Legacy Encryption Downgrade Attacks against LibrePGP and CMS

Work in the scope of Project 480 – "PQC@Thunderbird"

Legacy Encryption Downgrade Attacks against LibrePGP and CMS

- work in the scope of Project 480 – "PQC@Thunderbird"
- Legacy encryption: encryption modes w/o integrity protection
- Target "modern encryption" modes in
  - Cryptographic Message Syntax
    - X.509 certificates
    - basis for S/MIME
  - LibrePGP
    - a recent fork of the OpenPGP standard

Legacy Encryption Downgrade Attacks against LibrePGP and CMS

- work in the scope of Project 480 – “PQC@Thunderbird”
- Legacy encryption: encryption modes w/o integrity protection
- Target “modern encryption” modes in
  - Cryptographic Message Syntax
    - X.509 certificates
    - basis for S/MIME
  - LibrePGP
    - a recent fork of the OpenPGP standard

Legacy Encryption Downgrade Attacks against LibrePGP and CMS

- work in the scope of Project 480 – "PQC@Thunderbird"
- Legacy encryption: encryption modes w/o integrity protection
- Target "modern encryption" modes in
  - Cryptographic Message Syntax
    - X.509 certificates
    - basis for S/MIME
  - LibrePGP
    - a recent fork of the OpenPGP standard

Legacy Encryption Downgrade Attacks against LibrePGP and CMS

MTG

- work in the scope of Project 480 – "PQC@Thunderbird"
- Legacy encryption: encryption modes w/o integrity protection
- Target "modern encryption" modes in
  - Cryptographic Message Syntax
    - X.509 certificates
    - basis for S/MIME
  - LibrePGP
    - a recent fork of the OpenPGP standard

Legacy Encryption Downgrade Attacks against LibrePGP and CMS

MTG

- work in the scope of Project 480 – "PQC@Thunderbird"
- Legacy encryption: encryption modes w/o integrity protection
- Target "modern encryption" modes in
  - Cryptographic Message Syntax
    - X.509 certificates
    - basis for S/MIME
  - LibrePGP
    - a recent fork of the OpenPGP standard

Legacy Encryption Downgrade Attacks against LibrePGP and CMS

- work in the scope of Project 480 – “PQC@Thunderbird”
- Legacy encryption: encryption modes w/o integrity protection
- Target “modern encryption” modes in
  - Cryptographic Message Syntax
    - X.509 certificates
    - basis for S/MIME
  - LibrePGP
    - a recent fork of the OpenPGP standard

Legacy Encryption Downgrade Attacks against LibrePGP and CMS

MTG

- work in the scope of Project 480 – "PQC@Thunderbird"
- Legacy encryption: encryption modes w/o integrity protection
- Target "modern encryption" modes in
  - Cryptographic Message Syntax
    - X.509 certificates
    - basis for S/MIME
  - LibrePGP
    - a recent fork of the OpenPGP standard

Legacy Encryption Downgrade Attacks against LibrePGP and CMS

MTG

- work in the scope of Project 480 – "PQC@Thunderbird"
- Legacy encryption: encryption modes w/o integrity protection
- Target "modern encryption" modes in
  - Cryptographic Message Syntax
    - X.509 certificates
    - basis for S/MIME
  - LibrePGP
    - a recent fork of the OpenPGP standard

Legacy Encryption Downgrade Attacks against LibrePGP and CMS

MTG

Introduction

Decryption Oracle Attacks against Cryptographic Message Syntax

Plaintext manipulation attacks against LibrePGP AEAD

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

Legacy Mode Downgrade Attacks against AES Key Wrap

Conclusion

Classical Decryption Oracle Attacks

- Goal: decryption of messages
- Asymmetric
  - Bleichenbacher's padding oracle attack against RSA with PKCS#1 v1.5 padding
  - Manger's Attack against RSA with OAEP padding
- Symmetric
  - Vaudenay's CBC padding oracle attacks
    - error answer based on correct or incorrect padding
  - Format oracles
    - plaintext processing checks format (character set, etc.)

Classical Decryption Oracle Attacks

- Goal: decryption of messages
- Asymmetric
  - Bleichenbacher's padding oracle attack against RSA with PKCS#1 v1.5 padding
  - Manger's Attack against RSA with OAEP padding
- Symmetric
  - Vaudenay's CBC padding oracle attacks
    - error answer based on correct or incorrect padding
  - Format oracles
    - plaintext processing checks format (character set, etc.)

Classical Decryption Oracle Attacks

- Goal: decryption of messages
- Asymmetric
  - Bleichenbacher's padding oracle attack against RSA with PKCS#1 v1.5 padding
  - Manger's Attack against RSA with OAEP padding
- Symmetric
  - Vaudenay's CBC padding oracle attacks
    - → error answer based on correct or incorrect padding
  - Format oracles
    - → plaintext processing checks format (character set, etc.)

Classical Decryption Oracle Attacks

- Goal: decryption of messages
- Asymmetric
  - Bleichenbacher's padding oracle attack against RSA with PKCS#1 v1.5 padding
  - Manger's Attack against RSA with OAEP padding
- Symmetric
  - Vaudenay's CBC padding oracle attacks
    - error answer based on correct or incorrect padding
  - Format oracles
    - plaintext processing checks format (character set, etc.)

Classical Decryption Oracle Attacks

MTG

- Goal: decryption of messages
- Asymmetric
  - Bleichenbacher's padding oracle attack against RSA with PKCS#1 v1.5 padding
  - Manger's Attack against RSA with OAEP padding
- Symmetric
  - Vaudenay's CBC padding oracle attacks
    - error answer based on correct or incorrect padding
  - Format oracles
    - plaintext processing checks format (character set, etc.)

Classical Decryption Oracle Attacks

- Goal: decryption of messages
- Asymmetric
  - Bleichenbacher's padding oracle attack against RSA with PKCS#1 v1.5 padding
  - Manger's Attack against RSA with OAEP padding
- Symmetric
  - Vaudenay's CBC padding oracle attacks
    - error answer based on correct or incorrect padding
  - Format oracles
    - plaintext processing checks format (character set, etc.)

Classical Decryption Oracle Attacks

- Goal: decryption of messages
- Asymmetric
  - Bleichenbacher's padding oracle attack against RSA with PKCS#1 v1.5 padding
  - Manger's Attack against RSA with OAEP padding
- Symmetric
  - Vaudenay's CBC padding oracle attacks
    - error answer based on correct or incorrect padding
  - Format oracles
    - plaintext processing checks format (character set, etc.)

Classical Decryption Oracle Attacks

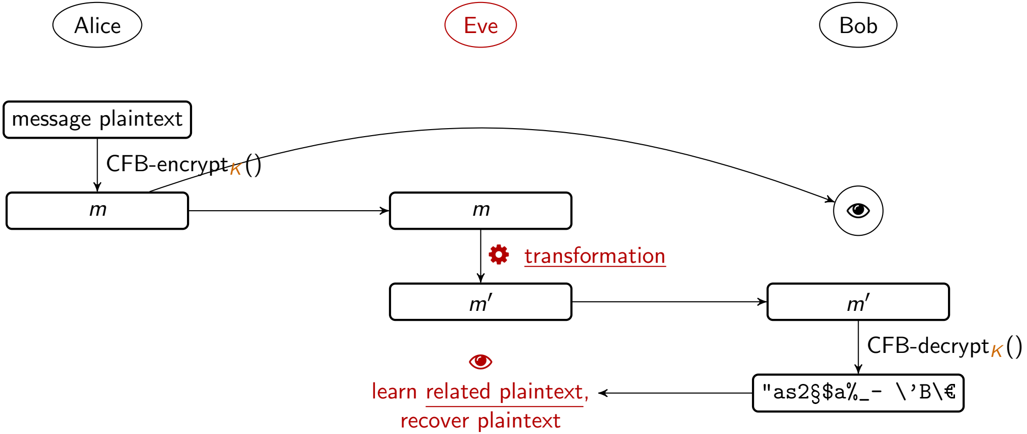
- Goal: decryption of messages
- Asymmetric
  - Bleichenbacher's padding oracle attack against RSA with PKCS#1 v1.5 padding
  - Manger's Attack against RSA with OAEP padding
- Symmetric
  - Vaudenay's CBC padding oracle attacks
    - error answer based on correct or incorrect padding
  - Format oracles
    - plaintext processing checks format (character set, etc.)

Classical Decryption Oracle Attacks

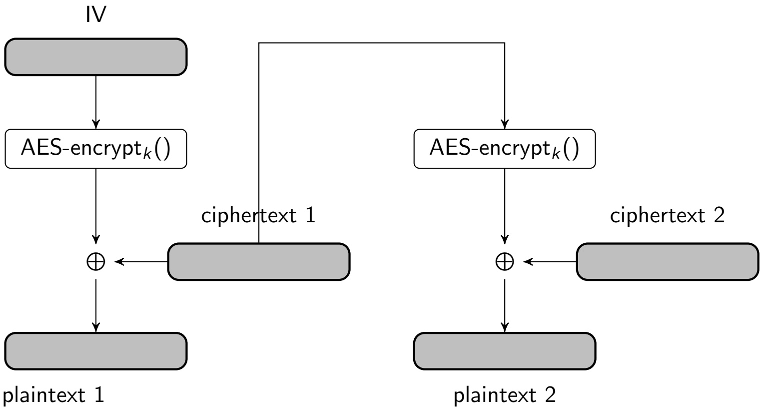
MTG

- Goal: decryption of messages
- Asymmetric
  - Bleichenbacher's padding oracle attack against RSA with PKCS#1 v1.5 padding
  - Manger's Attack against RSA with OAEP padding
- Symmetric
  - Vaudenay's CBC padding oracle attacks
    - error answer based on correct or incorrect padding
  - Format oracles
    - plaintext processing checks format (character set, etc.)

Decryption Oracle Attack against OpenPGP SED Packets

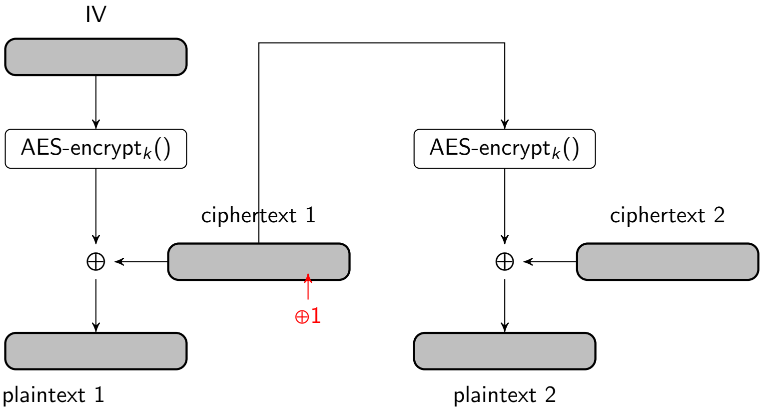


CFB decryption and its malleability

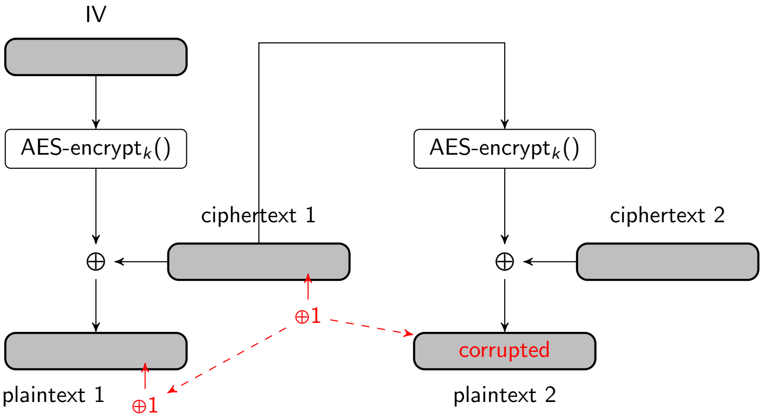


CFB decryption and its malleability

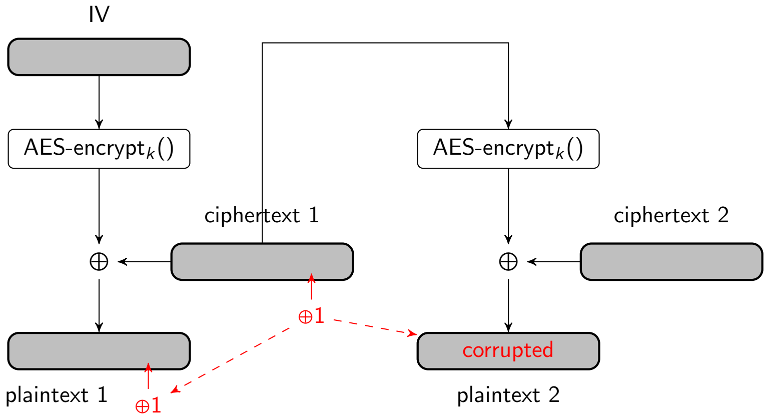
MTG



CFB decryption and its malleability

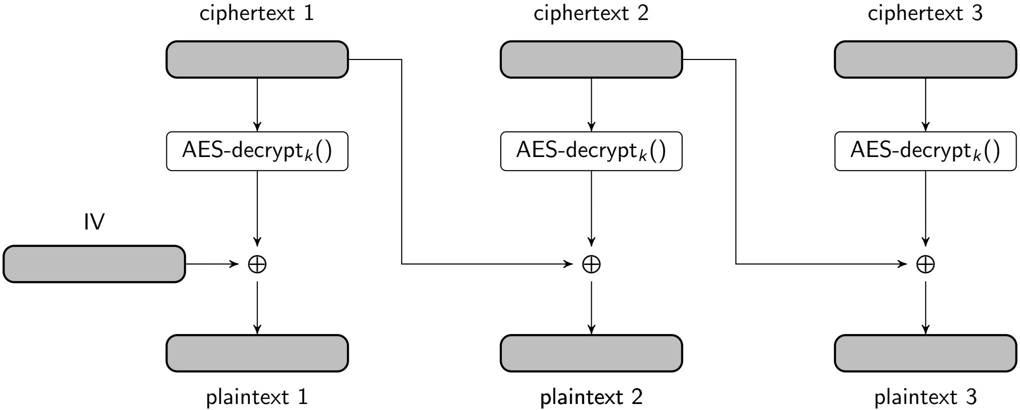


CFB decryption and its malleability

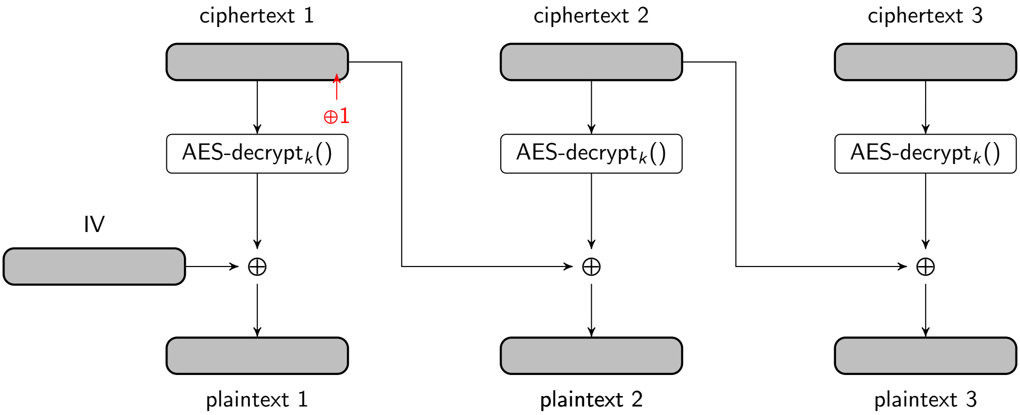


ability to mask actual
plaintext in a reversible way

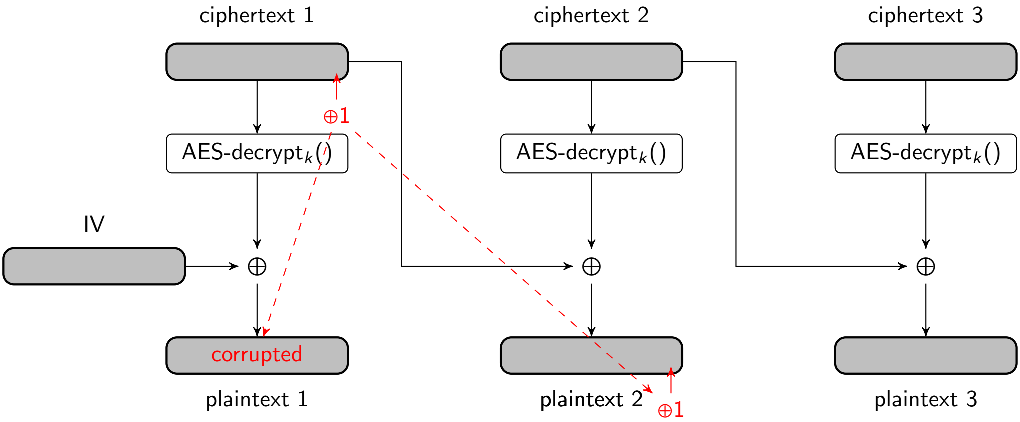
CBC decryption and its malleability



CBC decryption and its malleability

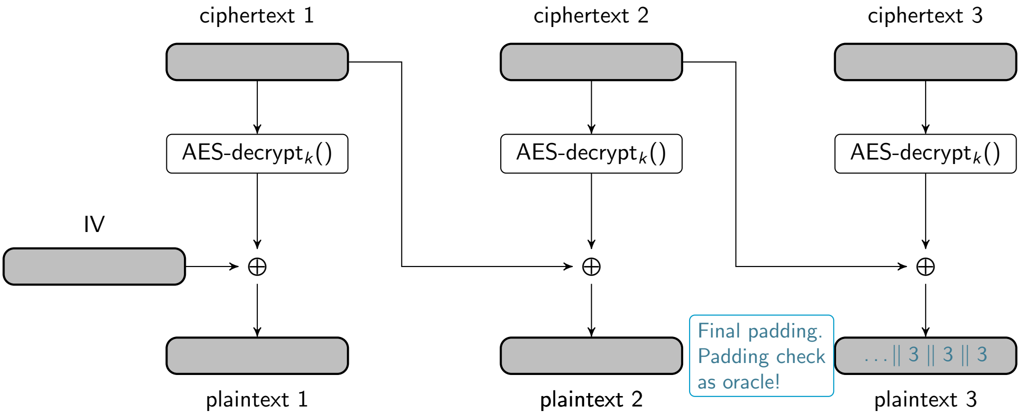


CBC decryption and its malleability

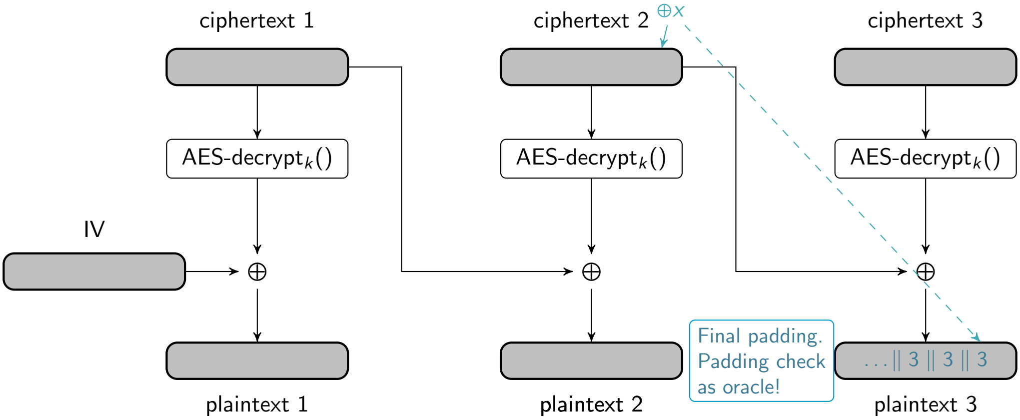


CBC decryption and its malleability

MTG

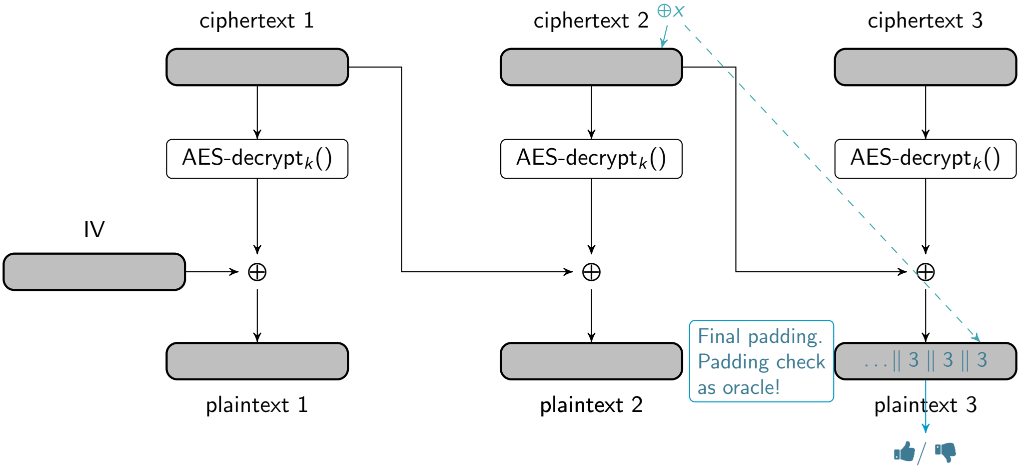


CBC decryption and its malleability

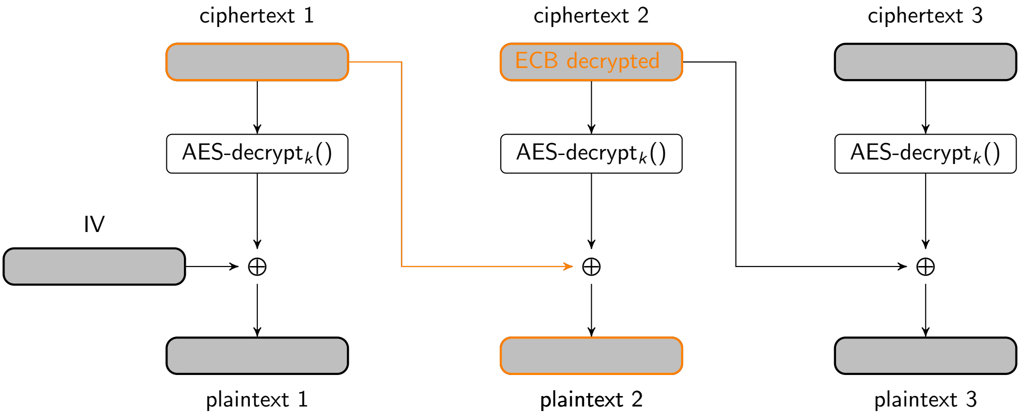


CBC decryption and its malleability

MTG



CBC decryption and its malleability



CBC decryption oracle = ECB decryption oracle

Decryption oracle attacks against modern cipher modes

MTG

- Classical oracle attacks
  - ◎ target ciphertext: CFB
  - 🔓 ciphertext decrypted by the oracle: CFB
- Oracle attacks using downgrades
  - ◎ target ciphertext: AEAD or AES Key Wrap
  - 🔓 ciphertext decrypted by the oracle: CFB (or CBC)

Decryption oracle attacks against modern cipher modes

- Classical oracle attacks
  - target ciphertext: CFB
  - ciphertext decrypted by the oracle: CFB
- Oracle attacks using downgrades
  - target ciphertext: AEAD or AES Key Wrap
  - ciphertext decrypted by the oracle: CFB (or CBC)

Decryption oracle attacks against modern cipher modes

- Classical oracle attacks
  - ◎ target ciphertext: CFB
  - 🔓 ciphertext decrypted by the oracle: CFB
- Oracle attacks using downgrades
  - ◎ target ciphertext: AEAD or AES Key Wrap
  - 🔓 ciphertext decrypted by the oracle: CFB (or CBC)

Decryption oracle attacks against modern cipher modes

MTG

- Classical oracle attacks
  - ◎ target ciphertext: CFB
  - 🔓 ciphertext decrypted by the oracle: CFB
- Oracle attacks using downgrades
  - ◎ target ciphertext: AEAD or AES Key Wrap
  - 🔓 ciphertext decrypted by the oracle: CFB (or CBC)

Decryption oracle attacks against modern cipher modes

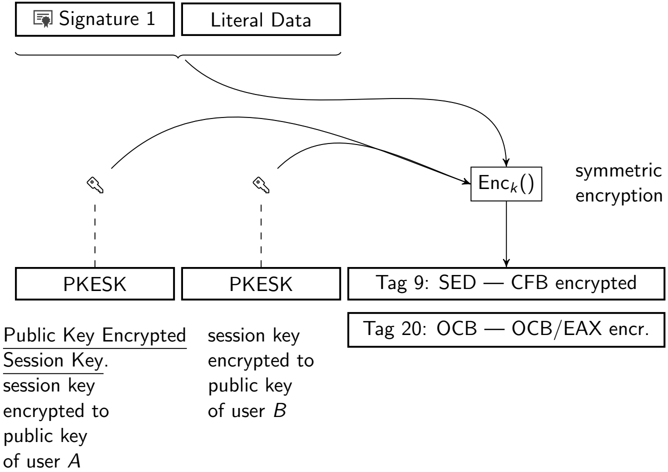
MTG

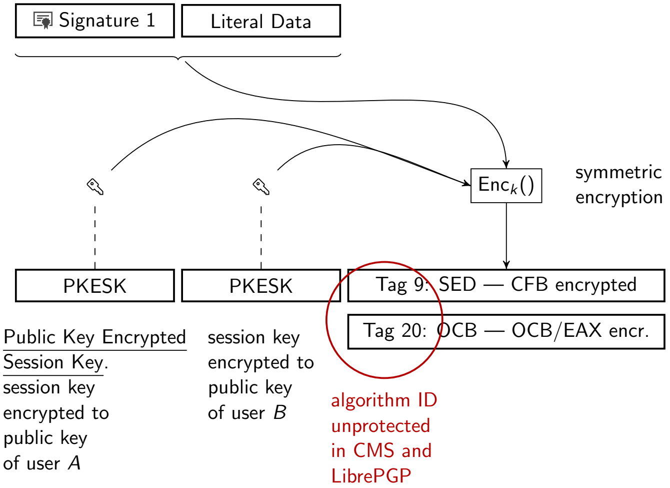
- Classical oracle attacks
  - ◎ target ciphertext: CFB
  - 🔓 ciphertext decrypted by the oracle: CFB
- Oracle attacks using downgrades
  - ◎ target ciphertext: AEAD or AES Key Wrap
  - 🔓 ciphertext decrypted by the oracle: CFB (or CBC)

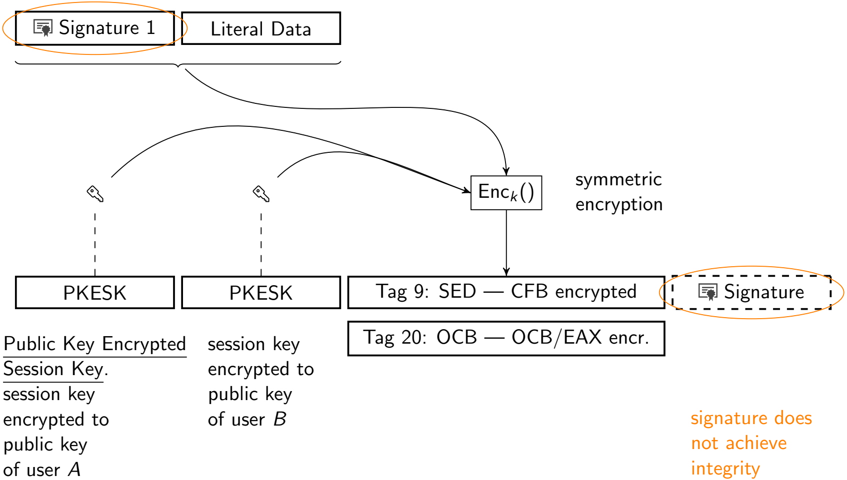
Decryption oracle attacks against modern cipher modes

MTG

- Classical oracle attacks
  - ◎ target ciphertext: CFB
  - 🔓 ciphertext decrypted by the oracle: CFB
- Oracle attacks using downgrades
  - ◎ target ciphertext: AEAD or AES Key Wrap
  - 🔓 ciphertext decrypted by the oracle: CFB (or CBC)


Signature 1
Literal Data
Enc$_k$()
symmetric encryption
PKESK
PKESK
Tag 9: SED — CFB encrypted
Tag 20: OCB — OCB/EAX encr.
Public Key Encrypted Session Key.
session key encrypted to public key of user *A*
session key encrypted to public key of user *B*


Signature 1
Literal Data
Enc$_k$()
symmetric
encryption
PKESK
PKESK
Tag 9: SED — CFB encrypted
Tag 20: OCB — OCB/EAX encr.
Public Key Encrypted
Session Key.
session key
encrypted to
public key
of user A
session key
encrypted to
public key
of user B
algorithm ID
unprotected
in CMS and
LibrePGP



Signature 1
Literal Data
Enc$_k$()
symmetric encryption
PKESK
PKESK
Tag 9: SED — CFB encrypted
Signature
Tag 20: OCB — OCB/EAX encr.
Public Key Encrypted Session Key.
session key encrypted to public key of user *A*
session key encrypted to public key of user *B*
signature does not achieve integrity

Decryption Oracle Attacks against Cryptographic Message Syntax (CMS)

- CMS realizes two AES-based AEAD modes
  - AES-CCM
  - AES-GCM
- both perform encryption using CTR-mode
- Legacy encryption mode in CMS: CBC
- previous work: Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013, 2013. https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography/.

Decryption Oracle Attacks against Cryptographic Message Syntax (CMS)

- CMS realizes two AES-based AEAD modes
  - AES-CCM
  - AES-GCM
- both perform encryption using CTR-mode
- Legacy encryption mode in CMS: CBC
- previous work: Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013, 2013. https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography/.

Decryption Oracle Attacks against Cryptographic Message Syntax (CMS)

- CMS realizes two AES-based AEAD modes
  - AES-CCM
  - AES-GCM
- both perform encryption using CTR-mode
- Legacy encryption mode in CMS: CBC
- previous work: Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013, 2013. https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography/.

Decryption Oracle Attacks against Cryptographic Message Syntax (CMS)

MTG

- CMS realizes two AES-based AEAD modes
  - AES-CCM
  - AES-GCM
- both perform encryption using CTR-mode
- Legacy encryption mode in CMS: CBC
- previous work: Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013, 2013. https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography/.

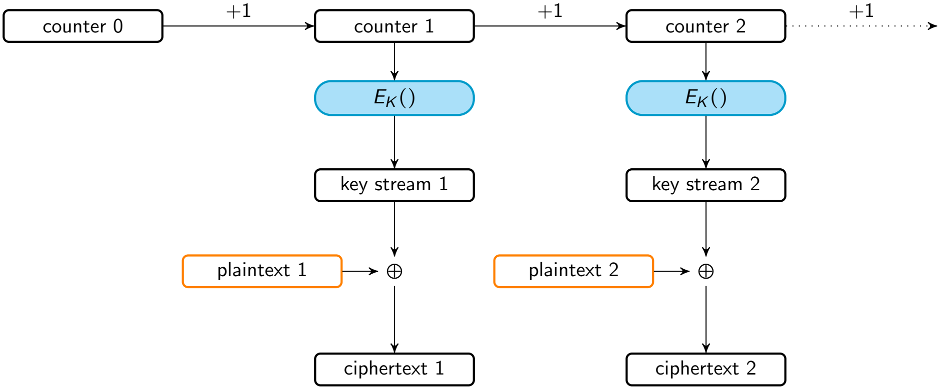
Decryption Oracle Attacks against Cryptographic Message Syntax (CMS)

- CMS realizes two AES-based AEAD modes
  - AES-CCM
  - AES-GCM
- both perform encryption using CTR-mode
- Legacy encryption mode in CMS: CBC
- previous work: Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013, 2013. https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography/.

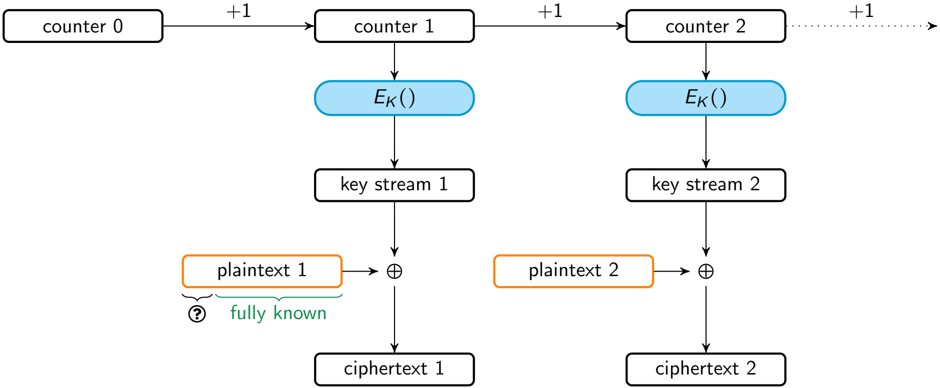
Decryption Oracle Attacks against Cryptographic Message Syntax (CMS)

- CMS realizes two AES-based AEAD modes
  - AES-CCM
  - AES-GCM
- both perform encryption using CTR-mode
- Legacy encryption mode in CMS: CBC
- previous work: Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013, 2013. https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography/.

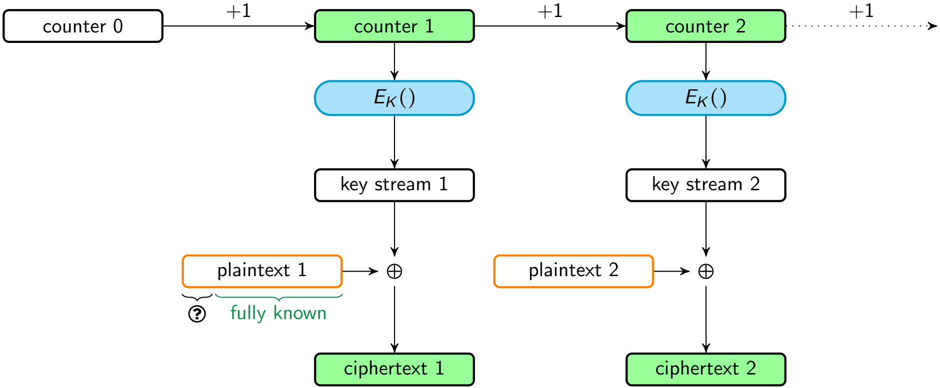
Revealing low entropy blocks in plaintext



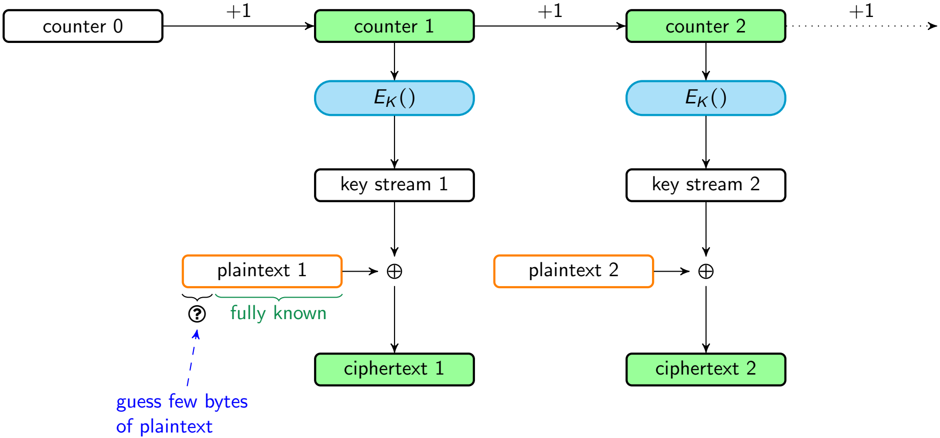
Revealing low entropy blocks in plaintext



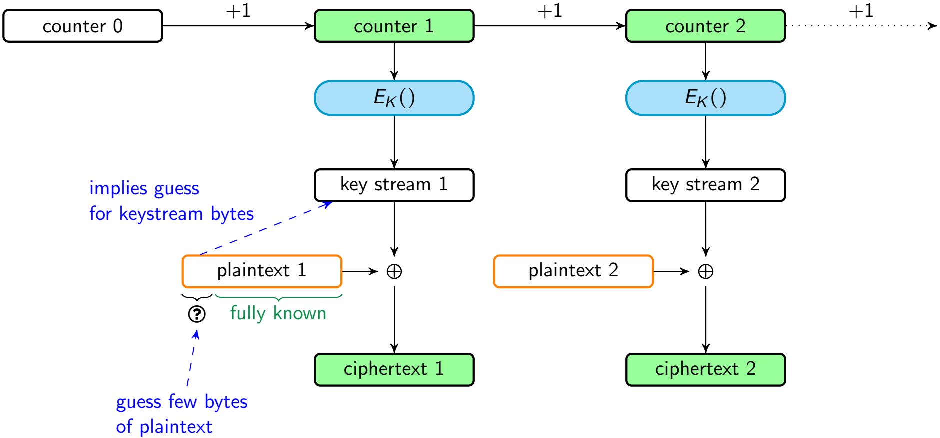
Revealing low entropy blocks in plaintext



Revealing low entropy blocks in plaintext

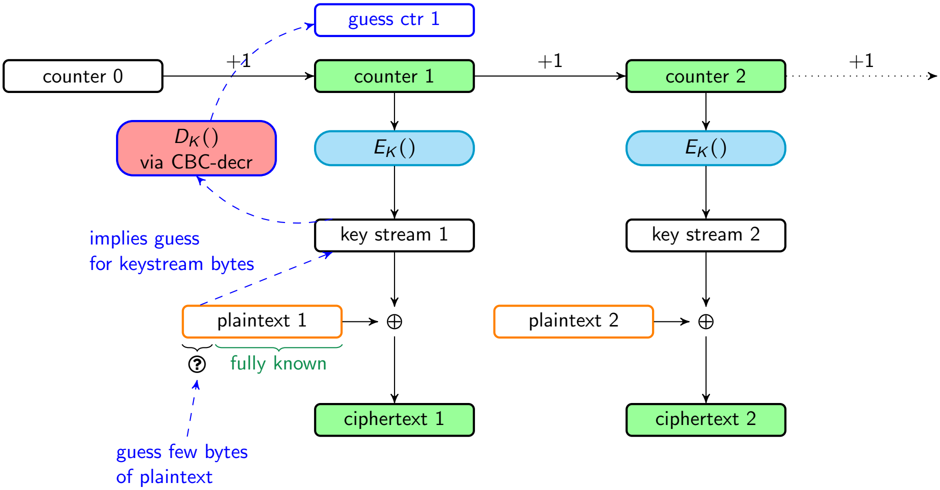


Revealing low entropy blocks in plaintext

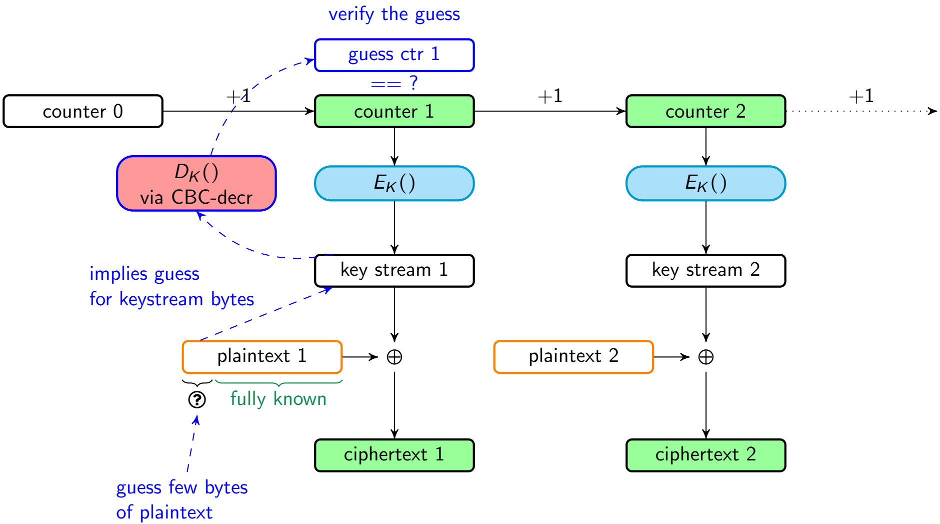


Revealing low entropy blocks in plaintext

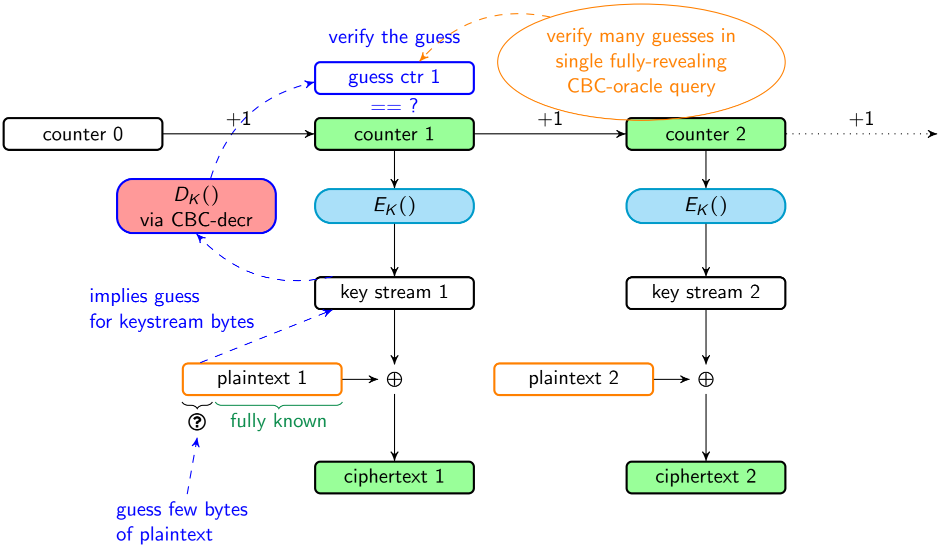
MTG



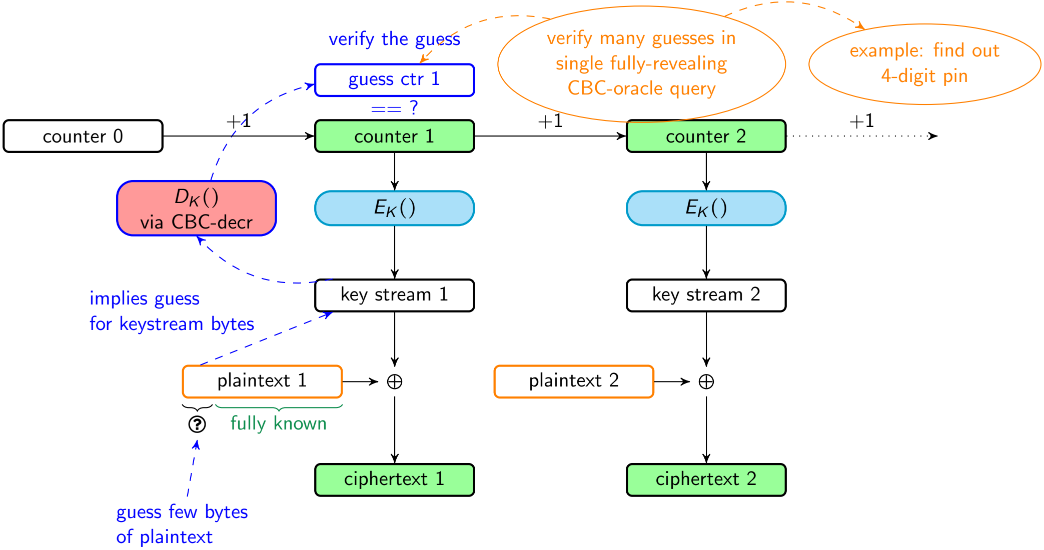
Revealing low entropy blocks in plaintext



Revealing low entropy blocks in plaintext



Revealing low entropy blocks in plaintext



CBC padding oracle

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - "playing" with the final bytes
  - padding error reveals information about value of final bytes

CBC padding oracle

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - "playing" with the final bytes
  - padding error reveals information about value of final bytes

CBC padding oracle

MTG

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - "playing" with the final bytes
  - padding error reveals information about value of final bytes

CBC padding oracle

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - "playing" with the final bytes
  - padding error reveals information about value of final bytes

CBC padding oracle

MTG

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - "playing" with the final bytes
  - padding error reveals information about value of final bytes

CBC padding oracle

MTG

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - "playing" with the final bytes
  - padding error reveals information about value of final bytes

CBC padding oracle

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - “playing” with the final bytes
  - padding error reveals information about value of final bytes

CBC padding oracle

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - "playing" with the final bytes
  - padding error reveals information about value of final bytes

CBC padding oracle

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - "playing" with the final bytes
  - padding error reveals information about value of final bytes

CBC padding oracle

- CBC operates only on full plaintext blocks
- padding needed
- padding in CMS:
  - *13 bytes of content* || 0x3 || 0x3 || 0x3
- padding oracle:
  - if decrypting party reveals whether padding was incorrect
- padding oracle attack
  - placing the target ciphertext block as final block
  - “playing” with the final bytes
  - padding error reveals information about value of final bytes

Countermeasure for CMS

MTG

- draft-ietf-lamps-cms-cek-hkdf-sha256
- defines new algorithm identifier for symmetric which indicates use of prior key HKDF derivation
- ```
  cea-CEKHKDFSHA256 CONTENT-ENCRYPTION ::= {
      IDENTIFIER id-alg-cek-hkdf-sha256
      PARAMS TYPE ContentEncryptionAlgorithmIdentifier ARE required
      SMIME-CAPS { IDENTIFIED BY id-alg-cek-hkdf-sha256 } }
  ```
  - PRK = HKDF-Extract(salt, IKM)
  - DEK = HKDF-Expand(PRK, AlgorithmID, OKM_SIZE)
- Cross-algorithm attack: derived key is different
  ```

Introduction

Decryption Oracle Attacks against Cryptographic Message Syntax

Plaintext manipulation attacks against LibrePGP AEAD

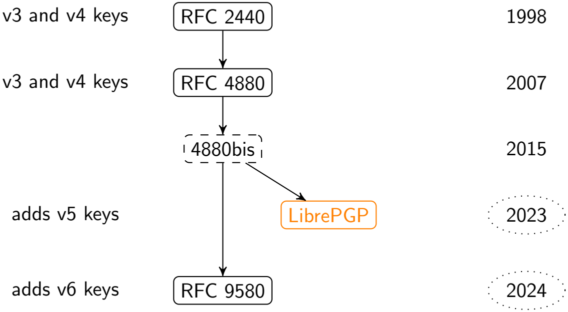
Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

Legacy Mode Downgrade Attacks against AES Key Wrap

Conclusion

Cross-Algorithm Attacks against LibrePGP AEAD

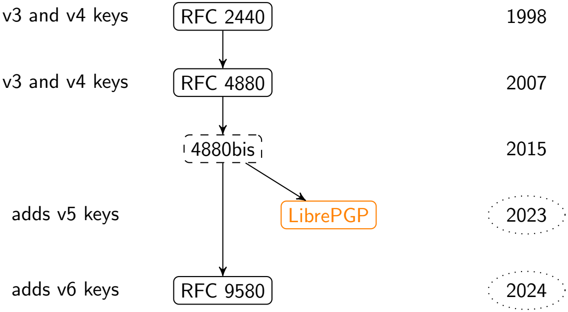
MTG



- LibrePGP AEAD:
  - GnuPG
  - RNP

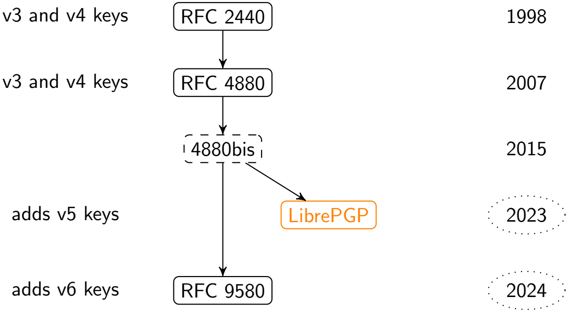
Cross-Algorithm Attacks against LibrePGP AEAD

MTG



- LibrePGP AEAD:
  - GnuPG
  - RNP

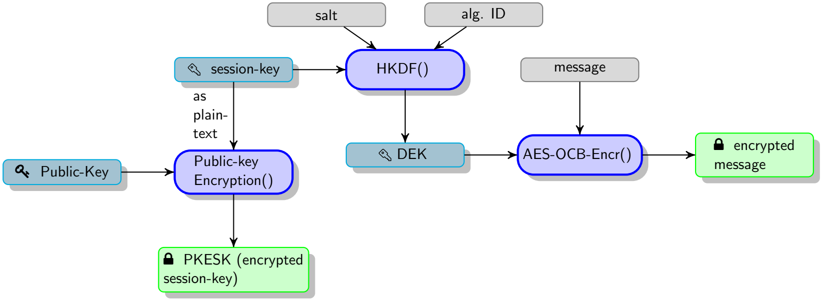
Cross-Algorithm Attacks against LibrePGP AEAD



- LibrePGP AEAD:
  - GnuPG
  - RNP

Key Derivation for AEAD in RFC 9580

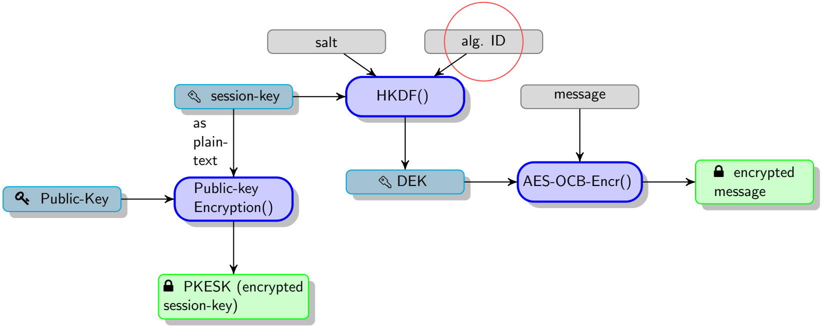
MTG



- LibrePGP: `session-key = DEK`

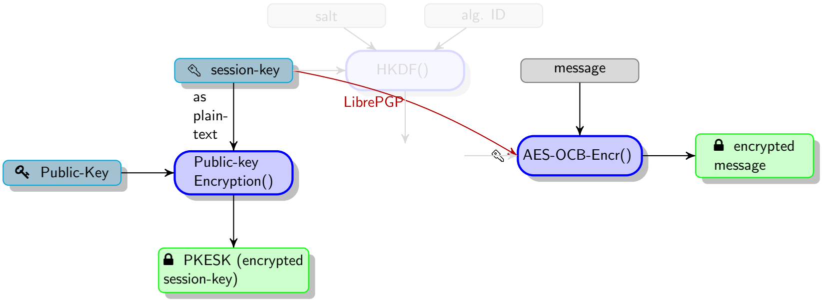
Key Derivation for AEAD in RFC 9580

MTG



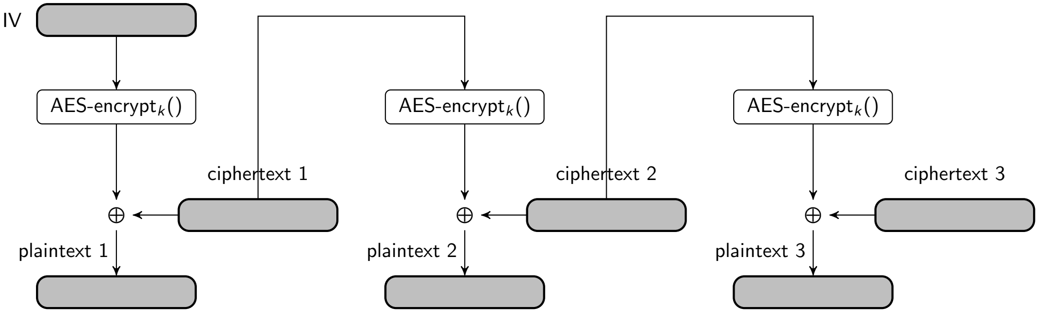
- LibrePGP: `session-key = DEK`

Key Derivation for AEAD in RFC 9580

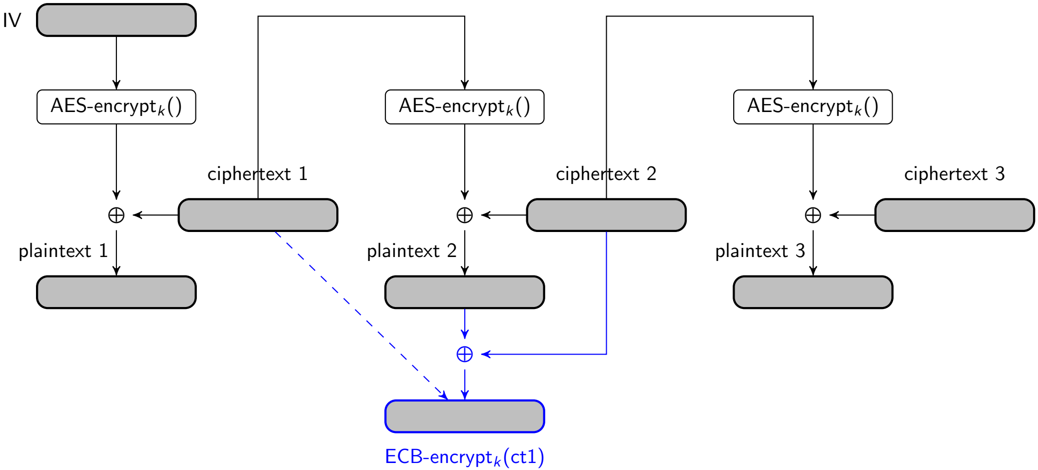


- LibrePGP: `session-key = DEK`

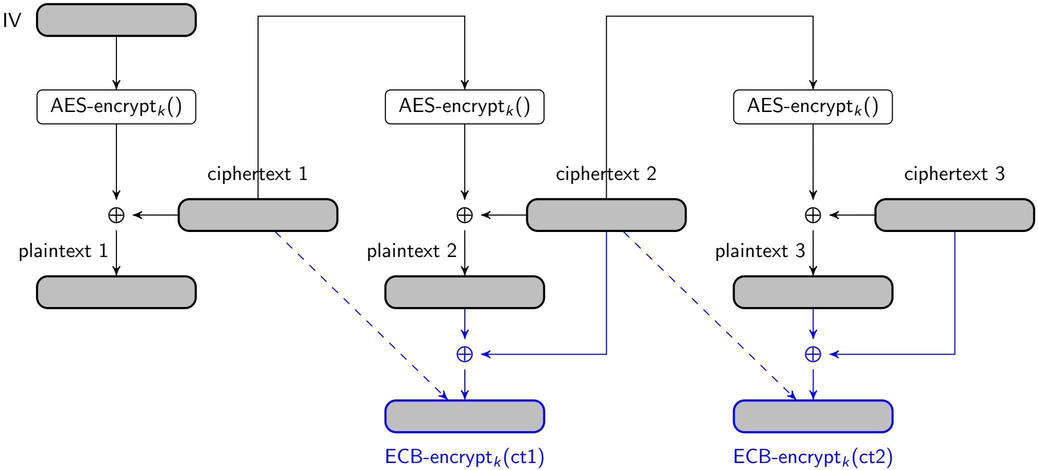
CFB decryption oracle as an ECB encryption oracle



CFB decryption oracle as an ECB encryption oracle



CFB decryption oracle as an ECB encryption oracle



What we can do with an OpenPGP SED decryption Oracle

- An SED decryption oracle can be used to decrypt SED packages (legacy attack)
- An SED decryption oracle provides an ECB encryption oracle
  - can we use an ECB encryption oracle to attack AEAD mode?
    - OpenPGP AEAD acc. to RFC 9580 [illegible]
    - but no key derivation in LibrePGP

What we can do with an OpenPGP SED decryption Oracle

- An SED decryption oracle can be used to decrypt SED packages (legacy attack)
- An SED decryption oracle provides an ECB encryption oracle
  - can we use an ECB encryption oracle to attack AEAD mode?
    - OpenPGP AEAD acc. to RFC 9580: ruled out by key derivation
    - but no key derivation in LibrePGP

What we can do with an OpenPGP SED decryption Oracle

- An SED decryption oracle can be used to decrypt SED packages (legacy attack)
- An SED decryption oracle provides an ECB encryption oracle
  - can we use an ECB encryption oracle to attack AEAD mode?
    - OpenPGP AEAD acc. to RFC 9580: ruled out by key derivation
    - but no key derivation in LibrePGP

What we can do with an OpenPGP SED decryption Oracle

MTG

- An SED decryption oracle can be used to decrypt SED packages (legacy attack)
- An SED decryption oracle provides an ECB encryption oracle
    - can we use an ECB encryption oracle to attack AEAD mode?
        - OpenPGP AEAD acc. to RFC 9580: ruled out by key derivation
        - but no key derivation in LibrePGP

What we can do with an OpenPGP SED decryption Oracle

- An SED decryption oracle can be used to decrypt SED packages (legacy attack)
- An SED decryption oracle provides an ECB encryption oracle
  - can we use an ECB encryption oracle to attack AEAD mode?
    - OpenPGP AEAD acc. to RFC 9580: ruled out by key derivation
    - but no key derivation in LibrePGP

LibrePGP AEAD Encryption

- "OCB Packet"
  - supported modes
    - OCB
    - EAX (deprecated)
  - "chunked AEAD"

LibrePGP AEAD Encryption

- "OCB Packet"
  - supported modes
    - OCB
    - EAX (deprecated)
  - "chunked AEAD"

LibrePGP AEAD Encryption

- "OCB Packet"
  - supported modes
    - OCB
    - EAX (deprecated)
  - "chunked AEAD"

LibrePGP AEAD Encryption

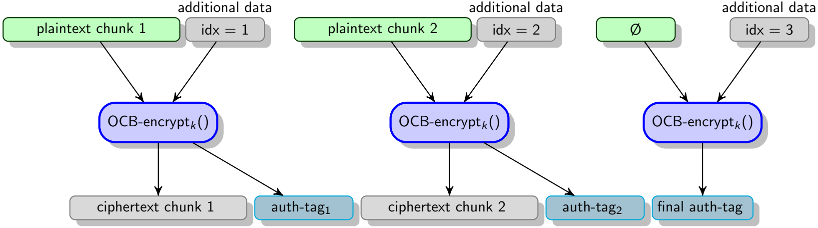
- "OCB Packet"
  - supported modes
    - OCB
    - EAX (deprecated)
  - "chunked AEAD"

LibrePGP AEAD Encryption

- “OCB Packet”
  - supported modes
    - OCB
    - EAX (deprecated)
  - “chunked AEAD”

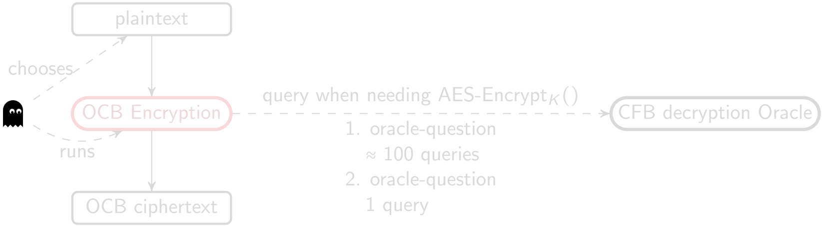
LibrePGP chunked AEAD

MTG



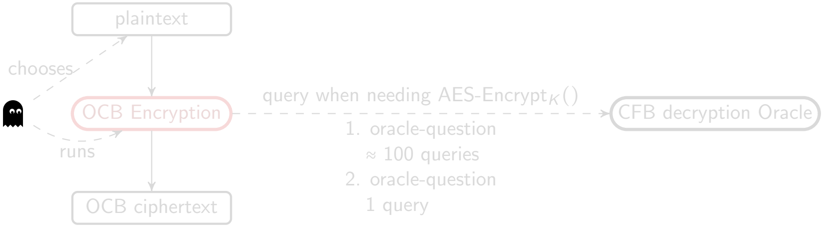
OCB Encryption

- OCB encryption uses only block cipher (e.g. AES) block-encryption
- CFB decryption uses only block-encryption
- Thus: use CFB-decryption as an oracle!
- ϟ Insert data into an existing LibrePGP OCB Packet



OCB Encryption

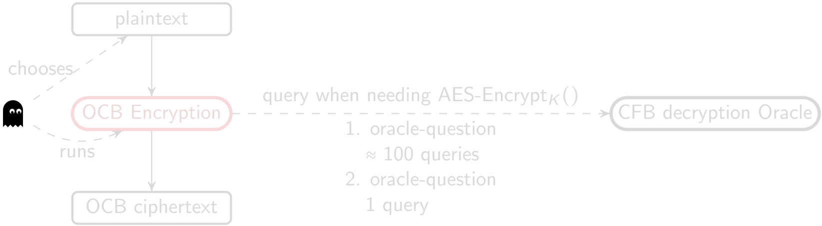
- OCB encryption uses only block cipher (e.g. AES) block-encryption
- CFB decryption uses only block-encryption
- Thus: use CFB-decryption as an oracle!
- ϟ Insert data into an existing LibrePGP OCB Packet



OCB Encryption

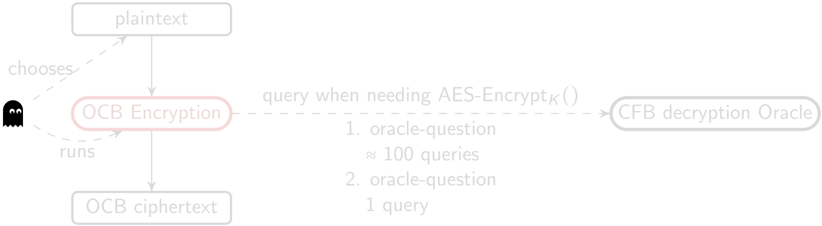
MTG

- OCB encryption uses only block cipher (e.g. AES) block-encryption
- CFB decryption uses only block-encryption
- Thus: use CFB-decryption as an oracle!
- ϟ Insert data into an existing LibrePGP OCB Packet



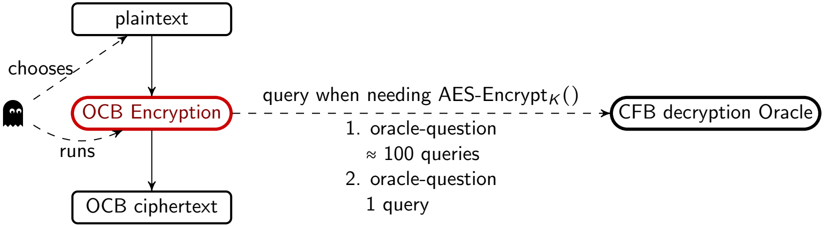
OCB Encryption

- OCB encryption uses only block cipher (e.g. AES) block-encryption
- CFB decryption uses only block-encryption
- Thus: use CFB-decryption as an oracle!
- ϟ Insert data into an existing LibrePGP OCB Packet



OCB Encryption

- OCB encryption uses only block cipher (e.g. AES) block-encryption
- CFB decryption uses only block-encryption
- Thus: use CFB-decryption as an oracle!
- ϟ Insert data into an existing LibrePGP OCB Packet



procedure OCB-ENCRYPT($k \in \{0,1\}^{\mathrm{keylen}}, N \in \{0,1\}^{120}, A \in \{0,1\}^*, P \in \{0,1\}^*$)

  $\tilde{m} = \lfloor |P|/128 \rfloor$

  parse P as $P_1 \| P_2 \| \dots \| P_{\tilde{m}} \| P_*$ where $|P_i| = 128$ for each $1 \le i \le \tilde{m}$ and $0 \le |P_*| < 128$

  compute values L_*, $L_\$$, and L_i for $0 \le i$ (dbl. in GF(2^{128}) using $E_k([0]^{128})$)

  ...

  $f = E_k(\mathcal{N}[1:122] \| [0]^6)$ // "Ktop"

  ...

  $G_0 = \dots$ // inital mask

  $s_0 = [0]^{128}$ // "Checksum"

Setup $L_{\dots}$ and G_0 values

  for $1 \le i \le \tilde{m}$ **do**

    $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$

    $C_i = G_i \oplus E_k(P_i \oplus G_i)$

    $s_i = s_{i-1} \oplus P_i$

Encryption loop with input and output whitening

  end for

  if $|P_*| > 0$ **then**

    $\tilde{n} \leftarrow \tilde{m} + 1$

    ...

    $u = E_k(G_{\tilde{n}})$ // "Pad"

    ...

Special case of non-full final plaintext block

  else

    $\tilde{n} \leftarrow \tilde{m}$

  end if

  $T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$ ← Compute the auth. tag

  return $C = C_1 \| C_2 \| \dots \| C_{\tilde{n}} \| T[1:\mathrm{taglen}]$

end procedure

procedure OCB-ENCRYPT($k \in \{0,1\}^{\text{keylen}}, N \in \{0,1\}^{120}, A \in \{0,1\}^*, P \in \{0,1\}^*$)

  $\tilde{m} = \lfloor |P|/128 \rfloor$

  parse P as $P_1 \| P_2 \| \ldots \| P_{\tilde{m}} \| P_*$ where $|P_i| = 128$ for each $1 \le i \le \tilde{m}$ and $0 \le |P_*| < 128$

  compute values L_*, $L_\$$, and L_i for $0 \le i$ (dbl. in GF(2^{128})) using $E_k([0]^{128})$

  ...

  $f = E_k(\mathcal{N}[1:122] \| [0]^6)$ // "Ktop"

  ...

  $G_0 = \ldots$ // inital mask

  $s_0 = [0]^{128}$ // "Checksum"

  for $1 \le i \le \tilde{m}$ **do**

    $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$

    $C_i = G_i \oplus E_k(P_i \oplus G_i)$

    $s_i = s_{i-1} \oplus P_i$

  end for

  if $|P_*| > 0$ **then**

    $\tilde{n} \leftarrow \tilde{m} + 1$

    ...

    $u = E_k(G_{\tilde{n}})$ // "Pad"

    ...

  else

    $\tilde{n} \leftarrow \tilde{m}$

  end if

  $T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$

  return $C = C_1 \| C_2 \| \ldots \| C_{\tilde{n}} \| T[1:\mathrm{taglen}]$

end procedure

Setup $L_{\ldots}$ and G_0 values

Encryption loop with input and output whitening

1st question to ECB oracle

Special case of non-full final plaintext block

Compute the auth. tag

procedure OCB-ENCRYPT($k \in \{0,1\}^{\mathrm{keylen}}, N \in \{0,1\}^{120}, A \in \{0,1\}^*, P \in \{0,1\}^*$)

  $\tilde{m} = \lfloor |P|/128 \rfloor$

  parse P as $P_1 \parallel P_2 \parallel \ldots \parallel P_{\tilde{m}} \parallel P_*$ where $|P_i| = 128$ for each $1 \leq i \leq \tilde{m}$ and $0 \leq |P_*| < 128$

  compute values L_*, $L_\$$, and L_i for $0 \leq i$ (dbl. in GF(2^{128}) using $E_k([0]^{128})$)

  ...

  $f = E_k(\mathcal{N}[1:122] \parallel [0]^6)$ // "Ktop"

  ...

  $G_0 = \ldots$ // inital mask

  $s_0 = [0]^{128}$ // "Checksum"

  for $1 \leq i \leq \tilde{m}$ **do**

    $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$

    $C_i = G_i \oplus E_k(P_i \oplus G_i)$

    $s_i = s_{i-1} \oplus P_i$

  end for

  if $|P_*| > 0$ **then**

    $\tilde{n} \leftarrow \tilde{m} + 1$

    ...

    $u = E_k(G_{\tilde{n}})$ // "Pad"

    ...

  else

    $\tilde{n} \leftarrow \tilde{m}$

  end if

  $T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$

  return $C = C_1 \parallel C_2 \parallel \ldots \parallel C_{\tilde{n}} \parallel T[1:\mathrm{taglen}]$

end procedure

Setup $L_{\ldots}$ and G_0 values

Encryption loop with input and output whitening

Special case of non-full final plaintext block

Compute the auth. tag

1st question to ECB oracle

2nd question to ECB oracle

2nd encr. call only

OCB Hash

procedure $\text{OCB-Hash}(\text{key } k \in \{0,1\}^{|\mathrm{K}|}, \text{ additional data } A \in \{0,1\}^*)$
    $L_* = E_k([0]^{128})$
    $L_\$ = \mathrm{ocbDouble}(L_*)$
    $L_0 = \mathrm{ocbDouble}(L_\$)$
    $L_i = \mathrm{ocbDouble}(L_{i-1})$ for any integer $i > 0$
    $m = \lfloor |A|/128 \rfloor$
    parse A as $A_1 \parallel A_2 \parallel \ldots \parallel A_m \parallel A_*$ where $|A_i| = 128$ for each $1 \leq i \leq m$ and $0 \leq |A_*| < 128$
    $F_0 = [0]^{128}$ // Offset
    for i ← 1 to m **do**
        $F_i = F_{i-1} \oplus L_{\mathrm{ntz}(i)}$
    end for
    if $|A_*| > 0$ **then**
        $n \leftarrow m + 1$
        $F_n = F_m \oplus L_*$
        $A_n = (A_* \parallel 1 \parallel [0]^{127-|A_*|})$
    else
        $n \leftarrow m$
    end if
    $S_0 = [0]^{128}$ // Sum
    for i ← 1 to n **do**
        $S_i = S_{i-1} \oplus E_k(A_i \oplus F_i)$
    end for
    return $S = S_n$
end procedure

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD
- OCB encryption
  - uses only block cipher encryption
  - $\rightarrow$ OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD
- OCB encryption
  - uses only block cipher encryption
  - $\rightarrow$ OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD
- OCB encryption
  - uses only block cipher encryption
  - → OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD
- OCB encryption
  - uses only block cipher encryption
  - →
    OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD
- OCB encryption
  - uses only block cipher encryption
  - $\rightarrow$ OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD
- OCB encryption
  - uses only block cipher encryption
  - → OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD
- OCB encryption
  - uses only block cipher encryption
  - $\rightarrow$ OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD
- OCB encryption
  - uses only block cipher encryption
  - → OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD ✉ ✉ ✉ …
- OCB encryption
  - uses only block cipher encryption
  - → OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD
- OCB encryption
  - uses only block cipher encryption
  - → OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

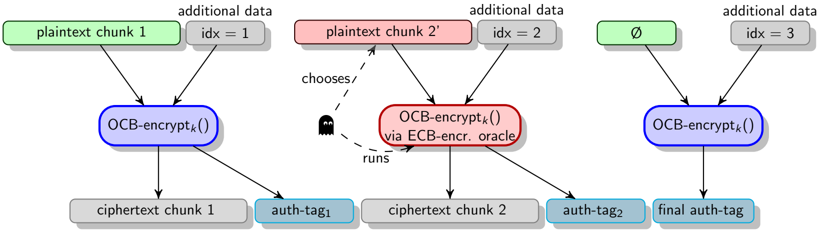
MTG

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD ✉ ✉ ✉ ...
- OCB encryption
  - uses only block cipher encryption
  - → OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Summary: LibrePGP OCB ciphertext manipulation

- For now, consider only the cryptographic layer
- Legacy SED Packets
  - implement CFB encryption
  - assuming a CFB decryption oracle
  - realizes an ECB encryption oracle
- LibrePGP OCB Packets
  - uses chunked AEAD ✉ ✉ ✉ ⋯
- OCB encryption
  - uses only block cipher encryption
  - → OCB encryption under unknown key is possible with access to ECB encryption oracle
- Attack
  - exchange or append chunk to existing ciphertext with unknown plaintext

Insertion of LibrePGP AEAD chunk



It's not that simple: expected plaintext structure in SED (CFB) oracle

- So far: cryptographic attack
- Not accounting for
  - availability of SED decryption
  - OpenPGP plaintext format

It's not that simple: expected plaintext structure in SED (CFB) oracle

MTG

- So far: cryptographic attack
- Not accounting for
  - availability of SED decryption
  - OpenPGP plaintext format

It's not that simple: expected plaintext structure in SED (CFB) oracle

- So far: cryptographic attack
- Not accounting for
  - availability of SED decryption
  - OpenPGP plaintext format

It's not that simple: expected plaintext structure in SED (CFB) oracle

- So far: cryptographic attack
- Not accounting for
  - availability of SED decryption
  - OpenPGP plaintext format

Availability and exploitability of SED decryption

MTG

- GnuPG (CLI) in default configuration
  - outputs the SED plaintext
  - non-zero exit code and warning (stderror)

```
gpg: WARNING: message was not integrity protected
gpg: decryption forced to fail!
```

- RNP
  - supports SED unrestricted
  - implements quick-check
    - omitted detail: OpenPGP SED CFB encryption uses two-step CFB encryption
    - Quick-check (redundancy test): requires the equality of two 2-byte pairs at the start of the plaintext
    - random ciphertext fails this check with 2^{-16}
    - Quick check is vulnerability in itself!

Two-step CFB encryption in SED Packet

procedure $\mathrm{SED\text{-}D\textsc{ec}}_K(H \parallel B_1 \parallel \ldots \parallel B_m)$ with $H \in \{0,1\}^{144}$ and $B_i \in \{0,1\}^{128}$

  $Y \leftarrow \mathrm{CFB\text{-}\textsc{decrypt}}_K([0]^{128}, H)$ // $Y \in \{0,1\}^{128+16}$

  if have quick-check AND $Y[96:127] \neq Y[128:143]$ **then**

    Abort with error

  end if

  $\mathrm{IV} \leftarrow H[16:143]$

  return $\mathrm{CFB\text{-}\textsc{decrypt}}_K(\mathrm{IV}, B_1 \parallel \ldots \parallel B_m)$

end procedure

OpenPGP plaintext format

MTG

- decrypted plaintext must be either
  - Literal Data (LIT) Packet
  - (Compressed Data)
  - (Signed Data)
  - (Encrypted Data)

OpenPGP plaintext format

- decrypted plaintext must be either
  - Literal Data (LIT) Packet
  - (Compressed Data)
  - (Signed Data)
  - (Encrypted Data)

OpenPGP plaintext format

- decrypted plaintext must be either
  - Literal Data (LIT) Packet
  - (Compressed Data)
  - (Signed Data)
  - (Encrypted Data)

OpenPGP plaintext format

- decrypted plaintext must be either
  - Literal Data (LIT) Packet
  - (Compressed Data)
  - (Signed Data)
  - (Encrypted Data)

OpenPGP plaintext format

- decrypted plaintext must be either
  - Literal Data (LIT) Packet
  - (Compressed Data)
  - (Signed Data)
  - (Encrypted Data)

Random appearance of LIT Packet in SED (CFB) oracle

- Attacker's ciphertext will decrypt to plaintexts with random appearance
- Plaintext must by chance be decodable as LIT Packet

- Problems:
  - Need to vary the leading part until a plaintext is returned
  - How to know the offset of returned plaintext into "message"?

Random appearance of LIT Packet in SED (CFB) oracle

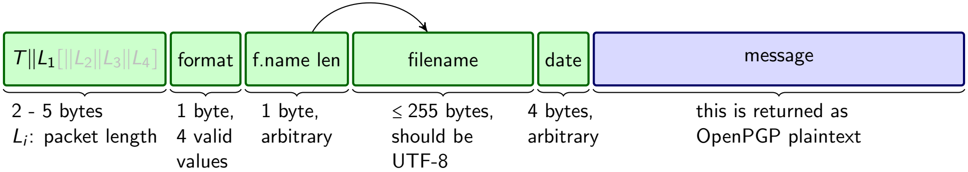
MTG

- Attacker's ciphertext will decrypt to plaintexts with random appearance
- Plaintext must by chance be decodable as LIT Packet

- Problems:
  - Need to vary the leading part until a plaintext is returned
  - How to know the offset of returned plaintext into "message"?

Random appearance of LIT Packet in SED (CFB) oracle

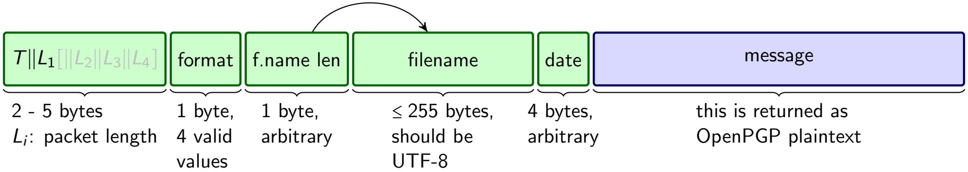
- Attacker's ciphertext will decrypt to plaintexts with random appearance
- Plaintext must by chance be decodable as LIT Packet



- Problems:
  - Need to vary the leading part until a plaintext is returned
  - How to know the offset of returned plaintext into "message"?

Random appearance of LIT Packet in SED (CFB) oracle

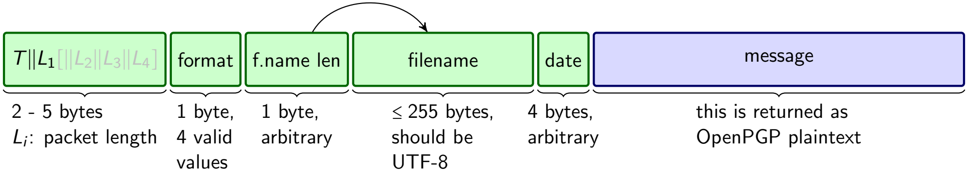
- Attacker's ciphertext will decrypt to plaintexts with random appearance
- Plaintext must by chance be decodable as LIT Packet



- Problems:
  - Need to vary the leading part until a plaintext is returned
  - How to know the offset of returned plaintext into "message"?

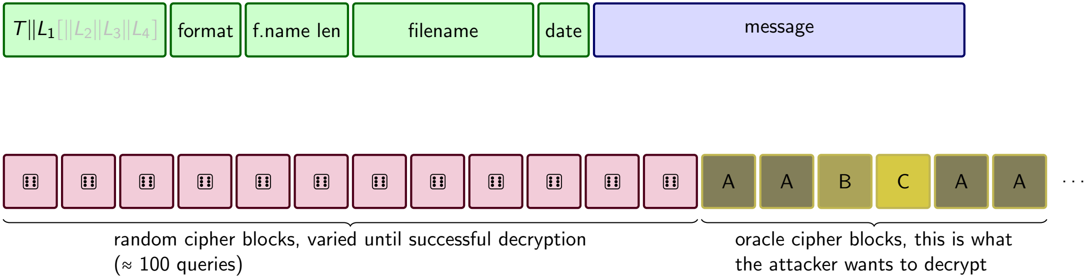
Random appearance of LIT Packet in SED (CFB) oracle

- Attacker's ciphertext will decrypt to plaintexts with random appearance
- Plaintext must by chance be decodable as LIT Packet

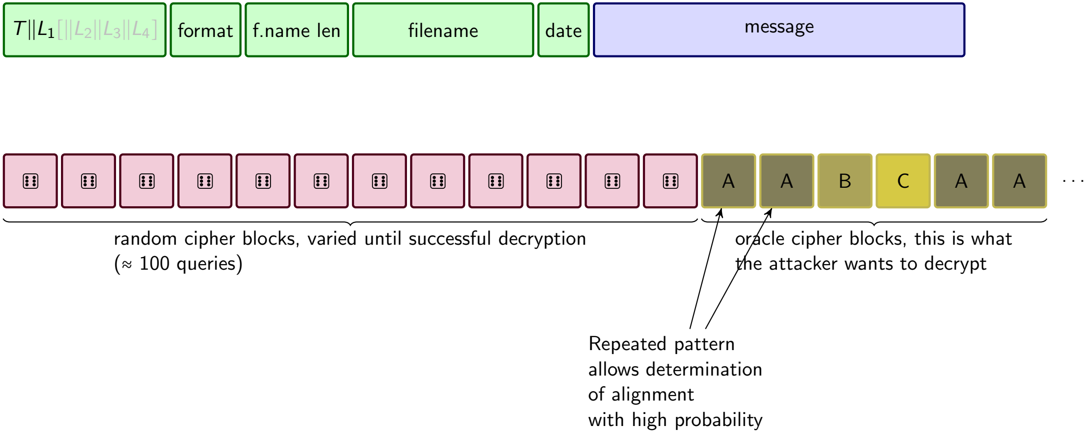


- Problems:
  - Need to vary the leading part until a plaintext is returned
  - How to know the offset of returned plaintext into "message"?

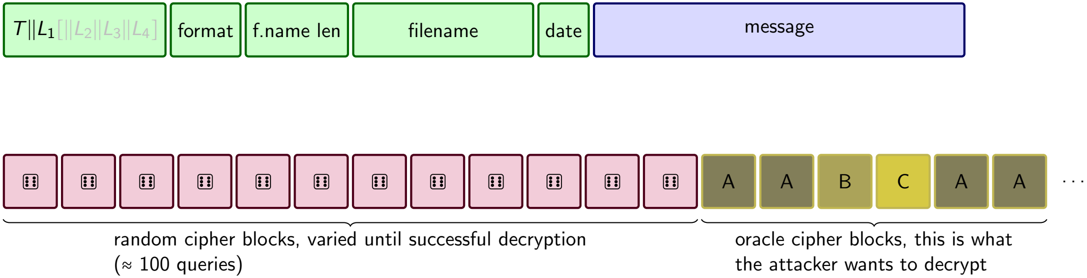
Crafting Ciphertexts for the initial oracle question



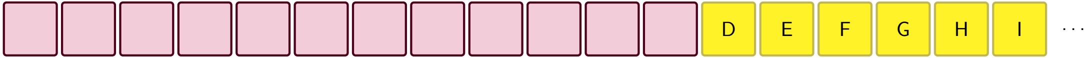
Crafting Ciphertexts for the initial oracle question

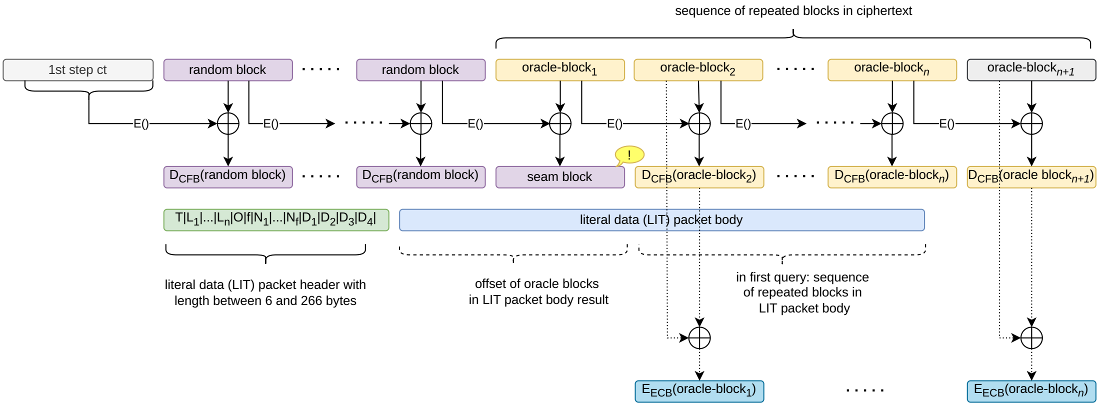


Crafting Ciphertexts for the initial oracle question



2n question: single query reusing the leading "random" blocks (offset now known):





sequence of repeated blocks in ciphertext
1st step ct
random block
random block
oracle-block$_1$
oracle-block$_2$
oracle-block$_n$
oracle-block$_{n+1}$
E()
E()
E()
E()
E()
E()
D_{CFB}(random block)
D_{CFB}(random block)
seam block
!
D_{CFB}(oracle-block$_2$)
D_{CFB}(oracle-block$_n$)
D_{CFB}(oracle block$_{n+1}$)
T|L$_1$|...|L$_n$|O|f|N$_1$|...|N$_f$|D$_1$|D$_2$|D$_3$|D$_4$|
literal data (LIT) packet body
literal data (LIT) packet header with length between 6 and 266 bytes
offset of oracle blocks in LIT packet body result
in first query: sequence of repeated blocks in LIT packet body
E_{ECB}(oracle-block$_1$)
E_{ECB}(oracle-block$_n$)

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for OCB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries ($\approx$ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries (≈ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Summary of LibrePGP AEAD OCB Chunk Encryption Attack

MTG

- Exploit legacy SED Packet decryption to encrypt OCB chunk under same key
- Challenge: valid LIT packet must appear in decrypted data
- Attack execution:
  - 1st oracle-question:
    - vary leading part of the ciphertext, place blocks for ECB encryption at end
    - many queries ($\approx$ 100)
  - 2nd oracle-question:
    - reuse the leading part that was successful
    - place new blocks for encryption query at the end
    - single query
- High number of queries for 1st question:
  - possibly compensate by multi-user attack
  - victim sees only 2 queries which it has to answer
- Successfully implemented (C++) against GnuPG

Introduction

Decryption Oracle Attacks against Cryptographic Message Syntax

Plaintext manipulation attacks against LibrePGP AEAD

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

Legacy Mode Downgrade Attacks against AES Key Wrap

Conclusion

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \| [0]^6)$ // 1st oracle-question
  - $G_0 = \ldots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_{\$}) \oplus \mathrm{HASH}(K, A)$~~ // not needed

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \,\|\, [0]^6)$ // 1st oracle-question
  - $G_0 = \ldots$
- For $1 \le i \le$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \| [0]^6)$ // 1st oracle-question
  - $G_0 = \ldots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

MTG

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \| [0]^6)$ // 1st oracle-question
  - $G_0 = \ldots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \,\|\, [0]^6)$ // 1st oracle-question
  - $G_0 = \ldots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_i
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

MTG

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \parallel [0]^6)$ // 1st oracle-question
  - $G_0 = \ldots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \| [0]^6)$ // 1st oracle-question
  - $G_0 = \dots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

MTG

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \,\|\, [0]^6)$ // 1st oracle-question
  - $G_0 = \ldots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

MTG

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \parallel [0]^6)$ // 1st oracle-question
  - $G_0 = \dots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

compute with CFB-oracle

G_i known through fist oracle-question. Assume low entropy plaintext block P_t. Guess for P_t implies guess for C_t.

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

MTG

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \,\|\, [0]^6)$ // 1st oracle-question
  - $G_0 = \ldots$
- For $1 \le i \le$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

compute with CFB-oracle

G_i known through fist oracle-question. Assume low entropy plaintext block P_t. Guess for P_t implies guess for C_t.

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

MTG

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \| [0]^6)$ // 1st oracle-question
  - $G_0 = \dots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

compute with CFB-oracle

G_i known through fist oracle-question. Assume low entropy plaintext block P_t. Guess for P_t implies guess for C_t.

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

MTG

OCB encryption

- ...
- compute mask values:
  - $E_k([0]^{128})$ // 1st oracle-question
  - $f = E_k(\mathcal{N}[1:122] \parallel [0]^6)$ // 1st oracle-question
  - $G_0 = \ldots$
- For $1 \leq i \leq$ block_count
  - $G_i = G_{i-1} \oplus L_{\mathrm{ntz}(i)}$ // (1st oracle-question)
  - $C_i = G_i \oplus E_k(P_i \oplus G_i)$ // 2nd oracle-question: Guesses for P_t
  - $s_i = s_{i-1} \oplus P_i$
- ...
- ~~$T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \mathrm{HASH}(K, A)$~~ // not needed

compute with CFB-oracle

G_i known through fist oracle-question. Assume low entropy plaintext block P_t. Guess for P_t implies guess for C_t.

Introduction

Decryption Oracle Attacks against Cryptographic Message Syntax

Plaintext manipulation attacks against LibrePGP AEAD

Plaintext recovery for low entropy blocks in LibrePGP OCB Packets

Legacy Mode Downgrade Attacks against AES Key Wrap

Conclusion

Legacy Mode Downgrade Attacks against AES Key Wrap

- AES Key Wrap in NIST SP 800-38F, RFC 3394
- Key encryption with 64-bit “integrity check register”
- AES Key Wrap decryption / unwrap uses only $D_k()$
- CMS defines AES Key Wrap
  - Legacy encryption mode: CBC
  - CBC decryption uses block $D_k()$

    is oracle for AES Key Wrap decryption
- was observed already by Jager et al.

Legacy Mode Downgrade Attacks against AES Key Wrap

- AES Key Wrap in NIST SP 800-38F, RFC 3394
- Key encryption with 64-bit "integrity check register"
- AES Key Wrap decryption / unwrap uses only $D_k()$
- CMS defines AES Key Wrap
  - Legacy encryption mode: CBC
  - CBC decryption uses block $D_k()$

    $\Rightarrow$ is oracle for AES Key Wrap decryption
- was observed already by Jager et al.

Legacy Mode Downgrade Attacks against AES Key Wrap

- AES Key Wrap in NIST SP 800-38F, RFC 3394
- Key encryption with 64-bit "integrity check register"
- AES Key Wrap decryption / unwrap uses only $D_k()$
- CMS defines AES Key Wrap
  - Legacy encryption mode: CBC
  - CBC decryption uses block $D_k()$

    is oracle for AES Key Wrap decryption
- was observed already by Jager et al.

Legacy Mode Downgrade Attacks against AES Key Wrap

MTG

- AES Key Wrap in NIST SP 800-38F, RFC 3394
- Key encryption with 64-bit "integrity check register"
- AES Key Wrap decryption / unwrap uses only $D_k()$
- CMS defines AES Key Wrap
  - Legacy encryption mode: CBC
  - CBC decryption uses block $D_k()$
    - is oracle for AES Key Wrap decryption
- was observed already by Jager et al.

Legacy Mode Downgrade Attacks against AES Key Wrap

MTG

- AES Key Wrap in NIST SP 800-38F, RFC 3394
- Key encryption with 64-bit "integrity check register"
- AES Key Wrap decryption / unwrap uses only $D_k()$
- CMS defines AES Key Wrap
  - Legacy encryption mode: CBC
  - CBC decryption uses block $D_k()$
    - is oracle for AES Key Wrap decryption
- was observed already by Jager et al.

Legacy Mode Downgrade Attacks against AES Key Wrap

MTG

- AES Key Wrap in NIST SP 800-38F, RFC 3394
- Key encryption with 64-bit "integrity check register"
- AES Key Wrap decryption / unwrap uses only $D_k()$
- CMS defines AES Key Wrap
  - Legacy encryption mode: CBC
  - CBC decryption uses block $D_k()$
    - is oracle for AES Key Wrap decryption
- was observed already by Jager et al.

Legacy Mode Downgrade Attacks against AES Key Wrap

- AES Key Wrap in NIST SP 800-38F, RFC 3394
- Key encryption with 64-bit "integrity check register"
- AES Key Wrap decryption / unwrap uses only $D_k()$
- CMS defines AES Key Wrap
  - Legacy encryption mode: CBC
  - CBC decryption uses block $D_k()$
    - is oracle for AES Key Wrap decryption
- was observed already by Jager et al.

Legacy Mode Downgrade Attacks against AES Key Wrap

MTG

- AES Key Wrap in NIST SP 800-38F, RFC 3394
- Key encryption with 64-bit "integrity check register"
- AES Key Wrap decryption / unwrap uses only $D_k()$
- CMS defines AES Key Wrap
  - Legacy encryption mode: CBC
  - CBC decryption uses block $D_k()$
    - is oracle for AES Key Wrap decryption
- was observed already by Jager et al.

```
Inputs:  Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}, and
           Key, K (the KEK).
Outputs: Plaintext, n 64-bit values {P1, P2, ..., Pn}.

1) Initialize variables.
   Set A[s] = C[0] where s = 6n
   For i = 1 to n
       R[s][i] = C[i]
c2) Calculate the intermediate values.
   For t = s to 1
       A[t-1] = MSB(64, AES-1(K, ((A[t] ^ t) | R[t][n]))
       R[t-1][1] = LSB(64, AES-1(K, ((A[t]^t) | R[t][n]))
       For i = 2 to n
           R[t-1][i] = R[t][i-1]
3) Output the results.
   If A[0] is an appropriate initial value (see 2.2.3),
   Then
       For i = 1 to n
           P[i] = R[0][i]
   Else
       Return an error
```

Number of oracle-questions

- assume 128-bit key:
  - $n = 2$
  - loop iterations: $s = 6 \times 2 = 12$
  - 12 oracle queries
- full plaintext (wrapped key) recovery
- CBC padding oracle might be leveraged

Number of oracle-questions

- assume 128-bit key:
  - $n = 2$
  - loop iterations: $s = 6 \times 2 = 12$
  - 12 oracle queries
- full plaintext (wrapped key) recovery
- CBC padding oracle might be leveraged

Number of oracle-questions

MTG

- assume 128-bit key:
  - $n = 2$
  - loop iterations: $s = 6 \times 2 = 12$
  - 12 oracle queries
- full plaintext (wrapped key) recovery
- CBC padding oracle might be leveraged

Number of oracle-questions

- assume 128-bit key:
  - $n = 2$
  - loop iterations: $s = 6 \times 2 = 12$
  - 12 oracle queries
- full plaintext (wrapped key) recovery
- CBC padding oracle might be leveraged

Number of oracle-questions

- assume 128-bit key:
  - $n = 2$
  - loop iterations: $s = 6 \times 2 = 12$
  - 12 oracle queries
- full plaintext (wrapped key) recovery
- CBC padding oracle might be leveraged

Number of oracle-questions

- assume 128-bit key:
  - $n = 2$
  - loop iterations: $s = 6 \times 2 = 12$
  - 12 oracle queries
- full plaintext (wrapped key) recovery
- CBC padding oracle might be leveraged

MTG

Summary

	Attacked AEAD mode & direction	oracle type	Exploited legacy mode	Nb questions	Nb queries
CMS	AES-CCM, AES-GCM decr (low entropy block)	inverse	CBC	1	1
	AES Key Wrap decr.	direct		≥ 12	≥ 12
LibrePGP	OCB encr	direct	CFB	2	≈ 100
	OCB decr (low entropy block)	inverse			

Conclusion

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

MTG

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

MTG

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

MTG

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

MTG

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

MTG

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

MTG

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

MTG

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Conclusion

- Cryptographic design errors in CMS and LibrePGP
  - email probably not affected (efail countermeasures should prevent this)
  - Thunderbird seems not affected according to test
- Countermeasure
  - note: signatures don't protect the integrity of a message
  - CMS: yes, new RFC (key derivation)
    - new mechanism, must be implemented and used
  - LibrePGP:
    - disable SED decryption
    - attack still possible against GnuPG
    - no update of the spec
  - OpenPGP (RFC 9580):
    - hard-wired key derivation

Thank you for your attention

Dr. Falko Strenzke
falko.strenzke@mtg.de
+49 6151 8000-24

MTG AG
www.mtg.de

AEAD in RFC 9580

```
           +-----------------------+    +--------------+
           | decrypted session key|    | salt value  |
           +-----------------------+    +-----+--------+       +------------------------+
                    |                         |                |  packet tag            |                 *****************
                    |                         v (SALT)         |  version number        |                 * RFC 9580 AEAD  *
                    | (IKM)  +--------------------+  (INFO)    |  cipher algo octet     |                 * key derivation *
                    +-------->|        HKDF        | <----------|  AEAD algo octet       |                 *****************
                             +--------------------+            |  chunk size            |
                                        |                      |                        |
                                        | M + N - 64 bits      +------------------------+
                                        v
                      +-----------------+---------------+
                      | leftmost M bits | N - 64 bits    |
                      +-----------------+---------------+
                              |                   |
                              |                   |
                              |                   |
                              |                   |
                              |                   |                +---------------------+
                              |                   +----------------| 64 bit chunk number |
                        (KEY) |     +-------------+                +---------------------+
                              |     | (INIT. VEC.                  +------------------+                     +--------------------------------------
                              v     v  / NONCE )         +---------| plaintext chunk | ...             ... | (empty str., AD incl. plaint. size)
                           +------------------+ <-----------+      +------------------+                     +--------------------------------------
                           | AEAD encrypt.    |
                           +------------------+ --------+
                                           ^             |         +------------------+ +------------+     +---------------------+
         +--------------------+            |             +--------> | ciphertext chunk | | auth. tag | ... | final auth. tag     |
         | packet tag         |            |                       +------------------+ +------------+     +---------------------+
         | version nr.        |            |                                                        ^
         | cipher algo octet  |            |                                                        |
         | AEAD algo octet    | (ADD. DATA) |                                                further ciphertext
         | chunk size         |-------------+                                                chunks with tags
         |                    |
         +--------------------+
```