insecure with respect to padding oracle attacks (introduced in Section 8.1):

Given the Encrypt-then-MAC construction, the secure decryption operation is given by first verifying the MAC, then performing decryption and finally performing the unpadding. However, checking the padding and indicating an error is possible after decryption, which is always independent of having verified the MAC, thus it is also possible to create vulnerable implementations of the decryption in the Encrypt-then-MAC construction.

Concerning the MAC-then-Encrypt construction, it is much more difficult to create an implementation which does not leak plaintext information via timing. Clearly, if the MAC is over the plaintext, the padding has to be removed before the MAC can be verified. Thus, in this case, it is highly important to perform the unpadding operation in a constant time manner, i.e. having a timing indepent of the actual number of padding bytes and the validity of the padding. By suppressing a possible padding error and carrying out the MAC verification unconditionally, again, in a manner that is independent of the actual number of padding bytes, (which influence the number of bytes fed into the MAC computation and thus per se the timing of the MAC computation), it is possible to reach timing security. As a matter of fact, this complicated procedure is necessary to reach secure implementations of TLS in certain configurations [26].

8.3. Decryption in Authenticated Encryption is not on-Line

It is important to understand that decryption in Authenticated Encryption may be on-line only in a very limited way: the decrypted plaintext depends on the authentication block which can only be verified when the whole ciphertext has been received. This is because of the way Authenticated Encryption is defined: if the authentication fails, no plaintext may be output. Violations of this rule potentially enable application oracle attacks, where application processing and corresponding error messages (or timing effects) function analogously to the processing of the padding in padding oracle attacks. An example for this is [1].

It is important to understand the difference concerning this issue in view of the two different security goals provided by the authentication: ensure authenticity in general and confidentiality in the presence of decryption oracles.

For the first goal of authenticity it is tolerable to process parts of the plaintext before its authenticity has been verified. This can be delayed until just before letting the plaintext “take effect”, the meaning of which certainly depends on the application context. It could for instance mean a database transaction or the booking of a money order. In this sense we will refer to this security goal as “effective authenticity”.

But for the second goal, confidentiality with respect to decryption oracles, it would only be tolerable to process parts of the yet unauthenticated plaintext if it could be ensured that this very processing does not provide such an oracle. This, however, is in general a futile task because it means carrying such notions as security against side channel and fault attacks from the cryptographic world into the world of general applications. This would mean a side channel and fault attack secure implementation of the application’s decoding and processing functionality, which we can consider as basically impossible in view of the multitude of unavoidable conditional branching typically involved in these operations. Only in very special cases of very simple operations on the plaintext this could be considered an option. Consequently, we label this security goal as “operational authenticity”. Operational authenticity is given when only properly authenticated data is being processed.

Figure 4 illustrates how correctly implemented Authenticated Encryption prevents decryption oracle attacks and Figure 5 depicts the connection between effective and operational authenticity using the

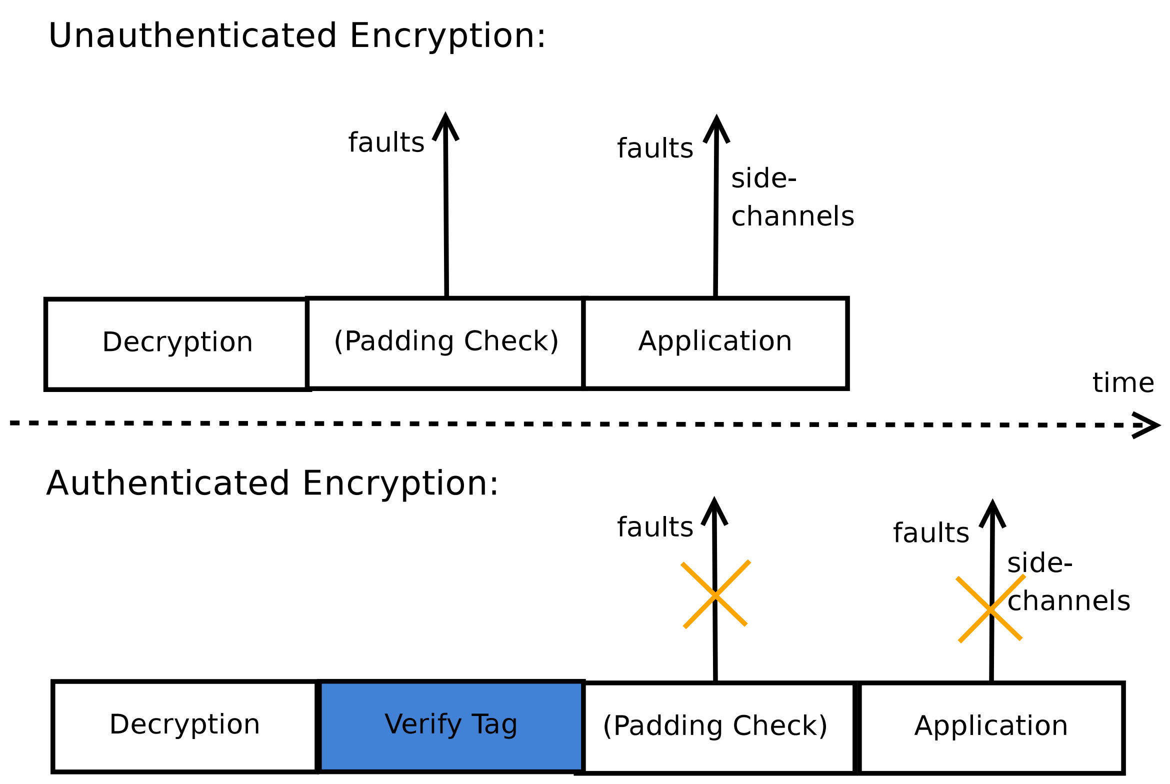

Figure 4: Depiction of order of steps in decryption in Authenticated Encryption to ensure confidentiality.

example of typical APIs of cryptographic libraries, the caveats of which are discussed in Appendix A.

Especially in the context of memory-constrained devices, one must assume a great temptation to wrongly implement Authenticated Encryption, since processing the plaintext as it becomes available is the only way to deal with messages that exceed the device's memory. In such cases it is strongly recommended to break down of the message into chunks that can be processed in the whole and are authenticated individually.

9. Hybrid Encryption

Hybrid Encryption is the term for the public key encryption of a symmetric key which in turn is used to encrypt the actual message. This is the preferred approach to the encryption of messages that exceed the capacity of public key plaintexts. It goes without saying that in this case the use of Authenticated Encryption is also mandatory. Using a public key scheme secure against adaptive chosen ciphertext attacks such as RSA-OAEP [27] together with unauthenticated encryption would be one of the most ironic constructions in cryptography that can be thought of: the adaptive chosen ciphertext vulnerability of padding or application processing of the plaintext would be left unaddressed while for symmetric key encryption with public key cryptosystems, much simpler schemes achieve security against these kinds of attack, for instance "Simple RSA" [28].

10. Conclusion

This work addresses the most prevalent security flaws that can be found in message encryption in applications and frameworks. As stated in the introduction, it is hard to get this understanding from current text books, as they do not treat the matter in a concise and focussed manner and usually do not make the clear

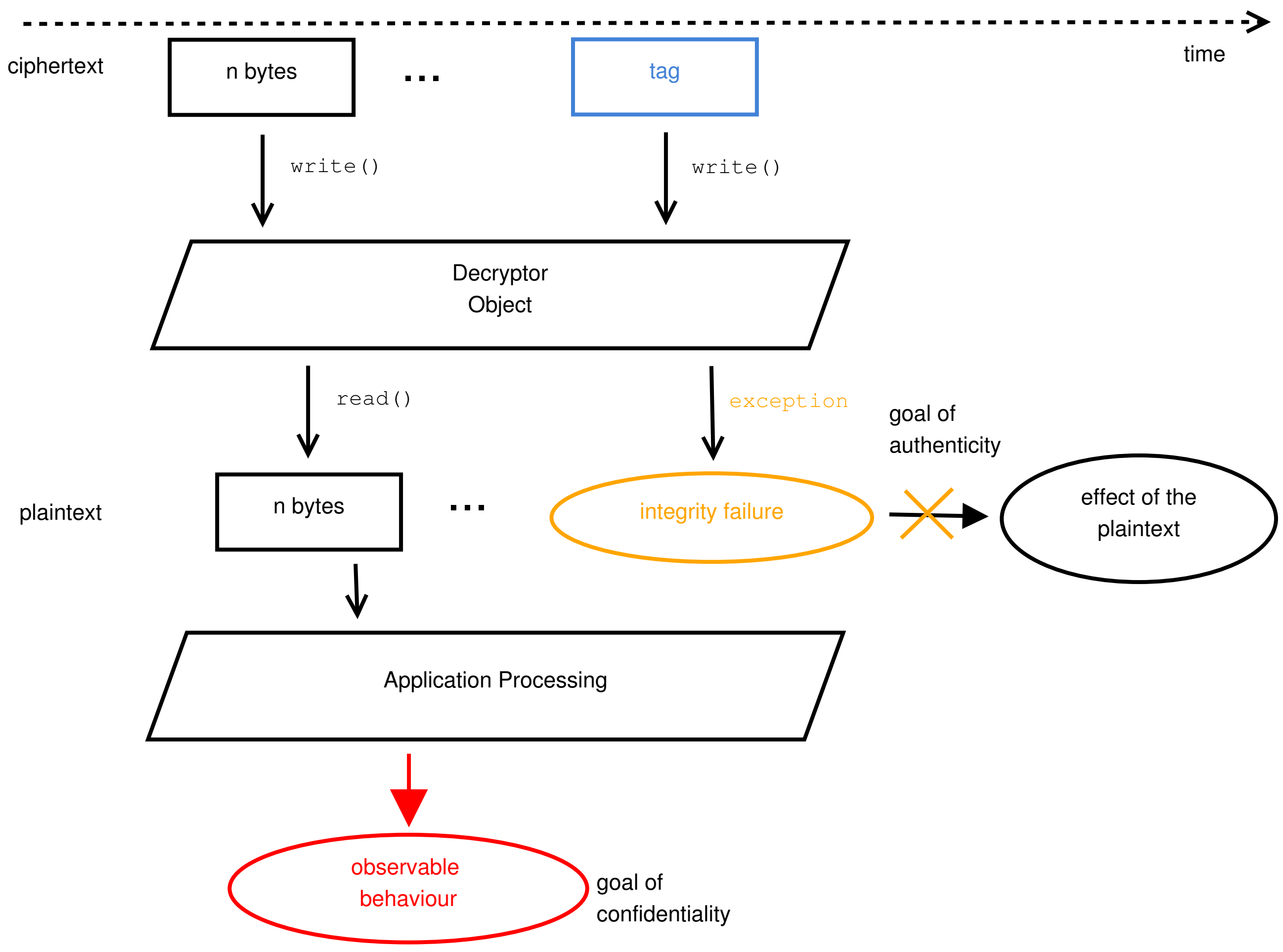

Figure 5: Depiction of general decryption oracles during on-line decryption of Authenticated Encryption ciphertexts using the example of the typical API of cryptographic libraries. The problem arises if parts of the plaintext are processed in an on-line manner by the next layer above the cryptographic one, which we unambiguously label as the application layer, before the authenticity of the ciphertext has been verified. Then, application errors arising during the processing of adaptively manipulated ciphertext may be revealed to an adversary, either through concrete error messages or through timing effects, allowing him to get information about the message content by and by, often leading to full decryption of messages. Note that in this example, contrasting to the goal of confidentiality, the goal of authenticity is reached, because the plaintext is prevented from taking effect.

distinction between internals of primitives, which are completely irrelevant to most software designers, and their correct application in protocols.

Summarizing the most important facts this work reviews we can give the following instructions for building secure protocols for encrypted and necessarily also authenticated messages:

1. Use Authenticated Encryption (either by combining a MAC with a BCM or using a dedicated Authenticated Encryption mode) (Section 2.2)
2. Adhere to requirements for the nonce and IV generation given in the respective standard (Section 2.4).
3. If padding of the message is part of the scheme (for instance when using CBC), make sure that the authenticity of the message is verified prior to the processing of the padding during decryption (Section 8.1).
4. Do not process any part of the decrypted message before the authenticity has been verified (Section 8.3).
5. Make sure that the length of the ciphertext does not provide any significant information to an adversary (Section 2.1). To generically complicate attacks that exploit the length leakage of ciphertexts, a generic countermeasure can be realized by adding a string of a few bytes of random length to the plaintext at the protocol layer.
6. Do not compress messages prior to encryption. This requirement should only be dropped if thorough analysis shows that attacks against low entropy strings are not possible in the respective application (Section 6.2). The generic countermeasure proposed in the previous item can also be considered, but then it is important to use (pseudo) random fill bytes, as for instance a string of zero bytes might be compressed very efficiently and the compressed length might be independent of its original length.
7. If CBC is used, one has to be aware of the restricted on-line property of CBC encryption in the presence of blockwise adaptive adversaries (Section 6.1).

References

[1] Jager, T., Somorovsky, J.: How to break XML encryption. In: Proceedings of the 18th ACM conference on Computer and communications security. CCS '11, New York, NY, USA, ACM (2011) 413–422

[2] Ferguson, N., Schneier, B., Kohno, T.: Cryptography Engineering. John Wiley & Sons, Inc. (2010)

[3] Morris Dworkin: Special Publication 800-38A 2001 Edition – Recommendation for Block Cipher Modes of Operation (2001)

[4] Bellare, M., Rogaway, P., Wagner, D.: The EAX Mode of Operation. In: Fast Software Encryption (FSE) 2004. (2004)

[5] Morris Dworkin: NIST Special Publication 800-38C – Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality (2004)

[6] Morris Dworkin: NIST Special Publication 800-38D – Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC (2007)

[7] Rebeiro, C., Selvakumar, D., Devi, A.: Bitslice implementation of aes. In Pointcheval, D., Mu, Y., Chen, K., eds.: Cryptology and Network Security. Volume 4301 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2006) 203–212

[8] Dolev, D., Dwork, C., Naor, M.: Non-malleable Cryptography. SIAM Journal on Computing 3 **2** (2000) 391–497

[9] Morris Dworkin: NIST Special Publication 800-38B – Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication (2005)

[10] Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: CRYPTO '96 Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1996) 1–15

[11] Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed hashing for message authentication. In: IETF RFC-2104. (1997)

[12] Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology. ASIACRYPT '00, London, UK, Springer-Verlag (2000) 531–545

[13] Bodo Möller: OpenSSL email archive (2002) `http://www.mail-archive.com/openssl-dev@openssl.org/msg10664.html`.

[14] Bard, G.V.: A challenging but feasible blockwise-adaptive chosen-plaintext attack on ssl. In: SECRYPT 2006, PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON SECURITY AND CRYPTOGRAPHY, SET'UBAL, INSTICC Press (2006) 7–10

[15] `http://www.ekoparty.org/2011/juliano-rizzo.php`

[16] Joux, A., Martinet, G., Valette, F.: Blockwise-adaptive attackers - revisiting the (in)security of some provably secure encryption modes: Cbc, gem, iacbc. In: In Proceedings of Advances in Cryptology - Crypto 2002, LNCS 2442, Springer-Verlag (2002) 2002

[17] Bellare, M., Kohno, T., Namprempre, C.: Provably Fixing the SSH Binary Packet Protocol (2002)

[18] Pierre-alain Fouque, Gwenaëlle Martinet and Guillaume Poupard: Practical symmetric on-line encryption. In: In Lecture Notes in Computer Science. Advances in Cryptology – FSE'03, Springer-Verlag (2003)

[19] Rogaway, P., Wooding, M., Zhang, H.: The Security of Ciphertext Stealing. In Canteaut, A., ed.: Fast Software Encryption. Volume 7549 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 180–195 slides available at `fse2012.inria.fr/SLIDES/48.pdf`.

[20] Kelsey, J.: Compression and information leakage of plaintext. In: Revised Papers from the 9th International Workshop on Fast Software Encryption. FSE '02, London, UK, UK, Springer-Verlag (2002) 263–276

[21] `http://www.ekoparty.org/eng/2012/thai-duong.php`

[22] Morris Dworkin: NIST Special Publication 800-38A – Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode (2010)

[23] `http://tools.ietf.org/html/rfc5652#section-6.3`

[24] Vaudenay, S.: Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS. In: Advances in Cryptology – EUROCRYPT 2002, Springer-Verlag (2002) 543–545

[25] Duong, T., Rizzo, J.: Cryptography in the Web: The Case of Cryptographic Design Flaws in ASP.NET. In: Security and Privacy (SP), 2011 IEEE Symposium on. (May 2011) 481 –489

[26] AlFardan, N.J., Paterson, K.G.: Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. (2013)

[27] RSA Laboratories, RSA Security Inc., 20 Crosby Drive, Bedford, MA 01730 USA: RSAES-OAEP Encryption Scheme (2000)

[28] David Pointcheval: How to encrypt properly with RSA RSA Laboratories' Crypto-Bytes, 5(1) (Winter/Spring 2002).

[29] The Botan Library: `botan.randombit.net`.

[30] The Crypto++ Library: `www.cryptopp.com`.

[31] BouncyCastle JCE Provider: `http://www.bouncycastle.org/java.html`.

A. Problematic Implementations of Authenticated Encryption in open Source cryptographic Libraries

The Botan Library [29] offers a convenient mechanism to handle the encryption and decryption of data streams. This is the so called Pipe API. If one writes code that for instance decrypts a stream of data in the previously mentioned EAX mode, one creates a corresponding Pipe object in the C++ source code. Then, one can call a `write()` function on this Pipe, and feed a part of the ciphertext. Afterwards, it is possible to read as much of the decrypted plaintext as is available so far, the length of which depends on the amount of ciphertext fed to the decryption pipe in an obvious way. This is done by calling a `read()` function on this object.

The problem is that a manipulation of the ciphertext is not recognized until the last part of the ciphertext is fed into the decryption pipe, as the integrity in EAX is provided by a MAC appended at the end of the ciphertext. As a consequence, only when the whole ciphertext has been processed, an exception is thrown by the `write()` function.

Thus in the Botan API, until the last ciphertext part is fed to the decryption pipe, calls to the `read()` function actually return manipulated plaintext if the ciphertext was manipulated. We verified this in an example implementation using version 1.7.2 of Botan. This of course implies the risk that the data is not only decrypted, but the resulting plaintext is also further processed by the client application, which in turn creates the threat of decryption oracles as presented in the previous sections. The EAX mode's encryption uses the CTR mode internally, which offers a plain XOR-homomorphicity without any corrupting properties, thus in general allowing for oracle attacks. This is depicted in Figure 5.

While function names and call sequences are different in the Crypto++ Library [30] (also written in C++), the problem is the same. We performed a test analogous to the one for the Botan Library using the authenticated GCM mode. Our tests are based on version 5.6.1.

The same behavior can be observed in the Java Cryptographic Architecture (JCA), where it is also possible to work on streams in basically the same manner as in the C++ libraries discussed. In the JCA, the concrete implementations of cryptographic algorithms are provided by exchangeable modules, the so called Cryptographic Providers. In our concrete test, we used the BouncyCastle Provider [31], version 1.45. It is probably not appropriate to blame the JCA itself for this, since it well allows to implement Authenticated Encryption correctly. But it would be favorable if the JCA enforced the correct treatment of Authenticated Encryption through the interface specification. We are not sure, however, if the concept of the JCA would enable such a solution.

It is certainly arguable whether it is desirable to have a cryptographic library that hold backs the decrypted plaintext until the MAC verification. In some scenarios, it might for instance be necessary to write the decrypted plaintext onto the hard disk due to limitations of the available RAM. The main problem is a missing distinction between interfaces that allow potentially dangerous access and should only be used by experts, like the examples above, and interfaces that prevent erroneous use of this kind and consequently certainly restrict the application programmer's freedom to some extend.