Decryption Oracle Attacks against unauthenticated Encryption based on Tag-Length-Value Decoding

Falko Strenzke

FlexSecure GmbH, Germany,
strenzke@flexsecure.de

March 9, 2011

Introduction

- Topic of this talk are decryption oracle attacks against unauthenticated encryption
- a lot of previous work on padding oracle attacks
- this work deal with *application* oracle attacks
- specifically: "Destinguished Encoding Rules" (DER), a type of Tag-Length-Value (TLV) encoding
- in order to prevent the drawing of the conclusion that without padding decryption oracle attacks are not a threat to symmetric encryption

Introduction

- Topic of this talk are decryption oracle attacks against unauthenticated encryption
- a lot of previous work on padding oracle attacks
- this work deal with *application* oracle attacks
- specifically: "Destinguished Encoding Rules" (DER), a type of Tag-Length-Value (TLV) encoding
- in order to prevent the drawing of the conclusion that without padding decryption oracle attacks are not a threat to symmetric encryption

Introduction

- Topic of this talk are decryption oracle attacks against unauthenticated encryption
- a lot of previous work on padding oracle attacks
- this work deal with *application* oracle attacks
- specifically: "Destinguished Encoding Rules" (DER), a type of Tag-Length-Value (TLV) encoding
- in order to prevent the drawing of the conclusion that without padding decryption oracle attacks are not a threat to symmetric encryption

Introduction

- Topic of this talk are decryption oracle attacks against unauthenticated encryption
- a lot of previous work on padding oracle attacks
- this work deal with *application* oracle attacks
- specifically: "Destinguished Encoding Rules" (DER), a type of Tag-Length-Value (TLV) encoding
- in order to prevent the drawing of the conclusion that without padding decryption oracle attacks are not a threat to symmetric encryption

Introduction

- Topic of this talk are decryption oracle attacks against unauthenticated encryption
- a lot of previous work on padding oracle attacks
- this work deal with *application* oracle attacks
- specifically: "Destinguished Encoding Rules" (DER), a type of Tag-Length-Value (TLV) encoding
- in order to prevent the drawing of the conclusion that without padding decryption oracle attacks are not a threat to symmetric encryption

Decryption Oracle Attacks

- Decryption Oracle Attacks work as follows:
  - The attacker wishes to decrypt a certain ciphertext
  - he has access to a decryption oracle (usually a device) that decrypts the ciphertext and reveals *some* property of the plaintext
  - he lets the device decrypt (adaptively) manipulated versions of the respective ciphertext
  - and gains information by observing the decryption
- the information about plaintext properties through
  - error messages
  - side channels

Decryption Oracle Attacks

- Decryption Oracle Attacks work as follows:
  - The attacker wishes to decrypt a certain ciphertext
  - he has access to a decryption oracle (usually a device) that decrypts the ciphertext and reveals *some* property of the plaintext
  - he lets the device decrypt (adaptively) manipulated versions of the respective ciphertext
  - and gains information by observing the decryption
- the information about plaintext properties through
  - error messages
  - side channels

Decryption Oracle Attacks

- Decryption Oracle Attacks work as follows:
  - The attacker wishes to decrypt a certain ciphertext
  - he has access to a decryption oracle (usually a device) that decrypts the ciphertext and reveals *some* property of the plaintext
  - he lets the device decrypt (adaptively) manipulated versions of the respective ciphertext
  - and gains information by observing the decryption
- the information about plaintext properties through
  - error messages
  - side channels

Decryption Oracle Attacks

- Decryption Oracle Attacks work as follows:
  - The attacker wishes to decrypt a certain ciphertext
  - he has access to a decryption oracle (usually a device) that decrypts the ciphertext and reveals *some* property of the plaintext
  - he lets the device decrypt (adaptively) manipulated versions of the respective ciphertext
  - and gains information by observing the decryption
- the information about plaintext properties through
  - error messages
  - side channels

Decryption Oracle Attacks

- Decryption Oracle Attacks work as follows:
  - The attacker wishes to decrypt a certain ciphertext
  - he has access to a decryption oracle (usually a device) that decrypts the ciphertext and reveals *some* property of the plaintext
  - he lets the device decrypt (adaptively) manipulated versions of the respective ciphertext
  - and gains information by observing the decryption
- the information about plaintext properties through
  - error messages
  - side channels

Decryption Oracle Attacks

- Decryption Oracle Attacks work as follows:
  - The attacker wishes to decrypt a certain ciphertext
  - he has access to a decryption oracle (usually a device) that decrypts the ciphertext and reveals *some* property of the plaintext
  - he lets the device decrypt (adaptively) manipulated versions of the respective ciphertext
  - and gains information by observing the decryption
- the information about plaintext properties through
  - error messages
  - side channels

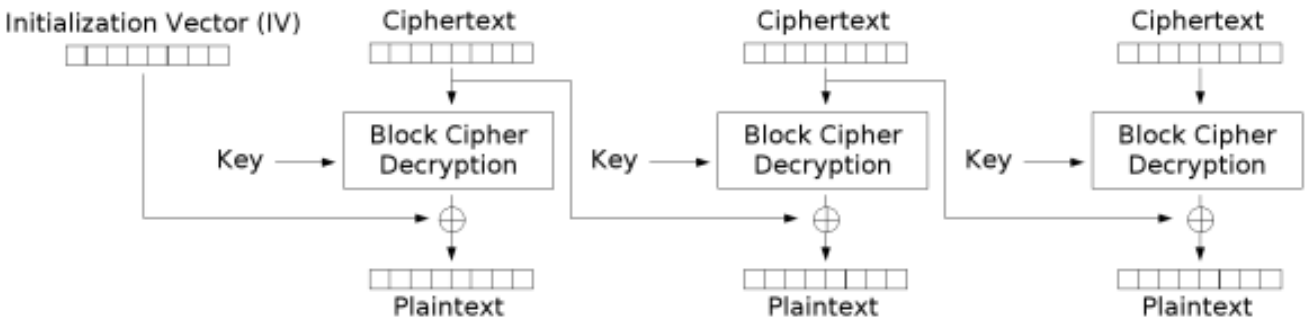
Decryption Oracle Attacks

- Decryption Oracle Attacks work as follows:
  - The attacker wishes to decrypt a certain ciphertext
  - he has access to a decryption oracle (usually a device) that decrypts the ciphertext and reveals *some* property of the plaintext
  - he lets the device decrypt (adaptively) manipulated versions of the respective ciphertext
  - and gains information by observing the decryption
- the information about plaintext properties through
  - error messages
  - side channels

Decryption Oracle Attacks

- Decryption Oracle Attacks work as follows:
  - The attacker wishes to decrypt a certain ciphertext
  - he has access to a decryption oracle (usually a device) that decrypts the ciphertext and reveals *some* property of the plaintext
  - he lets the device decrypt (adaptively) manipulated versions of the respective ciphertext
  - and gains information by observing the decryption
- the information about plaintext properties through
  - error messages
  - side channels

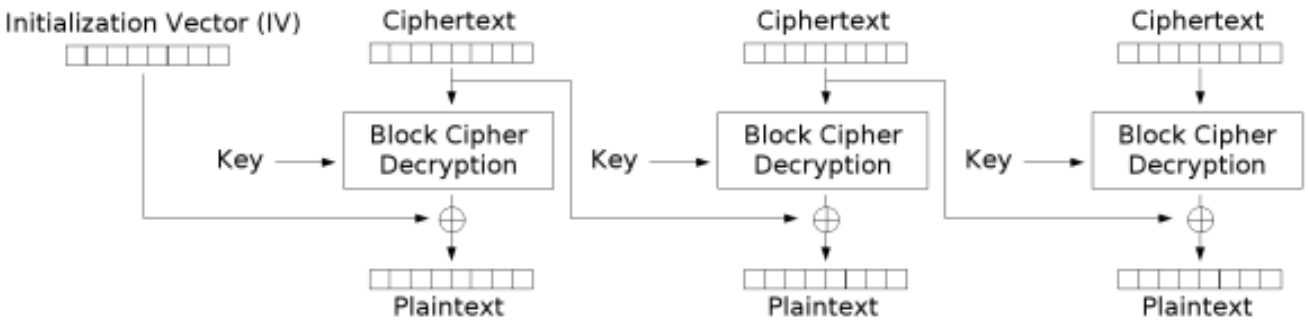
The Cipher Block Chaining (CBC) Mode



Cipher Block Chaining (CBC) mode decryption

- CBC features an XOR-homomorphicity with corruption:
  - flipping a bit in ciphertext block C_i causes a flip of the corresponding bit position in the plaintext block P_{i+1}
  - and due to the pseudorandom properties of the block cipher it corrupts the plaintext block P_i entirely
- blocks of ciphertext that are CBC encrypted under the same key can be reordered
  - because the decryption of one block only depends on two subsequent ciphertext blocks
  - however, reordering also introduces corrupted blocks

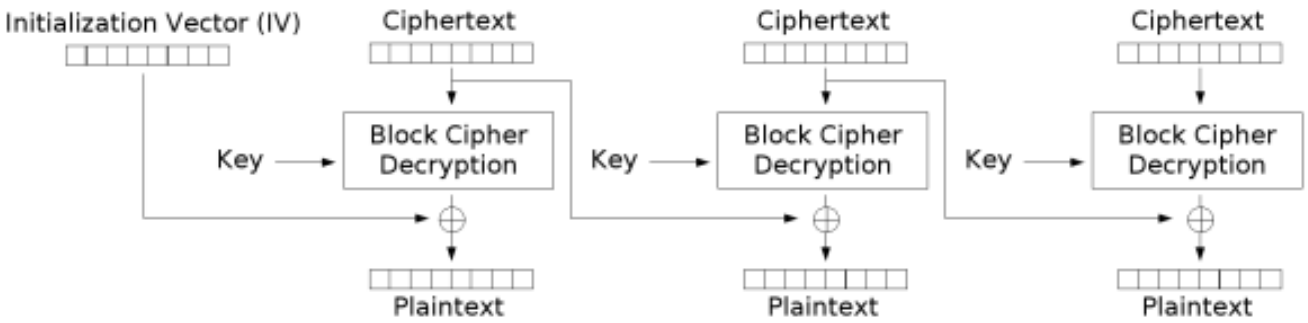
The Cipher Block Chaining (CBC) Mode



Cipher Block Chaining (CBC) mode decryption

- CBC features an XOR-homomorphicity with corruption:
  - flipping a bit in ciphertext block C_i causes a flip of the corresponding bit position in the plaintext block P_{i+1}
  - and due to the pseudorandom properties of the block cipher it corrupts the plaintext block P_i entirely
- blocks of ciphertext that are CBC encrypted under the same key can be reordered
  - because the decryption of one block only depends on two subsequent ciphertext blocks
  - however, reordering also introduces corrupted blocks

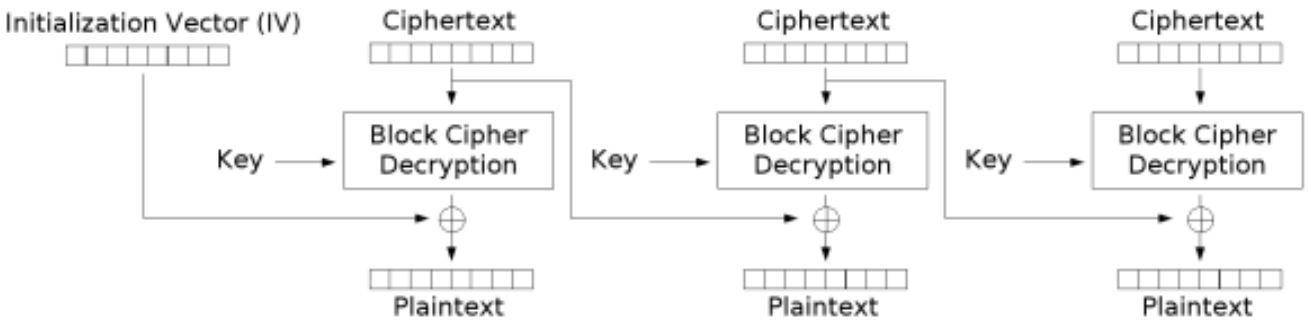
The Cipher Block Chaining (CBC) Mode



Cipher Block Chaining (CBC) mode decryption

- CBC features an XOR-homomorphicity with corruption:
  - flipping a bit in ciphertext block C_i causes a flip of the corresponding bit position in the plaintext block P_{i+1}
  - and due to the pseudorandom properties of the block cipher it corrupts the plaintext block P_i entirely
- blocks of ciphertext that are CBC encrypted under the same key can be reordered
  - because the decryption of one block only depends on two subsequent ciphertext blocks
  - however, reordering also introduces corrupted blocks

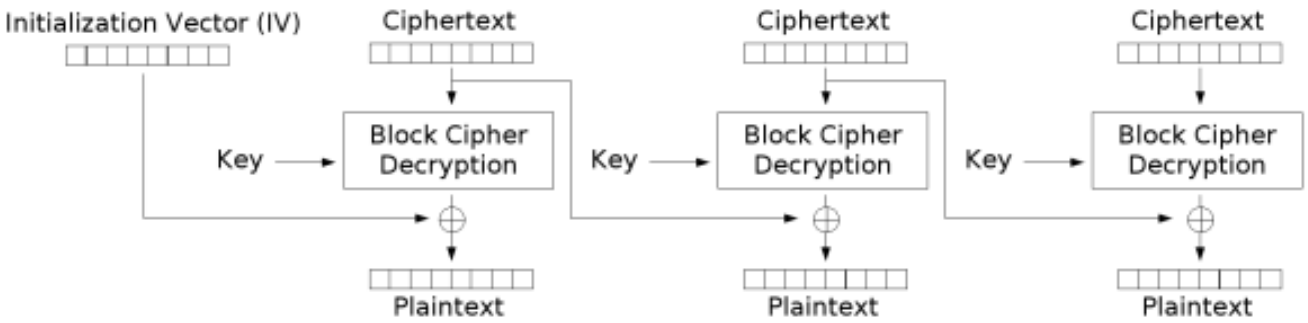
The Cipher Block Chaining (CBC) Mode



Cipher Block Chaining (CBC) mode decryption

- CBC features an XOR-homomorphicity with corruption:
  - flipping a bit in ciphertext block C_i causes a flip of the corresponding bit position in the plaintext block P_{i+1}
  - and due to the pseudorandom properties of the block cipher it corrupts the plaintext block P_i entirely
- blocks of ciphertext that are CBC encrypted under the same key can be reordered
  - because the decryption of one block only depends on two subsequent ciphertext blocks
  - however, reordering also introduces corrupted blocks

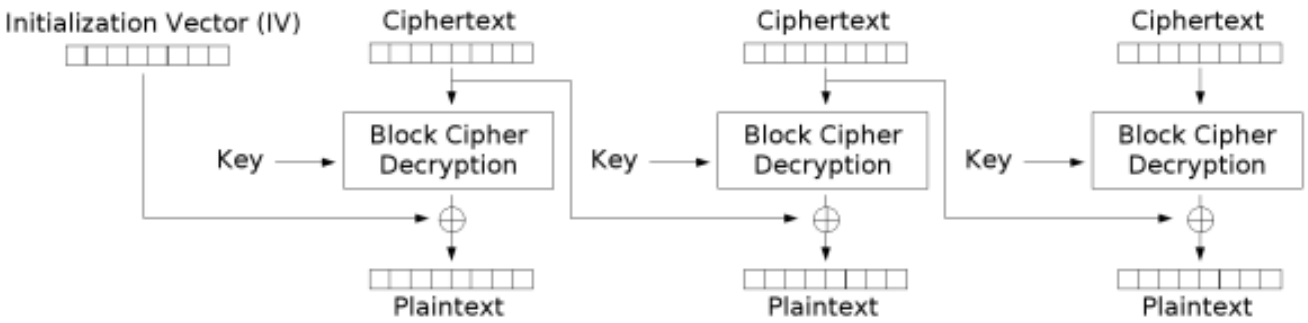
The Cipher Block Chaining (CBC) Mode



Cipher Block Chaining (CBC) mode decryption

- CBC features an XOR-homomorphicity with corruption:
  - flipping a bit in ciphertext block C_i causes a flip of the corresponding bit position in the plaintext block P_{i+1}
  - and due to the pseudorandom properties of the block cipher it corrupts the plaintext block P_i entirely
- blocks of ciphertext that are CBC encrypted under the same key can be reordered
  - because the decryption of one block only depends on two subsequent ciphertext blocks
  - however, reordering also introduces corrupted blocks

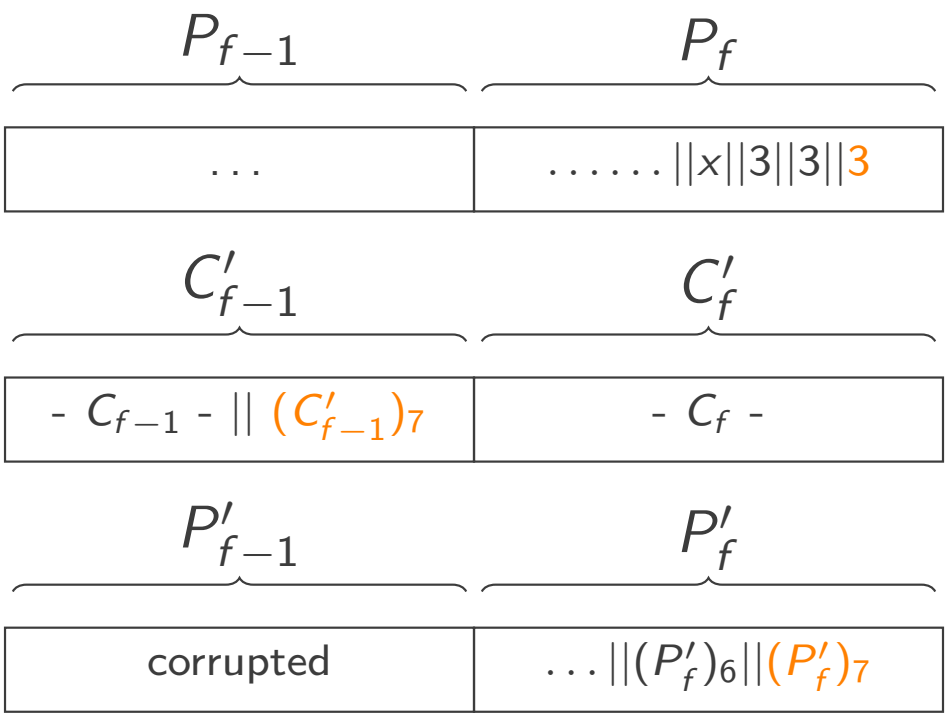
The Cipher Block Chaining (CBC) Mode



Cipher Block Chaining (CBC) mode decryption

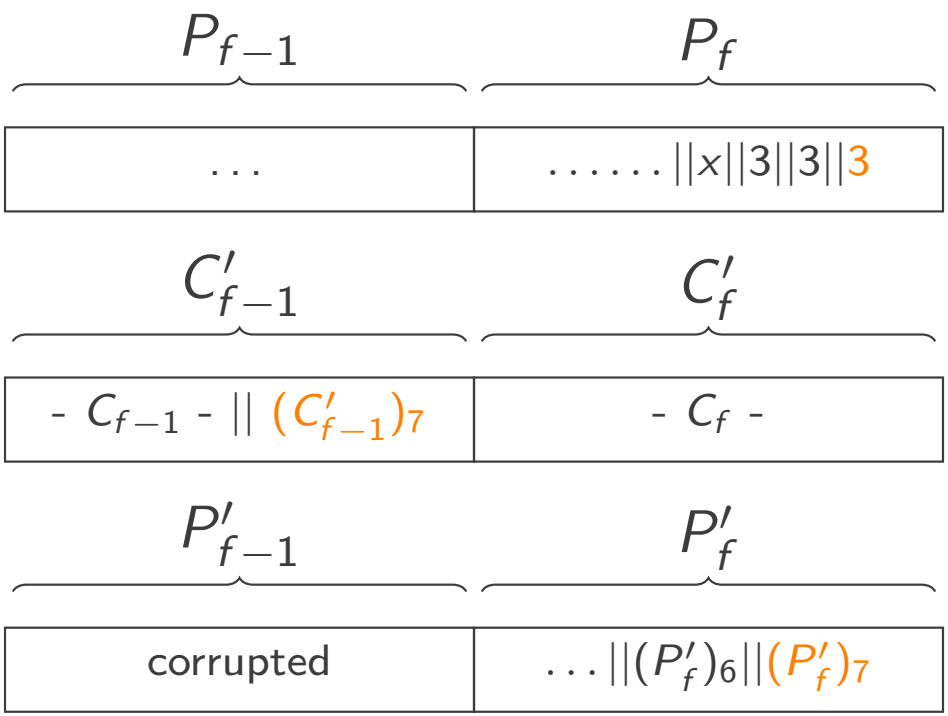
- CBC features an XOR-homomorphicity with corruption:
  - flipping a bit in ciphertext block C_i causes a flip of the corresponding bit position in the plaintext block P_{i+1}
  - and due to the pseudorandom properties of the block cipher it corrupts the plaintext block P_i entirely
- blocks of ciphertext that are CBC encrypted under the same key can be reordered
  - because the decryption of one block only depends on two subsequent ciphertext blocks
  - however, reordering also introduces corrupted blocks

PKCS#7 Padding Decryption Oracle



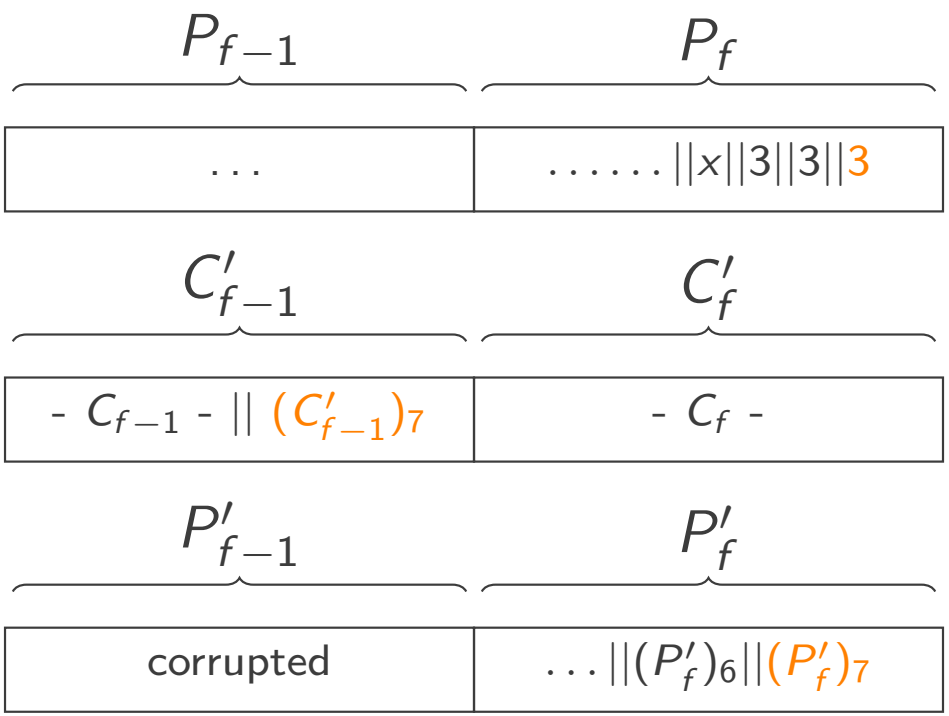
- $(C'_{f-1})_7 = (C_{f-1})_7 \oplus A$
- seek for value of A such that no padding exception can be triggered by any value of $(C'_{f-1})_6$:
- then $(P'_f)_7 = 0x01 \rightarrow$ the padding length (3) is recovered

PKCS#7 Padding Decryption Oracle



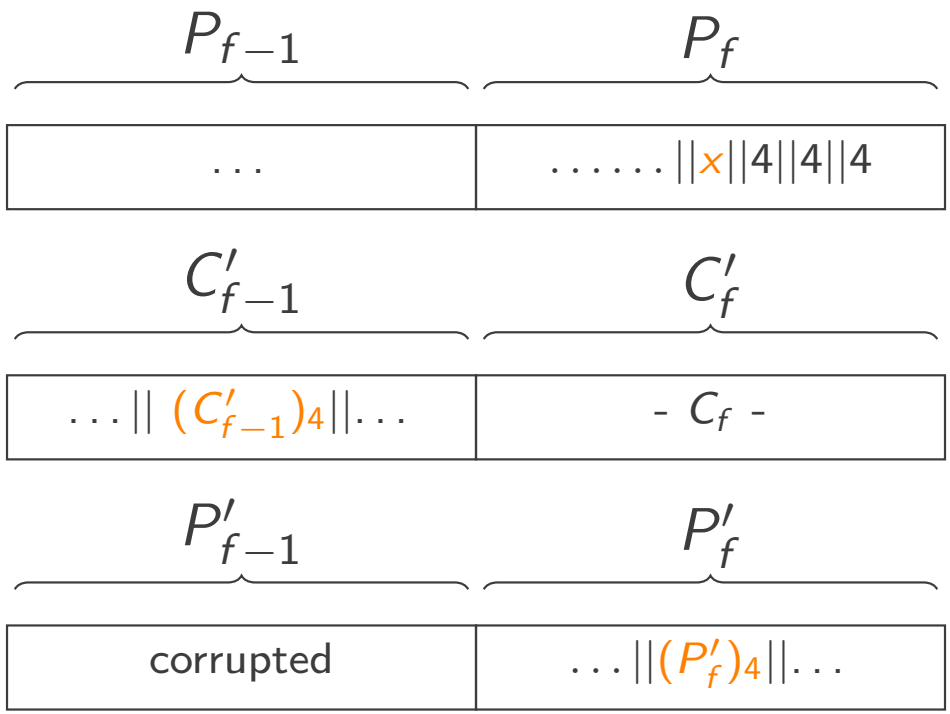
- $(C'_{f-1})_7 = (C_{f-1})_7 \oplus A$
- seek for value of A such that no padding exception can be triggered by any value of $(C'_{f-1})_6$:
- then $(P'_f)_7 = 0x01 \rightarrow$ the padding length (3) is recovered

PKCS#7 Padding Decryption Oracle



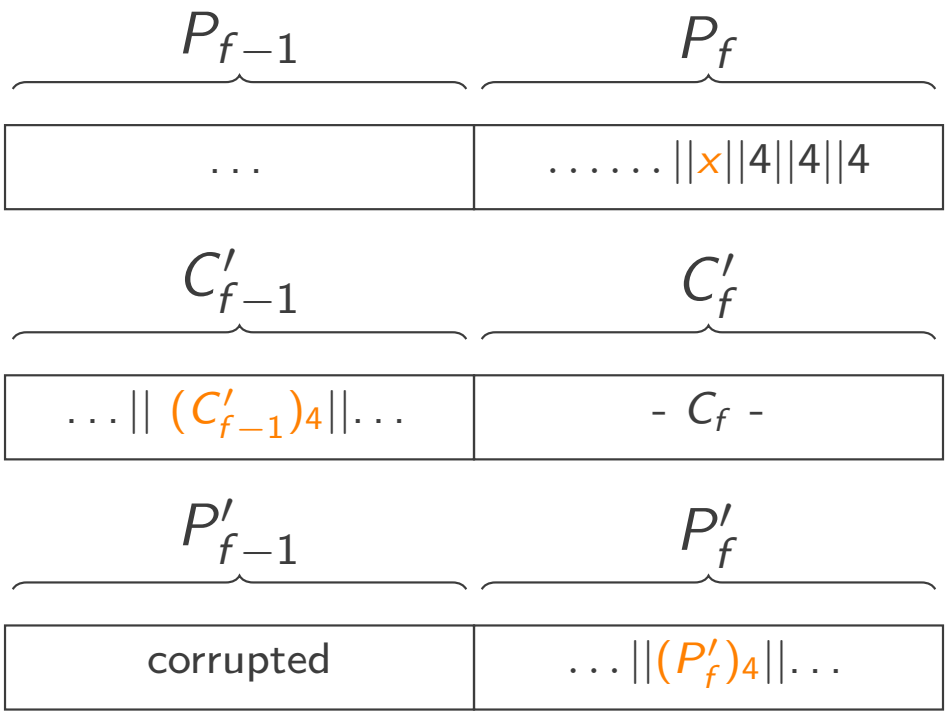
- $(C'_{f-1})_7 = (C_{f-1})_7 \oplus A$
- seek for value of A such that no padding exception can be triggered by any value of $(C'_{f-1})_6$:
- then $(P'_f)_7 = 0x01 \rightarrow$ the padding length (3) is recovered

PKCS#7 Padding Decryption Oracle



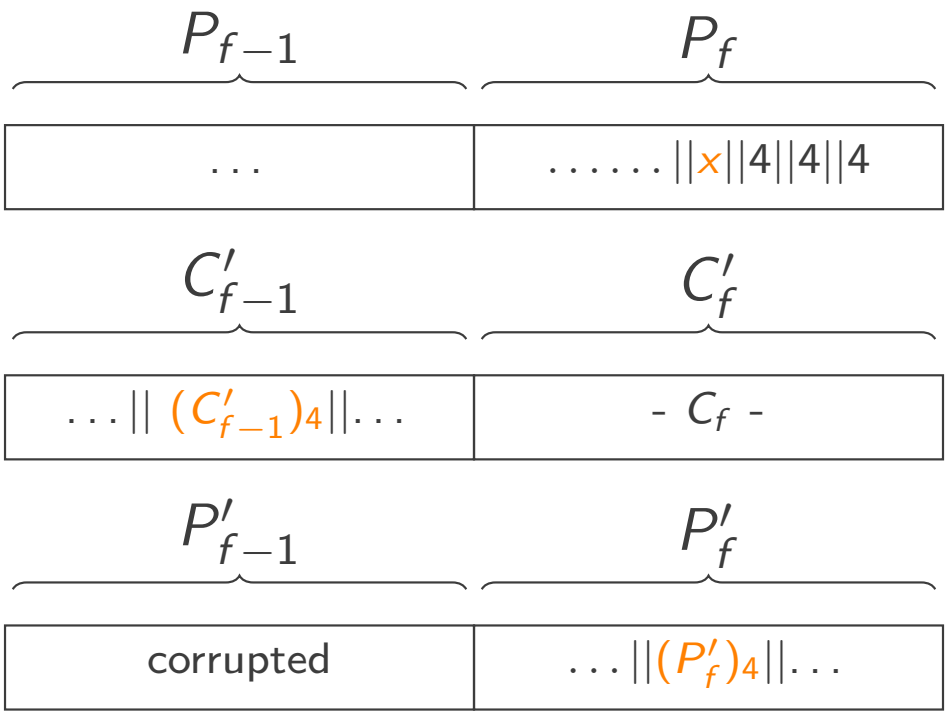
- now the message bytes are attacked one by one
- any block can be placed as the final block
- (due to the block reordering properties of CBC)

PKCS#7 Padding Decryption Oracle



- now the message bytes are attacked one by one
- any block can be placed as the final block
- (due to the block reordering properties of CBC)

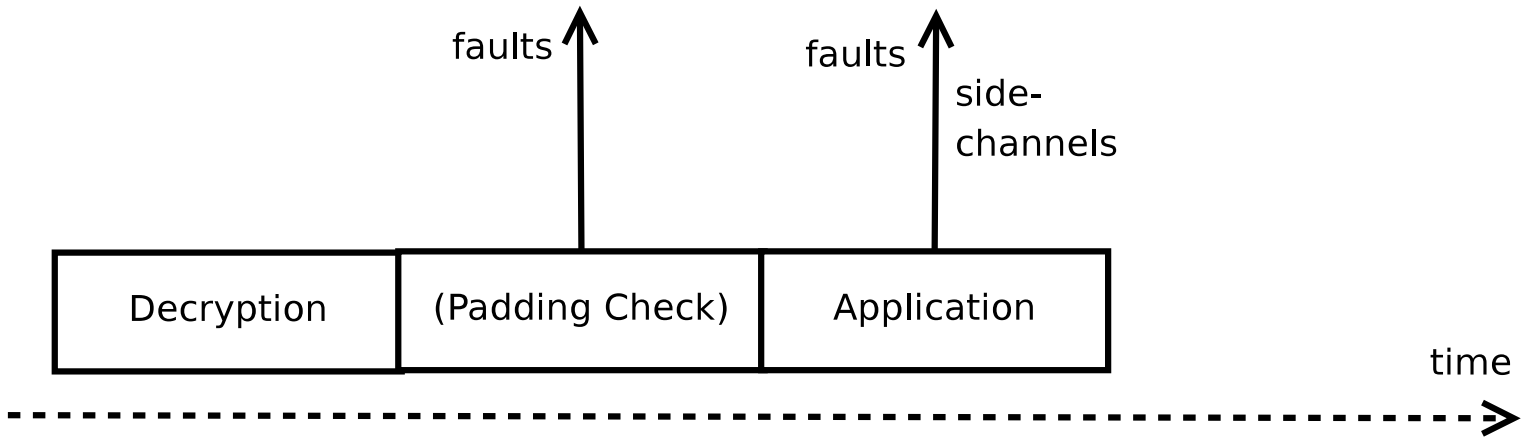
PKCS#7 Padding Decryption Oracle



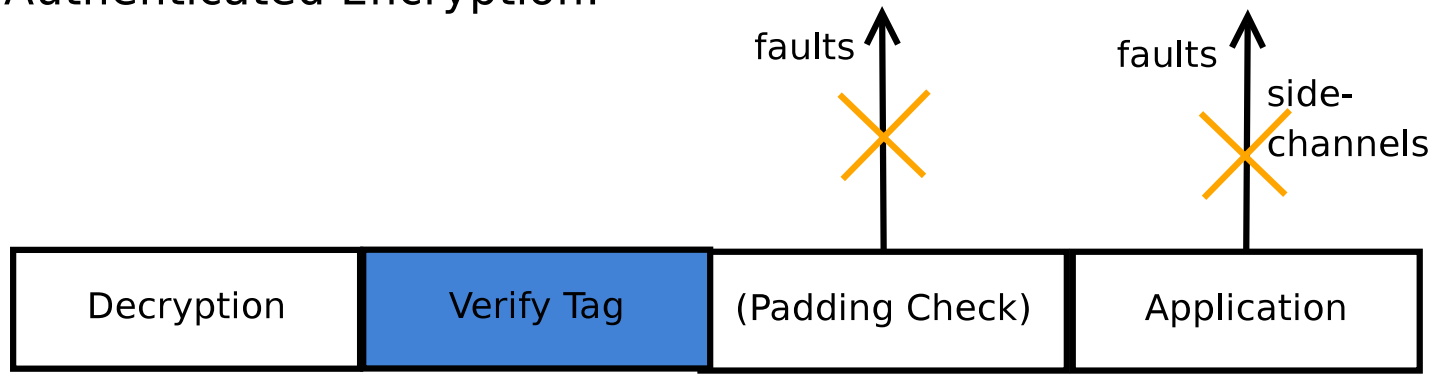
- now the message bytes are attacked one by one
- any block can be placed as the final block
- (due to the block reordering properties of CBC)

Application Oracles - Authenticated and Unauthenticated Encryption

Unauthenticated Encryption:



Authenticated Encryption:



DER as specific Type of Tag-Length-Value Encoding

- When investigating application oracles, the application should be as general as possible
- Length fields are already exploited in Padding Oracle Attacks
- $\rightarrow$ Tag-Length-Value (TLV) Decoding seem most promising
- TLV = nested structure of the concatenation of tag, length, and value fields
- the most general and most widespread standardized type of TLV encoding is the DER encoding for Abstract Syntax Notation 1 (ASN.1)
- DER = "Destinguished Encoding Rules", a binary format
- used for encoding of cryptographic objects such as public and private key, and in the X.509 certificate format

DER as specific Type of Tag-Length-Value Encoding

- When investigating application oracles, the application should be as general as possible
- Length fields are already exploited in Padding Oracle Attacks
- $\rightarrow$ Tag-Length-Value (TLV) Decoding seem most promising
- TLV = nested structure of the concatenation of tag, length, and value fields
- the most general and most widespread standardized type of TLV encoding is the DER encoding for Abstract Syntax Notation 1 (ASN.1)
- DER = "Destinguished Encoding Rules", a binary format
- used for encoding of cryptographic objects such as public and private key, and in the X.509 certificate format

DER as specific Type of Tag-Length-Value Encoding

- When investigating application oracles, the application should be as general as possible
- Length fields are already exploited in Padding Oracle Attacks
- $\longrightarrow$ Tag-Length-Value (TLV) Decoding seem most promising
- TLV = nested structure of the concatenation of tag, length, and value fields
- the most general and most widespread standardized type of TLV encoding is the DER encoding for Abstract Syntax Notation 1 (ASN.1)
- DER = "Destinguished Encoding Rules", a binary format
- used for encoding of cryptographic objects such as public and private key, and in the X.509 certificate format

DER as specific Type of Tag-Length-Value Encoding

- When investigating application oracles, the application should be as general as possible
- Length fields are already exploited in Padding Oracle Attacks
- $\longrightarrow$ Tag-Length-Value (TLV) Decoding seem most promising
- TLV = nested structure of the concatenation of tag, length, and value fields
- the most general and most widespread standardized type of TLV encoding is the DER encoding for Abstract Syntax Notation 1 (ASN.1)
- DER = "Destinguished Encoding Rules", a binary format
- used for encoding of cryptographic objects such as public and private key, and in the X.509 certificate format

DER as specific Type of Tag-Length-Value Encoding

- When investigating application oracles, the application should be as general as possible
- Length fields are already exploited in Padding Oracle Attacks
- $\longrightarrow$ Tag-Length-Value (TLV) Decoding seem most promising
- TLV = nested structure of the concatenation of tag, length, and value fields
- the most general and most widespread standardized type of TLV encoding is the DER encoding for Abstract Syntax Notation 1 (ASN.1)
- DER = "Destinguished Encoding Rules", a binary format
- used for encoding of cryptographic objects such as public and private key, and in the X.509 certificate format

DER as specific Type of Tag-Length-Value Encoding

- When investigating application oracles, the application should be as general as possible
- Length fields are already exploited in Padding Oracle Attacks
- $\longrightarrow$ Tag-Length-Value (TLV) Decoding seem most promising
- TLV = nested structure of the concatenation of tag, length, and value fields
- the most general and most widespread standardized type of TLV encoding is the DER encoding for Abstract Syntax Notation 1 (ASN.1)
- DER = "Destinguished Encoding Rules", a binary format
- used for encoding of cryptographic objects such as public and private key, and in the X.509 certificate format

DER as specific Type of Tag-Length-Value Encoding

- When investigating application oracles, the application should be as general as possible
- Length fields are already exploited in Padding Oracle Attacks
- $\rightarrow$ Tag-Length-Value (TLV) Decoding seem most promising
- TLV = nested structure of the concatenation of tag, length, and value fields
- the most general and most widespread standardized type of TLV encoding is the DER encoding for Abstract Syntax Notation 1 (ASN.1)
- DER = "Destinguished Encoding Rules", a binary format
- used for encoding of cryptographic objects such as public and private key, and in the X.509 certificate format

ECDSA Signatures in ASN.1 and DER encoding

- ECDSA signatures consist of two integers r and s
- we don't need to know anything else about ECDSA for the remainder of the talk
- *ASN.1 notation:*

  ECDSASignature ::= SEQUENCE {
  r INTEGER,
  s INTEGER }
- *DER encoding:*

  $0x30||L_u||0x02||L_r||r_1||r_2||\ldots||r_{L_r}||0x02||L_s||s_1||s_2||\ldots||s_{L_s}$

  $L_u = L_r + L_s + 4$

ECDSA Signatures in ASN.1 and DER encoding

- ECDSA signatures consist of two integers r and s
- we don't need to know anything else about ECDSA for the remainder of the talk
- *ASN.1 notation:*

```
ECDSASignature ::= SEQUENCE {
        r INTEGER,
        s INTEGER }
```

- *DER encoding:*

$$0x30||L_u||0x02||L_r||r_1||r_2||\ldots||r_{L_r}||0x02||L_s||s_1||s_2||\ldots||s_{L_s}$$

$$L_u = L_r + L_s + 4$$

ECDSA Signatures in ASN.1 and DER encoding

- ECDSA signatures consist of two integers r and s
- we don't need to know anything else about ECDSA for the remainder of the talk
- *ASN.1 notation:*
  ECDSASignature ::= SEQUENCE {
  r INTEGER,
  s INTEGER }
- *DER encoding:*
  $0x30||L_u||0x02||L_r||r_1||r_2||\ldots||r_{L_r}||0x02||L_s||s_1||s_2||\ldots||s_{L_s}$
  $L_u = L_r + L_s + 4$

ECDSA Signatures in ASN.1 and DER encoding

- ECDSA signatures consist of two integers r and s
- we don't need to know anything else about ECDSA for the remainder of the talk
- *ASN.1 notation:*
  ECDSASignature ::= SEQUENCE {
  r INTEGER,
  s INTEGER }
- *DER encoding:*
  $0x30||L_u||0x02||L_r||r_1||r_2||\ldots||r_{L_r}||0x02||L_s||s_1||s_2||\ldots||s_{L_s}$
  $L_u = L_r + L_s + 4$

Basic Idea for the DER / Signature Verification Oracle

- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($= 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

Basic Idea for the DER / Signature Verification Oracle

- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($= 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

Basic Idea for the DER / Signature Verification Oracle

- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($= 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

Basic Idea for the DER / Signature Verification Oracle

- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($= 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

Basic Idea for the DER / Signature Verification Oracle

- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($== 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

Basic Idea for the DER / Signature Verification Oracle

- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($== 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

Basic Idea for the DER / Signature Verification Oracle

- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($== 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

Basic Idea for the DER / Signature Verification Oracle

- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($= 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

Basic Idea for the DER / Signature Verification Oracle

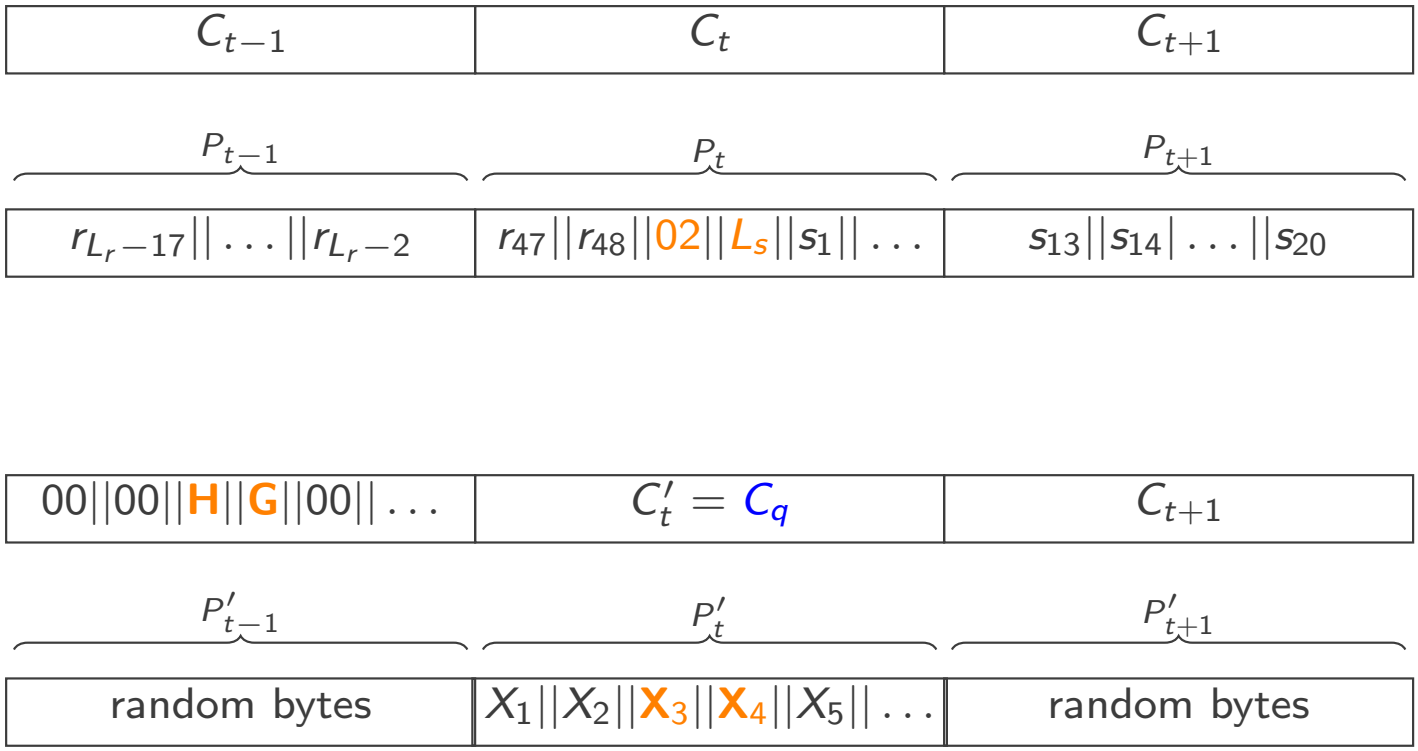
- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($= 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

Basic Idea for the DER / Signature Verification Oracle

- Assumption: a device receives a CBC ciphertext (unauthenticated) containing an ECDSA signed X.509 certificate
  1. the device decrypts the ciphertext
  2. the device first decodes the certificate
  3. then verifies the ECDSA signature
- Oracle queries (time to error) shall indicate whether
  - the DER decoding failed (signature verification skipped)
  - or the signature verification fails
- the tag and length fields of the second integer of the signature (single bytes each) shall act as the oracle bytes:
  - when they are both correct ($= 0x02||L_s$, i.e. one value out of 65336 possible) the decoding succeeds
  - for all other values either the tag will be wrong, or the length will be inconsistent

The Decryption Oracle

- decryption of two bytes inside any target ciphertext block C_q CBC-encrypted under the same key as the oracle ciphertext



The Decryption Oracle

- once those values of the bytes H and G so ,that no decoding error occurs, the corresponding two bytes in the target ciphertext block C_q can be decrypted
- (by also incorporating the ciphertext block preceding C_q in the original ciphertext)
- experimental realization with the Botan C++ library (ECDSA and DER decoding)
  - time to error for DER decoding failure around 200ms
  - time to error for signature verification about twice as high
  - $\rightarrow$ works as a timing attack
  - requires 2^{16} oracle querries on avarage

The Decryption Oracle

- once those values of the bytes H and G so ,that no decoding error occurs, the corresponding two bytes in the target ciphertext block C_q can be decrypted
- (by also incorporating the ciphertext block preceding C_q in the original ciphertext)
- experimental realization with the Botan C++ library (ECDSA and DER decoding)
  - time to error for DER decoding failure around 200ms
  - time to error for signature verification about twice as high
  - $\rightarrow$ works as a timing attack
  - requires 2^{16} oracle querries on avarage

The Decryption Oracle

- once those values of the bytes H and G so ,that no decoding error occurs, the corresponding two bytes in the target ciphertext block C_q can be decrypted
- (by also incorporating the ciphertext block preceding C_q in the original ciphertext)
- experimental realization with the Botan C++ library (ECDSA and DER decoding)
  - time to error for DER decoding failure around 200ms
  - time to error for signature verification about twice as high
  - $\longrightarrow$ works as a timing attack
  - requires 2^{16} oracle querries on avarage

The Decryption Oracle

- once those values of the bytes H and G so ,that no decoding error occurs, the corresponding two bytes in the target ciphertext block C_q can be decrypted
- (by also incorporating the ciphertext block preceding C_q in the original ciphertext)
- experimental realization with the Botan C++ library (ECDSA and DER decoding)
  - time to error for DER decoding failure around 200ms
  - time to error for signature verification about twice as high
  - $\longrightarrow$ works as a timing attack
  - requires 2^{16} oracle querries on avarage

The Decryption Oracle

- once those values of the bytes H and G so ,that no decoding error occurs, the corresponding two bytes in the target ciphertext block C_q can be decrypted
- (by also incorporating the ciphertext block preceding C_q in the original ciphertext)
- experimental realization with the Botan C++ library (ECDSA and DER decoding)
  - time to error for DER decoding failure around 200ms
  - time to error for signature verification about twice as high
  - $\longrightarrow$ works as a timing attack
  - requires 2^{16} oracle querries on avarage

The Decryption Oracle

- once those values of the bytes H and G so ,that no decoding error occurs, the corresponding two bytes in the target ciphertext block C_q can be decrypted
- (by also incorporating the ciphertext block preceding C_q in the original ciphertext)
- experimental realization with the Botan C++ library (ECDSA and DER decoding)
  - time to error for DER decoding failure around 200ms
  - time to error for signature verification about twice as high
  - $\longrightarrow$ works as a timing attack
  - requires 2^{16} oracle querries on avarage

The Decryption Oracle

- once those values of the bytes H and G so ,that no decoding error occurs, the corresponding two bytes in the target ciphertext block C_q can be decrypted
- (by also incorporating the ciphertext block preceding C_q in the original ciphertext)
- experimental realization with the Botan C++ library (ECDSA and DER decoding)
  - time to error for DER decoding failure around 200ms
  - time to error for signature verification about twice as high
  - $\longrightarrow$ works as a timing attack
  - requires 2^{16} oracle querries on avarage

Generalization of the Oracle

- The ECDSA signature verification is a contrived application
- But it is replacable by other applications
- The specific ASN.1/DER structure of ECDSA signature is needed to hide the corrupted blocks entirely for to delay its effect until after the DER decoding
- (this is necessary only for CBC as we will see)

Generalization of the Oracle

- The ECDSA signature verification is a contrived application
- But it is replacable by other applications
- The specific ASN.1/DER structure of ECDSA signature is needed to hide the corrupted blocks entirely for to delay its effect until after the DER decoding
- (this is necessary only for CBC as we will see)

Generalization of the Oracle

- The ECDSA signature verification is a contrived application
- But it is replacable by other applications
- The specific ASN.1/DER structure of ECDSA signature is needed to hide the corrupted blocks entirely for to delay its effect until after the DER decoding
- (this is necessary only for CBC as we will see)

Generalization of the Oracle

- The ECDSA signature verification is a contrived application
- But it is replacable by other applications
- The specific ASN.1/DER structure of ECDSA signature is needed to hide the corrupted blocks entirely for to delay its effect until after the DER decoding
- (this is necessary only for CBC as we will see)

Mobility of the Oracle Bytes

- So far we can only decrypt two bytes per ciphertext block
- one way to decrypt more bytes is to find DER oracles with different offsets inside the block
- another way is to exploit multi-byte tag and length fields:
  - multi-byte tag fields:

$$T_{\text{long}} = xxx11111_2 || 1xxx\,xxxx_2 || \ldots || 0xxx\,xxxx_2$$

  - multi-byte length fields ("long definite length"):

$$L_{\text{long}} = 1\#\#\#\#\#_2 || L_1 || L_2 || \cdots || L_{(0\#\#\#\#\#_2)}$$

- at least in the Botan Library's DER decoder implementation, in this way up to 8 oracle bytes per block are possible

Mobility of the Oracle Bytes

- So far we can only decrypt two bytes per ciphertext block
- one way to decrypt more bytes is to find DER oracles with different offsets inside the block
- another way is to exploit multi-byte tag and length fields:
  - multi-byte tag fields:

$$T_{\text{long}} = \text{xxx}11111_2 || 1\text{xxx xxxx}_2 || \ldots || 0\text{xxx xxxx}_2$$

  - multi-byte length fields ("long definite length"):

$$L_{\text{long}} = 1\#\#\#\#\#_2 || L_1 || L_2 || \cdots || L_{(0\#\#\#\#\#_2)}$$

- at least in the Botan Library's DER decoder implementation, in this way up to 8 oracle bytes per block are possible

Mobility of the Oracle Bytes

- So far we can only decrypt two bytes per ciphertext block
- one way to decrypt more bytes is to find DER oracles with different offsets inside the block
- another way is to exploit multi-byte tag and length fields:
  - multi-byte tag fields:

$$T_{\text{long}} = xxx1\,1111_2 || 1xxx\,xxxx_2 || \ldots || 0xxx\,xxxx_2$$

  - multi-byte length fields ("long definite length"):

$$L_{\text{long}} = 1lll\,llll_2 || L_1 || L_2 || \ldots || L_{(0lll\,llll_2)}$$

- at least in the Botan Library's DER decoder implementation, in this way up to 8 oracle bytes per block are possible

Mobility of the Oracle Bytes

- So far we can only decrypt two bytes per ciphertext block
- one way to decrypt more bytes is to find DER oracles with different offsets inside the block
- another way is to exploit multi-byte tag and length fields:
  - multi-byte tag fields:

$$T_{\text{long}} = xxx1\,1111_2 || 1xxx\,xxxx_2 || \ldots || 0xxx\,xxxx_2$$

  - multi-byte length fields ("long definite length"):

$$L_{\text{long}} = 1lll\,llll_2 || L_1 || L_2 || \ldots || L_{(0lll\,llll_2)}$$

- at least in the Botan Library's DER decoder implementation, in this way up to 8 oracle bytes per block are possible

Mobility of the Oracle Bytes

- So far we can only decrypt two bytes per ciphertext block
- one way to decrypt more bytes is to find DER oracles with different offsets inside the block
- another way is to exploit multi-byte tag and length fields:
  - multi-byte tag fields:

$$T_{\text{long}} = xxx1\,1111_2 || 1xxx\,xxxx_2 || \ldots || 0xxx\,xxxx_2$$

  - multi-byte length fields ("long definite length"):

$$L_{\text{long}} = 1lll\,llll_2 || L_1 || L_2 || \ldots || L_{(0lll\,llll_2)}$$

- at least in the Botan Library's DER decoder implementation, in this way up to 8 oracle bytes per block are possible

Mobility of the Oracle Bytes

- So far we can only decrypt two bytes per ciphertext block
- one way to decrypt more bytes is to find DER oracles with different offsets inside the block
- another way is to exploit multi-byte tag and length fields:
  - multi-byte tag fields:

$$T_{\text{long}} = xxx1\,1111_2 || 1xxx\,xxxx_2 || \ldots || 0xxx\,xxxx_2$$

  - multi-byte length fields ("long definite length"):

$$L_{\text{long}} = 1lll\,llll_2 || L_1 || L_2 || \ldots || L_{(0lll\,llll_2)}$$

- at least in the Botan Library's DER decoder implementation, in this way up to 8 oracle bytes per block are possible

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - → for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given
    - due to the plain XOR-homomorphicity DER length fields in the nested structure can be consistently modified
    - → an OpenPGP Signature Verification oracle, e.g., MSBs of the first integer r in the ECDSA signature could be arbitrarily manipulated

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - → for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given
    - due to the plain XOR-homomorphicity DER length fields in the nested structure can be consistently manipulated
    - → an OpenPGP Signature Verification oracle e.g. MPI size of the first integer r in the ECDSA signature could also be manipulated

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - → for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given
    - due to the plain XOR homomorphicity DER length fields in the nested structure can be consistently manipulated
    - → an DER/X.509 Signature Verification Oracle e.g. MSs of the first integer in the ECDSA signature could also be decrypted

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - $\longrightarrow$ for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given:
      - due to the plain XOR-homomorphicity, DER length fields in the nested structure can be consistently modified
      - $\longrightarrow$ in DER / Signature Verification oracle, the size of the first integer r in the ECDSA signature could be artificially increased

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - $\longrightarrow$ for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given:
      - due to the plain XOR-homomorphicity, DER length fields in the nested structure can be consistently modified
      - $\longrightarrow$ in DER / Signature Verification oracle, the size of the first integer r in the ECDSA signature could be artificially increased

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - $\longrightarrow$ for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given:
    - due to the plain XOR-homomorphicity, DER length fields in the nested structure can be consistently modified
    - $\longrightarrow$ in DER / Signature Verification oracle, the size of the first integer r in the ECDSA signature could be artificially increased

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - $\longrightarrow$ for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given:
    - due to the plain XOR-homomorphicity, DER length fields in the nested structure can be consistently modified
    - $\longrightarrow$ in DER / Signature Verification oracle, the size of the first integer r in the ECDSA signature could be artificially increased

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - $\longrightarrow$ for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given:
    - due to the plain XOR-homomorphicity, DER length fields in the nested structure can be consistently modified
    - $\longrightarrow$ in DER / Signature Verification oracle, the size of the first integer r in the ECDSA signature could be artificially increased

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - $\longrightarrow$ for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given:
      - due to the plain XOR-homomorphicity, DER length fields in the nested structure can be consistently modified
      - $\longrightarrow$ in DER / Signature Verification oracle, the size of the first integer r in the ECDSA signature could be artificially increased

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - $\longrightarrow$ for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given:
    - due to the plain XOR-homomorphicity, DER length fields in the nested structure can be consistently modified
    - $\longrightarrow$ in DER / Signature Verification oracle, the size of the first integer r in the ECDSA signature could be artificially increased

Other Block Cipher Modes of Operation

- CBC
  - offers an XOR-homomorphicity with corruption of whole blocks
  - allows for reordering of ciphertext blocks (with meaningful decryption)
- The "Stream Cipher Modes" CTR, CFB and OFB
  - offer a plain XOR-homomorphicity without any corruption
  - do not allow for the reordering of ciphertext blocks
  - $\longrightarrow$ for the DER based oracles this means
    - only blocks following the DER object in the same ciphertext can be decrypted
    - but a far greater (forward) mobility of the oracle byte position is given:
    - due to the plain XOR-homomorphicity, DER length fields in the nested structure can be consistently modified
    - $\longrightarrow$ in DER / Signature Verification oracle, the size of the first integer r in the ECDSA signature could be artificially increased

DER / Memory Allocation Oracles

- During the attacks based on the DER / Signature Verification oracle, large delays and heavy system load could be observed, as well as memory exhaustion exceptions by the Botan Library
- explanation: long definite length fields

$$L_{\text{long}} = 1\text{/// ////}_2||L_1||L_2||\cdots||L_{(0\text{/// ////}_2)}$$

can cause large memory allocations
  - that cannot be satisfied
  - or that can be satisfied but with long response times
  - the Botan DER decoder limits the value of /// ////_2 to 4 (32 bit addressable memory size)

DER / Memory Allocation Oracles

- During the attacks based on the DER / Signature Verification oracle, large delays and heavy system load could be observed, as well as memory exhaustion exceptions by the Botan Library
- explanation: long definite length fields

$$L_{\text{long}} = 1\text{/// ////}_2 || L_1 || L_2 || \ldots || L_{(0\text{/// ////}_2)}$$

can cause large memory allocations

- that cannot be satisfied
- or that can be satisfied but with long response times
- the Botan DER decoder limits the value of /// ////$_2$ to 4 (32 bit addressable memory size)

DER / Memory Allocation Oracles

- During the attacks based on the DER / Signature Verification oracle, large delays and heavy system load could be observed, as well as memory exhaustion exceptions by the Botan Library
- explanation: long definite length fields

$$L_{\text{long}} = 1\text{/// ////}_2||L_1||L_2||\ldots||L_{(0\text{/// ////}_2)}$$

can cause large memory allocations

  - that cannot be satisfied
  - or that can be satisfied but with long response times
  - the Botan DER decoder limits the value of /// ////$_2$ to 4 (32 bit addressable memory size)

DER / Memory Allocation Oracles

- During the attacks based on the DER / Signature Verification oracle, large delays and heavy system load could be observed, as well as memory exhaustion exceptions by the Botan Library
- explanation: long definite length fields

$$L_{\text{long}} = 1\text{/// ////}_2||L_1||L_2||\ldots||L_{(0\text{/// ////}_2)}$$

can cause large memory allocations

  - that cannot be satisfied
  - or that can be satisfied but with long response times
  - the Botan DER decoder limits the value of /// ////$_2$ to 4 (32 bit addressable memory size)

DER / Memory Allocation Oracles

- During the attacks based on the DER / Signature Verification oracle, large delays and heavy system load could be observed, as well as memory exhaustion exceptions by the Botan Library
- explanation: long definite length fields

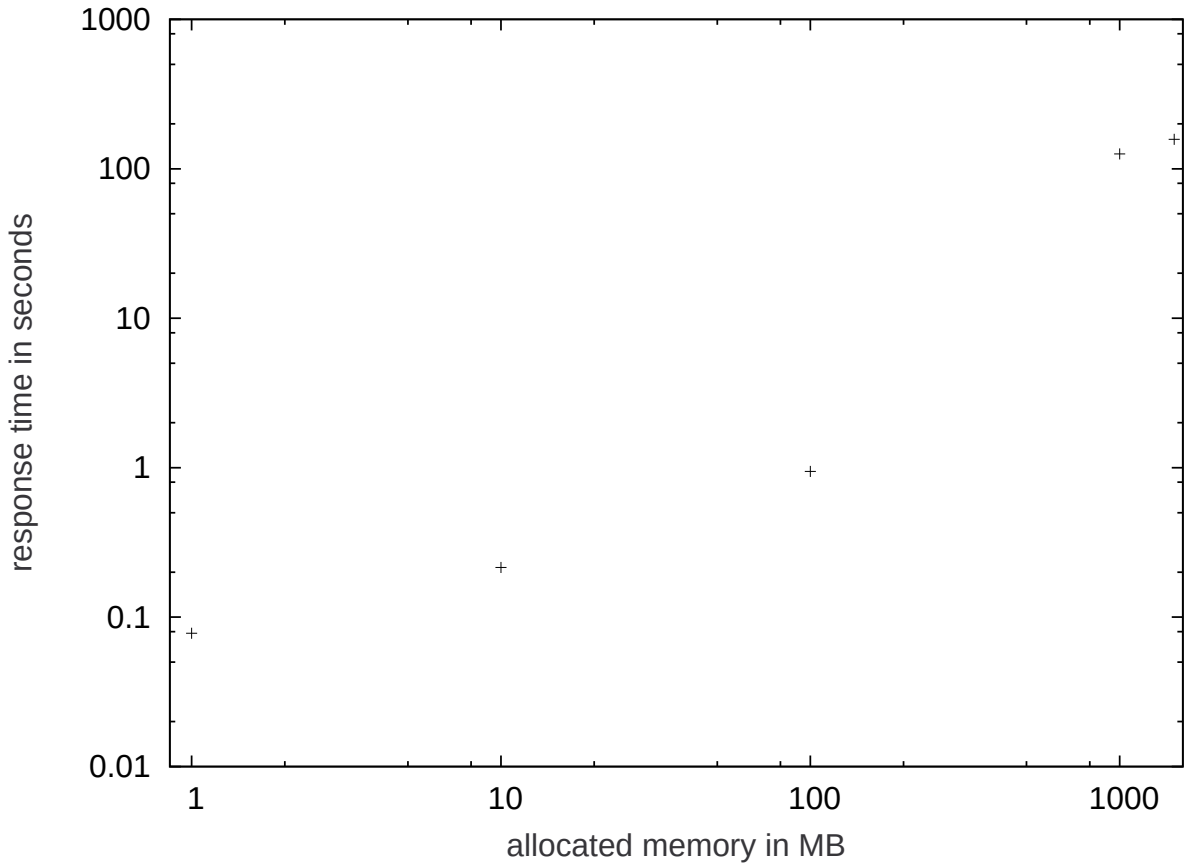
$$L_{\text{long}} = 1\text{/// ////}_2 || L_1 || L_2 || \ldots || L_{(0\text{/// ////}_2)}$$

can cause large memory allocations

  - that cannot be satisfied
  - or that can be satisfied but with long response times
  - the Botan DER decoder limits the value of /// ////$_2$ to 4 (32 bit addressable memory size)

Timing Effects of Memory Allocation Requests

- we measured the response time for the allocation memory chunks of different (large) sizes on a Linux computer with 1GB RAM



Authenticated Encryption

- Authenticated Encryption removes application oracles
- (and also padding oracles if implemented correctly)
- two variants of Authenticated Encryption
  - dedicated authenticated Block Cipher Modes of Operation
  - compination of any Block Cipher Mode with a Message Authentication Code (MAC)
- in any case, Authenticated Encryption is reached by appending a tag at end of the ciphertext
- but there are some pitfalls . . .

Authenticated Encryption

- Authenticated Encryption removes application oracles
- (and also padding oracles if implemented correctly)
- two variants of Authenticated Encryption
  - dedicated authenticated Block Cipher Modes of Operation
  - compination of any Block Cipher Mode with a Message Authentication Code (MAC)
- in any case, Authenticated Encryption is reached by appending a tag at end of the ciphertext
- but there are some pitfalls . . .

Authenticated Encryption

- Authenticated Encryption removes application oracles
- (and also padding oracles if implemented correctly)
- two variants of Authenticated Encryption
  - dedicated authenticated Block Cipher Modes of Operation
  - compination of any Block Cipher Mode with a Message Authentication Code (MAC)
- in any case, Authenticated Encryption is reached by appending a tag at end of the ciphertext
- but there are some pitfalls . . .

Authenticated Encryption

- Authenticated Encryption removes application oracles
- (and also padding oracles if implemented correctly)
- two variants of Authenticated Encryption
  - dedicated authenticated Block Cipher Modes of Operation
  - compination of any Block Cipher Mode with a Message Authentication Code (MAC)
- in any case, Authenticated Encryption is reached by appending a tag at end of the ciphertext
- but there are some pitfalls . . .

Authenticated Encryption

- Authenticated Encryption removes application oracles
- (and also padding oracles if implemented correctly)
- two variants of Authenticated Encryption
  - dedicated authenticated Block Cipher Modes of Operation
  - compination of any Block Cipher Mode with a Message Authentication Code (MAC)
- in any case, Authenticated Encryption is reached by appending a tag at end of the ciphertext
- but there are some pitfalls . . .

Authenticated Encryption

- Authenticated Encryption removes application oracles
- (and also padding oracles if implemented correctly)
- two variants of Authenticated Encryption
  - dedicated authenticated Block Cipher Modes of Operation
  - compination of any Block Cipher Mode with a Message Authentication Code (MAC)
- in any case, Authenticated Encryption is reached by appending a tag at end of the ciphertext
- but there are some pitfalls . . .

Authenticated Encryption

- Authenticated Encryption removes application oracles
- (and also padding oracles if implemented correctly)
- two variants of Authenticated Encryption
  - dedicated authenticated Block Cipher Modes of Operation
  - compination of any Block Cipher Mode with a Message Authentication Code (MAC)
- in any case, Authenticated Encryption is reached by appending a tag at end of the ciphertext
- but there are some pitfalls . . .

Implementation of Authenticated Enryption in Open Source Cryptographic Libraries

- we analyzed three open source cryptographic libraries:
- Botan (C++), Crypto++ (C++), BouncyCastle (Java)
- with respect to the implementation of the EAX Authenticated Block Cipher Mode:
- they allow on-line decryption in the EAX mode
- thus decypted plaintext is made available before the verification of the integrity!

Implementation of Authenticated Enryption in Open Source Cryptographic Libraries

- we analyzed three open source cryptographic libraries:
- Botan (C++), Crypto++ (C++), BouncyCastle (Java)
- with respect to the implementation of the EAX Authenticated Block Cipher Mode:
- they allow on-line decryption in the EAX mode
- thus decypted plaintext is made available before the verification of the integrity!

Implementation of Authenticated Enryption in Open Source Cryptographic Libraries

- we analyzed three open source cryptographic libraries:
- Botan (C++), Crypto++ (C++), BouncyCastle (Java)
- with respect to the implementation of the EAX Authenticated Block Cipher Mode:
- they allow on-line decryption in the EAX mode
- thus decypted plaintext is made available before the verification of the integrity!

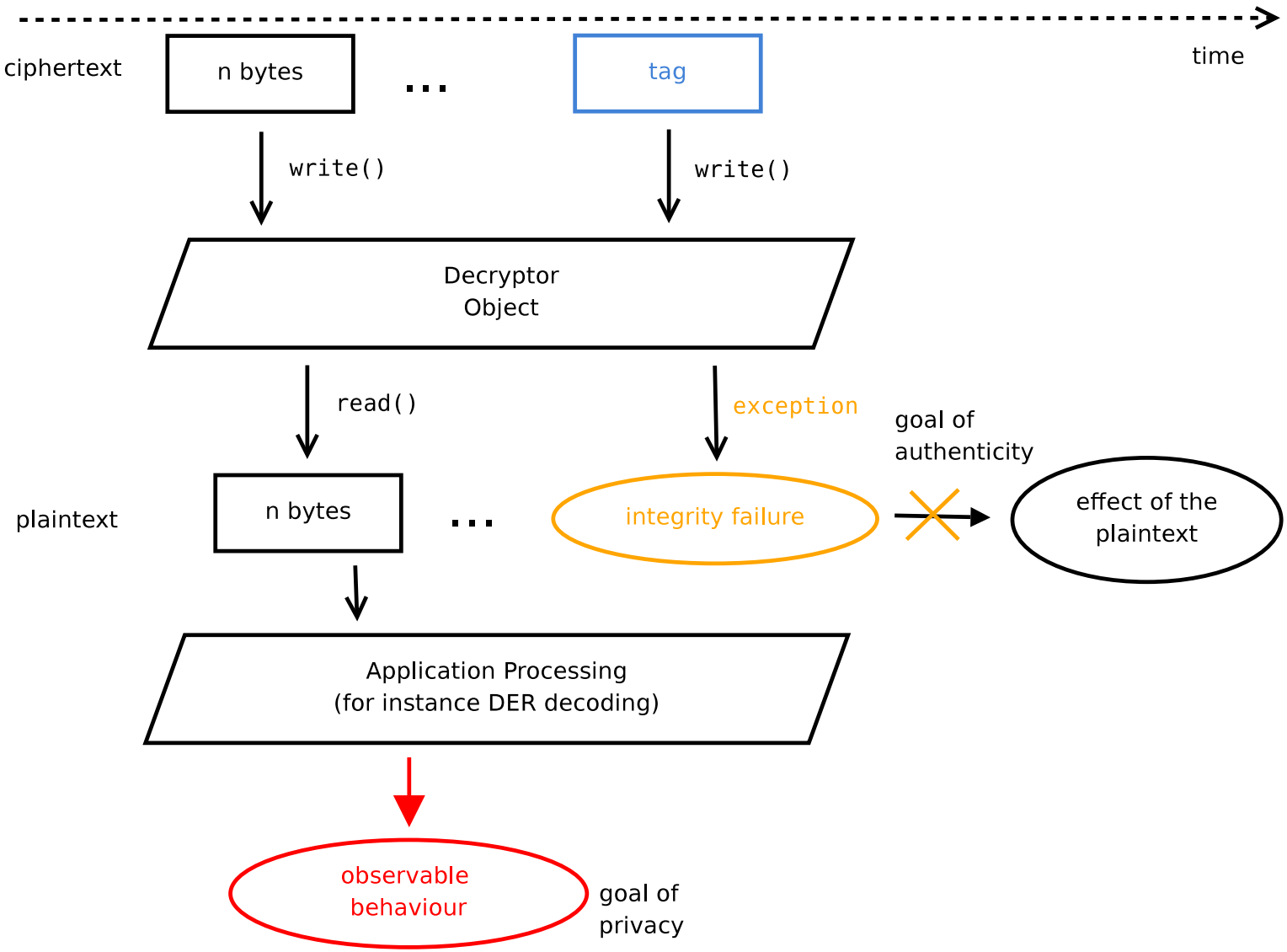
Implementation of Authenticated Enryption in Open Source Cryptographic Libraries

- we analyzed three open source cryptographic libraries:
- Botan (C++), Crypto++ (C++), BouncyCastle (Java)
- with respect to the implementation of the EAX Authenticated Block Cipher Mode:
- they allow on-line decryption in the EAX mode
- thus decypted plaintext is made available before the verification of the integrity!

Implementation of Authenticated Enryption in Open Source Cryptographic Libraries

- we analyzed three open source cryptographic libraries:
- Botan (C++), Crypto++ (C++), BouncyCastle (Java)
- with respect to the implementation of the EAX Authenticated Block Cipher Mode:
- they allow on-line decryption in the EAX mode
- thus decypted plaintext is made available before the verification of the integrity!

Decryption in Authenticated Encryption is not on-line



Conclusion

- We have seen the dangers of decryption oracles through TLV decoding
- specifically, the DER / Memory Allocation based oracles do not demand any other application
- Authenticated Encryption certainly is the solution and in general necessary
- but it must be implemented correctly – most open source libraries enable incorrect use of Authenticated Encryption that threatens the goal of privacy (but not necessarily that of authenticity!)

Conclusion

- We have seen the dangers of decryption oracles through TLV decoding
- specifically, the DER / Memory Allocation based oracles do not demand any other application
- Authenticated Encryption certainly is the solution and in general necessary
- but it must be implemented correctly – most open source libraries enable incorrect use of Authenticated Encryption that threatens the goal of privacy (but not necessarily that of authenticity!)

Conclusion

- We have seen the dangers of decryption oracles through TLV decoding
- specifically, the DER / Memory Allocation based oracles do not demand any other application
- Authenticated Encryption certainly is the solution and in general necessary
- but it must be implemented correctly – most open source libraries enable incorrect use of Authenticated Encryption that threatens the goal of privacy (but not necessarily that of authenticity!)

Conclusion

- We have seen the dangers of decryption oracles through TLV decoding
- specifically, the DER / Memory Allocation based oracles do not demand any other application
- Authenticated Encryption certainly is the solution and in general necessary
- but it must be implemented correctly – most open source libraries enable incorrect use of Authenticated Encryption that threatens the goal of privacy (but not necessarily that of authenticity!)

- Thank You!