Foto: Eisenhans / © fotolia

(Post Quantum) Signatures in CMS, OpenPGP, and LibrePGP

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

Overview

- Cryptographic Message Formats with Digital Signature:
  - Cryptographic Message Syntax
  - OpenPGP
  - LibrePGP (OpenPGP fork)
- Our work on CMS was done in the scope of Project 480 – “PQC@Thunderbird”
  - Standardization and implementation of PQC in OpenPGP
  - draft-ietf-openpgpg-pqc

Overview

- Cryptographic Message Formats with Digital Signature:
  - Cryptographic Message Syntax
  - OpenPGP
  - LibrePGP (OpenPGP fork)
- Our work on CMS was done in the scope of Project 480 – “PQC@Thunderbird”
  - Standardization and implementation of PQC in OpenPGP
  - draft-ietf-openpgpg-pqc

Overview

- Cryptographic Message Formats with Digital Signature:
  - Cryptographic Message Syntax
  - OpenPGP
  - LibrePGP (OpenPGP fork)
- Our work on CMS was done in the scope of Project 480 – "PQC@Thunderbird"
  - Standardization and implementation of PQC in OpenPGP
  - draft-ietf-openpgpg-pqc

Overview

- Cryptographic Message Formats with Digital Signature:
  - Cryptographic Message Syntax
  - OpenPGP
  - LibrePGP (OpenPGP fork)
- Our work on CMS was done in the scope of Project 480 – "PQC@Thunderbird"
  - Standardization and implementation of PQC in OpenPGP
  - draft-ietf-openpgpg-pqc

Overview

- Cryptographic Message Formats with Digital Signature:
  - Cryptographic Message Syntax
  - OpenPGP
  - LibrePGP (OpenPGP fork)
- Our work on CMS was done in the scope of Project 480 – “PQC@Thunderbird”
  - Standardization and implementation of PQC in OpenPGP
  - draft-ietf-openpgpg-pqc

Overview

- Cryptographic Message Formats with Digital Signature:
  - Cryptographic Message Syntax
  - OpenPGP
  - LibrePGP (OpenPGP fork)
- Our work on CMS was done in the scope of Project 480 – “PQC@Thunderbird”
  - Standardization and implementation of PQC in OpenPGP
  - draft-ietf-openpgpg-pqc

Overview

- Cryptographic Message Formats with Digital Signature:
  - Cryptographic Message Syntax
  - OpenPGP
  - LibrePGP (OpenPGP fork)
- Our work on CMS was done in the scope of Project 480 – “PQC@Thunderbird”
  - Standardization and implementation of PQC in OpenPGP
  - draft-ietf-openpgpg-pqc

Introduction

CMS: Old EUF-CMA Violations

CMS: (generalized) EUF-CMA Problem in Current Proposal for Composite Signatures

OpenPGP: Natural Strong Non-Separability of Composite Signatures

LibrePGP: EUF-CMA Violation through Signature Version Aliasing

OpenPGP: Unsigned Packet Meta Data

Other Aspects of Post Quantum Signatures in Protocols

Introduction

CMS: Old EUF-CMA Violations

CMS: (generalized) EUF-CMA Problem in Current Proposal for Composite Signatures

OpenPGP: Natural Strong Non-Separability of Composite Signatures

LibrePGP: EUF-CMA Violation through Signature Version Aliasing

OpenPGP: Unsigned Packet Meta Data

Other Aspects of Post Quantum Signatures in Protocols

Introduction

- EUF-CMA problems with signatures on the protocol level:
  - ambiguity of what is signed (hashed)
  - EUF-CMA: Existential Unforgeability under Chosen Message Attack

Introduction

- EUF-CMA problems with signatures on the protocol level:
  - ambiguity of what is signed (hashed)
  - EUF-CMA: Existential Unforgeability under Chosen Message Attack

Introduction

- EUF-CMA problems with signatures on the protocol level:
  - ambiguity of what is signed (hashed)
  - EUF-CMA: Existential Unforgeability under Chosen Message Attack

EUF-CMA

- EUF-CMA game:
  - adversary can query signing oracle for any message
    - choose $\{m_i\}$
    - receive $\{s_i \mid s_i = \text{sign}(m_i)\}$
  - goal:
    - find valid signature for $m' \neq m_i \forall i$
- Generalization
  - EUF-CMA is restricted to <u>same</u> signature algorithm for <u>query</u> and <u>forgery</u>
  - generalization: allow different signature algorithms

EUF-CMA

- EUF-CMA game:
  - adversary can query signing oracle for any message
    - choose $\{m_i\}$
    - receive $\{s_i \mid s_i = \text{sign}(m_i)\}$
  - goal:
    - find valid signature for $m' \neq m_i \forall i$
- Generalization
  - EUF-CMA is restricted to same signature algorithm for query and forgery
  - generalization: allow different signature algorithms

EUF-CMA

- EUF-CMA game:
  - adversary can query signing oracle for any message
    - choose $\{m_i\}$
    - receive $\{s_i \mid s_i = \text{sign}(m_i)\}$
  - goal:
    - find valid signature for $m' \neq m_i \forall i$
- Generalization
  - EUF-CMA is restricted to same signature algorithm for query and forgery
  - generalization: allow different signature algorithms

EUF-CMA

- EUF-CMA game:
    - adversary can query signing oracle for any message
        - choose $\{m_i\}$
        - receive $\{s_i \mid s_i = \text{sign}(m_i)\}$
    - goal:
        - find valid signature for $m' \neq m_i \forall i$
- Generalization
    - EUF-CMA is restricted to same signature algorithm for query and forgery
    - generalization: allow different signature algorithms

EUF-CMA

- EUF-CMA game:
  - adversary can query signing oracle for any message
    - choose $\{m_i\}$
    - receive $\{s_i \mid s_i = \text{sign}(m_i)\}$
  - goal:
    - find valid signature for $m' \neq m_i \forall i$
- Generalization
  - EUF-CMA is restricted to same signature algorithm for query and forgery
  - generalization: allow different signature algorithms

EUF-CMA

- EUF-CMA game:
  - adversary can query signing oracle for any message
    - choose $\{m_i\}$
    - receive $\{s_i \mid s_i = \mathrm{sign}(m_i)\}$
  - goal:
    - find valid signature for $m' \neq m_i \forall i$
- Generalization
  - EUF-CMA is restricted to <u>same</u> signature algorithm for <u>query</u> and <u>forgery</u>
  - generalization: allow different signature algorithms

EUF-CMA

- EUF-CMA game:
  - adversary can query signing oracle for any message
    - choose $\{m_i\}$
    - receive $\{s_i \mid s_i = \text{sign}(m_i)\}$
  - goal:
    - find valid signature for $m' \neq m_i \forall i$
- Generalization
  - EUF-CMA is restricted to <u>same</u> signature algorithm for <u>query</u> and <u>forgery</u>
  - generalization: allow different signature algorithms

EUF-CMA

- EUF-CMA game:
  - adversary can query signing oracle for any message
    - choose $\{m_i\}$
    - receive $\{s_i \mid s_i = \text{sign}(m_i)\}$
  - goal:
    - find valid signature for $m' \neq m_i \forall i$
- Generalization
  - EUF-CMA is restricted to <u>same</u> signature algorithm for <u>query</u> and <u>forgery</u>
  - generalization: allow different signature algorithms

EUF-CMA

- EUF-CMA game:
  - adversary can query signing oracle for any message
    - choose $\{m_i\}$
    - receive $\{s_i \mid s_i = \mathrm{sign}(m_i)\}$
  - goal:
    - find valid signature for $m' \neq m_i \forall i$
- Generalization
  - EUF-CMA is restricted to same signature algorithm for query and forgery
  - generalization: allow different signature algorithms

Cryptographic Message Syntax (CMS)

- sign & encrypt etc. based on X.509 certificates
- protocols building on CMS
  - S/MIME
  - German Smart Metering
  - ...
- as PKCS#7 since 1998

Cryptographic Message Syntax (CMS)

- sign & encrypt etc. based on X.509 certificates
- protocols building on CMS
  - S/MIME
  - German Smart Metering
  - ...
- as PKCS#7 since 1998

Cryptographic Message Syntax (CMS)

- sign & encrypt etc. based on X.509 certificates
- protocols building on CMS
  - S/MIME
  - German Smart Metering
  - ...
- as PKCS#7 since 1998

Cryptographic Message Syntax (CMS)

- sign & encrypt etc. based on X.509 certificates
- protocols building on CMS
  - S/MIME
  - German Smart Metering
  - ...
- as PKCS#7 since 1998

Cryptographic Message Syntax (CMS)

- sign & encrypt etc. based on X.509 certificates
- protocols building on CMS
  - S/MIME
  - German Smart Metering
  - ...
- as PKCS#7 since 1998

Cryptographic Message Syntax (CMS)

- sign & encrypt etc. based on X.509 certificates
- protocols building on CMS
  - S/MIME
  - German Smart Metering
  - . . .
- as PKCS#7 since 1998

Introduction

CMS: Old EUF-CMA Violations

CMS: (generalized) EUF-CMA Problem in Current Proposal for Composite Signatures

OpenPGP: Natural Strong Non-Separability of Composite Signatures

LibrePGP: EUF-CMA Violation through Signature Version Aliasing

OpenPGP: Unsigned Packet Meta Data

Other Aspects of Post Quantum Signatures in Protocols

Old EUF-CMA Violations in CMS

- CMS signs message content directly
- i.e., no metadata is signed
- source of the problem:
  - CMS allows 2 variants of what is signed
- Falko Strenzke: "ForgedAttributes: An Existential Forgery Vulnerability of CMS and PKCS#7 Signatures"
  - `https://eprint.iacr.org/2023/1801`

Old EUF-CMA Violations in CMS

- CMS signs message content directly
- i.e., no metadata is signed
- source of the problem:
  - CMS allows 2 variants of what is signed
- Falko Strenzke: "ForgedAttributes: An Existential Forgery Vulnerability of CMS and PKCS#7 Signatures"
  - `https://eprint.iacr.org/2023/1801`

Old EUF-CMA Violations in CMS

- CMS signs message content directly
- i.e., no metadata is signed
- source of the problem:
  - CMS allows 2 variants of what is signed
- Falko Strenzke: “ForgedAttributes: An Existential Forgery Vulnerability of CMS and PKCS#7 Signatures”
  - `https://eprint.iacr.org/2023/1801`

Old EUF-CMA Violations in CMS

- CMS signs message content directly
- i.e., no metadata is signed
- source of the problem:
  - CMS allows 2 variants of what is signed
- Falko Strenzke: “ForgedAttributes: An Existential Forgery Vulnerability of CMS and PKCS#7 Signatures”
  - `https://eprint.iacr.org/2023/1801`

Old EUF-CMA Violations in CMS

- CMS signs message content directly
- i.e., no metadata is signed
- source of the problem:
  - CMS allows 2 variants of what is signed
- Falko Strenzke: "ForgedAttributes: An Existential Forgery Vulnerability of CMS and PKCS#7 Signatures"
  - `https://eprint.iacr.org/2023/1801`

Old EUF-CMA Violations in CMS

- CMS signs message content directly
- i.e., no metadata is signed
- source of the problem:
  - CMS allows 2 variants of what is signed
- Falko Strenzke: “ForgedAttributes: An Existential Forgery Vulnerability of CMS and PKCS#7 Signatures”
  - `https://eprint.iacr.org/2023/1801`

SignerInfo Structure

```
 SignerInfo ::= SEQUENCE {
        version CMSVersion,
        sid SignerIdentifier,
        digestAlgorithm DigestAlgorithmIdentifier,
        signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
        signatureAlgorithm SignatureAlgorithmIdentifier,
        signature SignatureValue,
        unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL }

SignedAttributes ::= SET SIZE (1..MAX) OF Attribute


Attribute ::= SEQUENCE {
        attrType OBJECT IDENTIFIER,
        attrValues SET OF AttributeValue }

AttributeValue ::= ANY
```

- signedAttrs:
  - messageDigest attribute:
    - contains Hash(M):

      ```
      messageDigestAttr ::= SEQUENCE {
              attrType OBJECT IDENTIFIER,
              attrValues SET {
                messageDigest OCTET STRING } }
      ```
  - $\mathrm{signedAttr}_M^{\mathrm{DER}} = \mathrm{DER\text{-}encode}(\mathrm{signedAttrs}(M))$
    - to indicate they contain Hash(M)

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

```
procedure CMS-SIGN( secret key $K_s$, message $M$ )
    if signedAttrs are absent then
        $D = \mathrm{HASH}(M)$
    else
        $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$
    end if
    return sign$(K_s, D)$
end procedure
```

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

```
procedure CMS-Sign( secret key $K_s$, message $M$ )
    if signedAttrs are absent then
        $D = \mathrm{HASH}(M)$ // $M = \mathrm{signedAttr}_{M'}^{\mathrm{DER}} \leftarrow$ 👻
    else
        $D = \mathrm{HASH}(\mathrm{signedAttr}_{M}^{\mathrm{DER}})$
    end if
    return sign$(K_s, D)$
end procedure
```

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

```
procedure CMS-SIGN( secret key $K_s$, message $M$ )
    if signedAttrs are absent then
        $D = \mathrm{HASH}(M)$ // $M = \mathrm{signedAttr}_{M'}^{\mathrm{DER}} \leftarrow$ 👻
    else
        $D = \mathrm{HASH}(\mathrm{signedAttr}_{M'}^{\mathrm{DER}})$
    end if
    return sign($K_s, D$)
end procedure
```

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

```
procedure CMS-Sign( secret key $K_s$, message $M$ )
    if signedAttrs are absent then
        $D = \text{HASH}(M)$ // $M = \text{signedAttr}^{\text{DER}}_{M'} \leftarrow$ 👻
    else
        $D = \text{HASH}(\text{signedAttr}^{\text{DER}}_{M'})$ // ←↑ cannot distinguish, signature valid for $M'$
    end if                                          // (👻 adds signedAttrs to ✉)
    return sign($K_s, D$)
end procedure
```

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

```
procedure CMS-SIGN( secret key $K_s$, message $M$ )
    if signedAttrs are absent then
        $D = \mathrm{HASH}(M)$ // $M = \mathrm{signedAttr}^{\mathrm{DER}}_{M'} \leftarrow$ 👻
    else
        $D = \mathrm{HASH}(\mathrm{signedAttr}^{\mathrm{DER}}_{M'})$   // ←↑ cannot distinguish, signature valid for $M'$
    end if                                        // (👻 adds signedAttrs to ✉)
    return sign$(K_s, D)$
end procedure
```

→ Can forge signatures for arbitrary attacker-chosen message

Attack variant 2: Let the signer sign any message with signedAttrs:

```
procedure CMS-SIGN( secret key $K_s$, message $M$ )
    if signedAttrs are absent then
        $D = \mathrm{HASH}(M)$
    else
        $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$
    end if
    return sign($K_s, D$)
end procedure
```

Attack variant 2: Let the signer sign any message with signedAttrs:

```
procedure $\textsc{CMS-Sign}$( secret key $K_s$, message $M$ )
    if signedAttrs are absent then
        $D = \mathrm{HASH}(M)$
    else
        $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$ // $M' = \mathrm{signedAttr}_M^{\mathrm{DER}} \leftarrow$ 👻
    end if
    return sign$(K_s, D)$
end procedure
```

Attack variant 2: Let the signer sign any message with signedAttrs:

```
procedure CMS-Sign( secret key $K_s$, message $M$ )
    if signedAttrs are absent then
        $D$ = HASH( M' ) // ← cannot be distinguished from this case (remove signedAttrs)
    else
        $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$ // $M' = \mathrm{signedAttr}_M^{\mathrm{DER}} \leftarrow$ 👻
    end if
    return sign($K_s, D$)
end procedure
```

Attack variant 2: Let the signer sign any message with signedAttrs:

```
procedure CMS-Sign( secret key $K_s$, message $M$ )
    if signedAttrs are absent then
        $D = \mathrm{HASH}(\text{M'})$ // ← cannot be distinguished from this case (remove signedAttrs)
    else
        $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$ // $M' = \mathrm{signedAttr}_M^{\mathrm{DER}} \leftarrow$ 👻
    end if
    return sign($K_s, D$)
end procedure
```

→ Can forge signatures for message of form $\mathrm{signedAttr}_M^{\mathrm{DER}}$

Format of the signedAttrs when generated by attacker (attack variant 1)

```
31 4b 30 18 06 09 2b 80 40 80 f0 0d 01 09 03 31   // fake OID / attribute
...                                               // further fake content
30 18 06 09 2a 86 48 86 f7 0d 01 09 03 31
0b 06 09 2a 86 48 86 f7 0d 01 07 01 30 2f 06 09
2a 86 48 86 f7 0d 01 09 04 31 22 04 20 e0 be bd
22 81 99 93 42 58 14 86 6b 62 70 1e 29 19 ea 26
f1 37 04 99 c1 03 7b 53 b9 d4 9c 2c 8a
30 ...
```

- fixed
- variable / attacker chosen
- assumption: attacker can make up own OID for unknown attribute [1]
- structure must contain mandatory attributes (messageDigest, contentType)

[1] https://datatracker.ietf.org/doc/html/rfc5652#section-2

Format of the signedAttrs when generated by signer (attack variant 2)

```
31 4b 30 18 06 09 2a 86 48 86 f7 0d 01 09 03 31
0b 06 09 2a 86 48 86 f7 0d 01 07 01 30 2f 06 09
2a 86 48 86 f7 0d 01 09 04 31 22 04 20 e0 be bd  // message digest
22 81 99 93 42 58 14 86 6b 62 70 1e 29 19 ea 26
f1 37 04 99 c1 03 7b 53 b9 d4 9c 2c 8a
```

- fixed (order and set of attributes may still vary, this is not indicated here)
- variable / potentially influenced by attacker

Conceivable vulnerable applications

- directly signing a firmware image
- dense message space (machine-to-machine)
- signing unstructured data – e.g. tokens
- ≈ strongest: external signatures over unstructured secret data with absent content:
    - attacker removes signedAttrs
    - attacker can produces signature over a secret chosen by them

Conceivable vulnerable applications

- directly signing a firmware image
- dense message space (machine-to-machine)
- signing unstructured data – e.g. tokens
- ≈ strongest: external signatures over unstructured secret data with absent content:
  - attacker removes signedAttrs
  - attacker can produces signature over a secret chosen by them

Conceivable vulnerable applications

- directly signing a firmware image
- dense message space (machine-to-machine)
- signing unstructured data – e.g. tokens
- ≈ strongest: external signatures over unstructured secret data with absent content:
  - attacker removes signedAttrs
  - attacker can produces signature over a secret chosen by them

Conceivable vulnerable applications

- directly signing a firmware image
- dense message space (machine-to-machine)
- signing unstructured data – e.g. tokens
- ≈ strongest: external signatures over unstructured secret data with absent content:
  - attacker removes signedAttrs
  - attacker can produces signature over a secret chosen by them

Conceivable vulnerable applications

- directly signing a firmware image
- dense message space (machine-to-machine)
- signing unstructured data – e.g. tokens
- ≈ strongest: external signatures over unstructured secret data with absent content:
  - attacker removes signedAttrs
  - attacker can produces signature over a secret chosen by them

Conceivable vulnerable applications

- directly signing a firmware image
- dense message space (machine-to-machine)
- signing unstructured data – e.g. tokens
- ≈ strongest: external signatures over unstructured secret data with absent content:
  - attacker removes signedAttrs
  - attacker can produces signature over a secret chosen by them

Summary: EUF-CMA violation in CMS through signedAttrs

MTG

- Two signature variants:
  - with signedAttrs (then they are signed)
  - without signedAttrs
- choice of these two variants is **not** protected by signature
- forgeries restricted:
  - either signer has to sign a message in specific format; forged message is arbitrary
  - or the signer has to sign any message; forged signature format is in specific format

Summary: EUF-CMA violation in CMS through signedAttrs

- Two signature variants:
  - with signedAttrs (then they are signed)
  - without signedAttrs
- choice of these two variants is **not** protected by signature
- forgeries restricted:
  - either signer has to sign a message in specific format; forged message is arbitrary
  - or the signer has to sign any message; forged signature format is in specific format

Summary: EUF-CMA violation in CMS through signedAttrs

- Two signature variants:
  - with signedAttrs (then they are signed)
  - without signedAttrs
- choice of these two variants is **not** protected by signature
- forgeries restricted:
  - either signer has to sign a message in specific format; forged message is arbitrary
  - or the signer has to sign any message; forged signature format is in specific format

Summary: EUF-CMA violation in CMS through signedAttrs

- Two signature variants:
  - with signedAttrs (then they are signed)
  - without signedAttrs
- choice of these two variants is **not** protected by signature
- forgeries restricted:
  - either signer has to sign a message in specific format; forged message is arbitrary
  - or the signer has to sign any message; forged signature format is in specific format

Summary: EUF-CMA violation in CMS through signedAttrs

- Two signature variants:
  - with signedAttrs (then they are signed)
  - without signedAttrs
- choice of these two variants is **not** protected by signature
- forgeries restricted:
  - either signer has to sign a message in specific format; forged message is arbitrary
  - or the signer has to sign any message; forged signature format is in specific format

Summary: EUF-CMA violation in CMS through signedAttrs

- Two signature variants:
  - with signedAttrs (then they are signed)
  - without signedAttrs
- choice of these two variants is **not** protected by signature
- forgeries restricted:
  - either signer has to sign a message in specific format; forged message is arbitrary
  - or the signer has to sign any message; forged signature format is in specific format

Summary: EUF-CMA violation in CMS through signedAttrs

- Two signature variants:
  - with signedAttrs (then they are signed)
  - without signedAttrs
- choice of these two variants is **not** protected by signature
- forgeries restricted:
  - either signer has to sign a message in specific format; forged message is arbitrary
  - or the signer has to sign any message; forged signature format is in specific format

General Countermeasure

- hardened implementations: prohibit messages of the form of signedAttrs
  - during signing
  - and verification
- enforce use of signedAttrs on the application level
  - some protocols already do it
- prohibit the use of signedAttrs
  - would rather be a step backwards
  - modern approch is to use signedAttrs always

General Countermeasure

- hardened implementations: prohibit messages of the form of signedAttrs
  - during signing
  - and verification
- enforce use of signedAttrs on the application level
  - some protocols already do it
- prohibit the use of signedAttrs
  - would rather be a step backwards
  - modern approch is to use signedAttrs always

General Countermeasure

- hardened implementations: prohibit messages of the form of signedAttrs
  - during signing
  - and verification
- enforce use of signedAttrs on the application level
  - some protocols already do it
- prohibit the use of signedAttrs
  - would rather be a step backwards
  - modern approch is to use signedAttrs always

General Countermeasure

- hardened implementations: prohibit messages of the form of signedAttrs
  - during signing
  - and verification
- enforce use of signedAttrs on the application level
  - some protocols already do it
- prohibit the use of signedAttrs
  - would rather be a step backwards
  - modern approch is to use signedAttrs always

General Countermeasure

- hardened implementations: prohibit messages of the form of signedAttrs
  - during signing
  - and verification
- enforce use of signedAttrs on the application level
  - some protocols already do it
- prohibit the use of signedAttrs
  - would rather be a step backwards
  - modern approch is to use signedAttrs always

General Countermeasure

- hardened implementations: prohibit messages of the form of signedAttrs
  - during signing
  - and verification
- enforce use of signedAttrs on the application level
  - some protocols already do it
- prohibit the use of signedAttrs
  - would rather be a step backwards
  - modern approch is to use signedAttrs always

General Countermeasure

- hardened implementations: prohibit messages of the form of signedAttrs
  - during signing
  - and verification
- enforce use of signedAttrs on the application level
  - some protocols already do it
- prohibit the use of signedAttrs
  - would rather be a step backwards
  - modern approch is to use signedAttrs always

General Countermeasure

- hardened implementations: prohibit messages of the form of signedAttrs
  - during signing
  - and verification
- enforce use of signedAttrs on the application level
  - some protocols already do it
- prohibit the use of signedAttrs
  - would rather be a step backwards
  - modern approch is to use signedAttrs always

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - h = internal-hash(len(ctx) || ctx || message)
    - then "actually" sign h
  - `ctx=‘‘with_signedAttrs’’`
  - `ctx=‘‘without_signedAttrs’’`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - $h =$ internal-hash(len(ctx) $\|$ ctx $\|$ message)
    - then "actually" sign h
  - `ctx=‘‘with_signedAttrs’’`
  - `ctx=‘‘without_signedAttrs’’`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - h = internal-hash(len(ctx) $\|$ ctx $\|$ message)
    - then “actually” sign h
  - `ctx=‘‘with_signedAttrs’’`
  - `ctx=‘‘without_signedAttrs’’`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - h = internal-hash(len(ctx) || ctx || message)
    - then "actually" sign h
  - ctx=‘‘with_signedAttrs’’
  - ctx=‘‘without_signedAttrs’’
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - h = internal-hash(len(ctx) || ctx || message)
    - then “actually” sign h
  - `ctx=‘‘with_signedAttrs’’`
  - `ctx=‘‘without_signedAttrs’’`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - h = internal-hash(len(ctx) || ctx || message)
    - then "actually" sign h
  - `ctx=''with_signedAttrs''`
  - `ctx=''without_signedAttrs''`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - h = internal-hash(len(ctx) || ctx || message)
    - then "actually" sign h
  - `ctx=‘‘with_signedAttrs’’`
  - `ctx=‘‘without_signedAttrs’’`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - $h =$ internal-hash(len(ctx) $\|$ ctx $\|$ message)
    - then “actually” sign h
  - ctx=‘‘with_signedAttrs’’
  - ctx=‘‘without_signedAttrs’’
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses …
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - h = internal-hash(len(ctx) || ctx || message)
    - then “actually” sign h
  - `ctx=‘‘with_signedAttrs’’`
  - `ctx=‘‘without_signedAttrs’’`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - h = internal-hash(len(ctx) || ctx || message)
    - then "actually" sign h
  - `ctx=''with_signedAttrs''`
  - `ctx=''without_signedAttrs''`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - h = internal-hash(len(ctx) || ctx || message)
    - then "actually" sign h
  - `ctx=''with_signedAttrs''`
  - `ctx=''without_signedAttrs''`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses . . .
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Countermeasures for PQC Algorithms

- PQC Algorithms ML-DSA and SLH-DSA define a context parameter
  - internally, the context is fed to the preprocessing hash
    - $h = \text{internal-hash}(\text{len}(\text{ctx}) \,||\, \text{ctx} \,||\, \text{message})$
    - then "actually" sign h
  - `ctx=''with_signedAttrs''`
  - `ctx=''without_signedAttrs''`
- context achieves domain separation
  - between signing w/ and w/o signedAttrs
  - can be extended to other uses ...
- Had presentation in LAMPS at IETF 121 with Daniel van Geest (Cryptonext)
- Hopefully an IETF draft
  - proposing different countermeasures

Introduction

CMS: Old EUF-CMA Violations

CMS: (generalized) EUF-CMA Problem in Current Proposal for Composite Signatures

OpenPGP: Natural Strong Non-Separability of Composite Signatures

LibrePGP: EUF-CMA Violation through Signature Version Aliasing

OpenPGP: Unsigned Packet Meta Data

Other Aspects of Post Quantum Signatures in Protocols

Hybrid / Multi-Algorithm Signatures

- insufficient trust in ML-DSA
- general recommendation: Multi-Algorithm Signatures
    - $s_1 = \text{sign-ECDSA}(m)$
    - $s_2 = \text{sign-ML-DSA}(m)$
    - $s = s_1 || s_2$

Hybrid / Multi-Algorithm Signatures

- insufficient trust in ML-DSA
- general recommendation: Multi-Algorithm Signatures
  - $s_1 = \text{sign-ECDSA}(m)$
  - $s_2 = \text{sign-ML-DSA}(m)$
  - $s = s_1 || s_2$

Hybrid / Multi-Algorithm Signatures

- insufficient trust in ML-DSA
- general recommendation: Multi-Algorithm Signatures
  - s_1 = sign-ECDSA(m)
  - s_2 = sign-ML-DSA(m)
  - $s = s_1 || s_2$

Hybrid / Multi-Algorithm Signatures

- insufficient trust in ML-DSA
- general recommendation: Multi-Algorithm Signatures
  - s_1 = sign-ECDSA(m)
  - s_2 = sign-ML-DSA(m)
  - $s = s_1 || s_2$

Hybrid / Multi-Algorithm Signatures

- insufficient trust in ML-DSA
- general recommendation: Multi-Algorithm Signatures
  - $s_1 = \text{sign-ECDSA}(m)$
  - $s_2 = \text{sign-ML-DSA}(m)$
  - $s = s_1 || s_2$

Protection against Signature Stripping Attacks

- Signature stripping attack:
  - Adversary removes one of the two signatures
  - → Standalone signature
  - with simple parallel signatures, this has no security implications
    - no change of message
    - verifier must always accept only secure signatures
    - verification at later point can be affected (availability)
- requires key-reuse
  - ECDSA-🔑$_1$ → ECDSA-standalone 📜 $_1$
  - ECDSA-🔑$_1$ → ML-DSA+ECDSA 📜 $_2$
  - **not** allowed by draft-ietf-lamps-pq-composite-sigs-03

EUF-CMA Problem in Current Proposal for Composite Signatures

- draft-ietf-lamps-pq-composite-sigs-03
- relevant for ML-DSA+X
- `Composite-ML-DSA.Sign (sk, M, ctx)`

```
M’ = OID || len(ctx) || ctx || M
mldsaSig = ML-DSA.Sign( key=mldsaSK, msg=M’, ctx=OID )
tradSig = Trad.Sign( tradSK, M’ )
```

- Aim of countermeasure: achieve Weak-Non-Separability
  - Leaves "artifact" in the message
  - ϟ Artifact is violation of (generalized) EUF-CMA
    - generalized because cross-algorithm CMA is needed
- (generalized) EUF-CMA forgeries
  - remove PQ part from "ML-DSA+ECDSA" signature
  - ECDSA signature valid
  - forged message: `OID` || `len(ctx)` || `ctx` || `M`

- Aim of countermeasure: achieve Weak-Non-Separability
  - Leaves “artifact” in the message
  - ϟ Artifact is violation of (generalized) EUF-CMA
    - generalized because cross-algorithm CMA is needed
- (generalized) EUF-CMA forgeries
  - remove PQ part from “ML-DSA+ECDSA” signature
  - ECDSA signature valid
  - forged message: $\texttt{OID} \,\|\, \texttt{len(ctx)} \,\|\, \texttt{ctx} \,\|\, \texttt{M}$

- Aim of countermeasure: achieve Weak-Non-Separability
    - Leaves "artifact" in the message
    - ↯ Artifact is violation of (generalized) EUF-CMA
        - generalized because cross-algorithm CMA is needed
- (generalized) EUF-CMA forgeries
    - remove PQ part from "ML-DSA+ECDSA" signature
    - ECDSA signature valid
    - forged message: `OID` $\|$ `len(ctx)` $\|$ `ctx` $\|$ `M`

- Aim of countermeasure: achieve Weak-Non-Separability
  - Leaves “artifact” in the message
  - ϟ Artifact is violation of (generalized) EUF-CMA
    - generalized because cross-algorithm CMA is needed
- (generalized) EUF-CMA forgeries
  - remove PQ part from “ML-DSA+ECDSA” signature
  - ECDSA signature valid
  - forged message: `OID` $\|$ `len(ctx)` $\|$ `ctx` $\|$ `M`

- Aim of countermeasure: achieve Weak-Non-Separability
  - Leaves “artifact” in the message
  - ϟ Artifact is violation of (generalized) EUF-CMA
    - generalized because cross-algorithm CMA is needed
- (generalized) EUF-CMA forgeries
  - remove PQ part from “ML-DSA+ECDSA” signature
  - ECDSA signature valid
  - forged message: $\texttt{OID} \parallel \texttt{len(ctx)} \parallel \texttt{ctx} \parallel \texttt{M}$

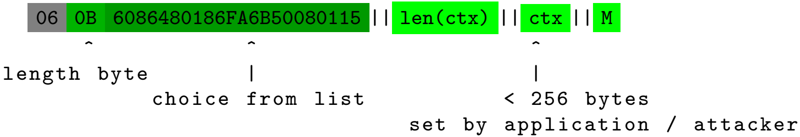
- Aim of countermeasure: achieve Weak-Non-Separability
  - Leaves “artifact” in the message
  - ↯ Artifact is violation of (generalized) EUF-CMA
    - generalized because cross-algorithm CMA is needed
- (generalized) EUF-CMA forgeries
  - remove PQ part from “ML-DSA+ECDSA” signature
  - ECDSA signature valid
  - forged message: `OID` ∥ `len(ctx)` ∥ `ctx` ∥ `M`

- Aim of countermeasure: achieve Weak-Non-Separability
  - Leaves “artifact” in the message
  - ↯ Artifact is violation of (generalized) EUF-CMA
    - generalized because cross-algorithm CMA is needed
- (generalized) EUF-CMA forgeries
  - remove PQ part from “ML-DSA+ECDSA” signature
  - ECDSA signature valid
  - forged message: `OID` $\|$ `len(ctx)` $\|$ `ctx` $\|$ `M`

- Aim of countermeasure: achieve Weak-Non-Separability
  - Leaves "artifact" in the message
  - ↯ Artifact is violation of (generalized) EUF-CMA
    - generalized because cross-algorithm CMA is needed
- (generalized) EUF-CMA forgeries
  - remove PQ part from "ML-DSA+ECDSA" signature
  - ECDSA signature valid
  - forged message: `OID` ‖ `len(ctx)` ‖ `ctx` ‖ `M`

Forged Message

- forged message: `OID || len(ctx) || ctx || M`
- OID: predefined list, but variable content



```
  06 0B 6086480186FA6B50080115 || len(ctx) || ctx || M
      ^           ^                    ^
length byte       |                    |
           choice from list          < 256 bytes
                          set by application / attacker
```

- protocols with valid messages starting with 06 potentially affected

Countermeasure

- Countermeasure: detectable constant prefix

  `<32 magic bytes> 06 0B 6086480186FA6B50080115 || len(ctx) || ctx || M`

- newer implementations can check for the magic bytes → attack detection

Introduction

CMS: Old EUF-CMA Violations

CMS: (generalized) EUF-CMA Problem in Current Proposal for Composite Signatures

OpenPGP: Natural Strong Non-Separability of Composite Signatures

LibrePGP: EUF-CMA Violation through Signature Version Aliasing

OpenPGP: Unsigned Packet Meta Data

Other Aspects of Post Quantum Signatures in Protocols

Non-Separability in OpenPGP Signatures

v6 signature packet

v6 = 0x06
sig-type 0x00
1B pk-algo = hybrid-. . .
1B hash-algo
2B hashed subpacket length
hashed subpackets
2B unhashed subpacket length
unhashed subpackets
2B checksum for hash-value
algorithm-specific signature data

sig 1

sig 2

Non-Separability in OpenPGP Signatures

v6 signature packet

hashed as meta data

- **v6 = 0x06**
- **sig-type 0x00**
- **1B pk-algo = hybrid-. . .**
- **1B hash-algo**
- **2B hashed subpacket length**
- **hashed subpackets**

2B unhashed subpacket length

unhashed subpackets

2B checksum for hash-value

algorithm-specific signature data

sig 1

sig 2

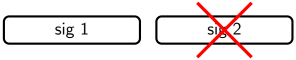
Non-Separability in OpenPGP Signatures

v6 signature packet

hashed as meta data

- **v6 = 0x06**
- **sig-type 0x00**
- **1B pk-algo = hybrid-. . .**
- **1B hash-algo**
- **2B hashed subpacket length**
- **hashed subpackets**

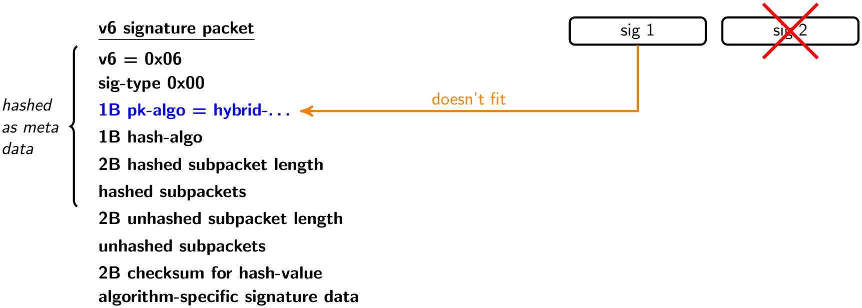
- **2B unhashed subpacket length**
- **unhashed subpackets**
- **2B checksum for hash-value**
- **algorithm-specific signature data**



sig 1

~~sig 2~~

Non-Separability in OpenPGP Signatures



v6 signature packet

v6 = 0x06

sig-type 0x00

1B pk-algo = hybrid-. . .

1B hash-algo

2B hashed subpacket length

hashed subpackets

2B unhashed subpacket length

unhashed subpackets

2B checksum for hash-value

algorithm-specific signature data

Introduction

CMS: Old EUF-CMA Violations

CMS: (generalized) EUF-CMA Problem in Current Proposal for Composite Signatures

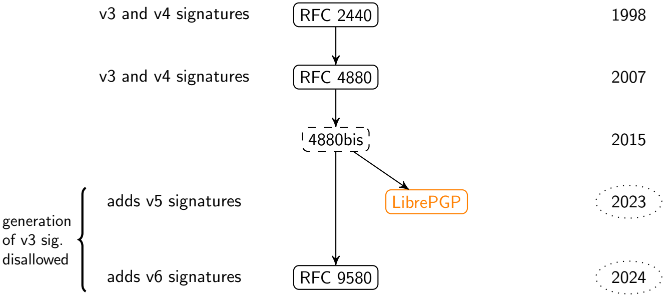
OpenPGP: Natural Strong Non-Separability of Composite Signatures

LibrePGP: EUF-CMA Violation through Signature Version Aliasing

OpenPGP: Unsigned Packet Meta Data

Other Aspects of Post Quantum Signatures in Protocols

LibrePGP


v3 and v4 signatures
RFC 2440
1998
v3 and v4 signatures
RFC 4880
2007
4880bis
2015
adds v5 signatures
LibrePGP
2023
generation of v3 sig. disallowed
adds v6 signatures
RFC 9580
2024

LibrePGP v5 Signatures

v5 signature packet

v5 = 0x05

sig-type 0x00 *// document signature*

1B pk-algo

1B hash-algo

2B hashed subpacket length

hashed subpackets

2B unhashed subpacket length

unhashed subpackets

2B checksum for hash-value
algorithm-specific signature data

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple “interpretations” of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - “semantics → hashed data”: always given
  - “semantics ← hashed data”: not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple “interpretations” of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - “semantics → hashed data”: always given
  - “semantics ← hashed data”: not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple “interpretations” of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - “semantics → hashed data”: always given
  - “semantics ← hashed data”: not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple "interpretations" of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - "semantics → hashed data": always given
  - "semantics ← hashed data": not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple "interpretations" of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - "semantics → hashed data": always given
  - "semantics ← hashed data": not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple “interpretations” of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - “semantics → hashed data”: always given
  - “semantics ← hashed data”: not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple “interpretations” of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - “semantics → hashed data”: always given
  - “semantics ← hashed data”: not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple “interpretations” of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - “semantics → hashed data”: always given
  - “semantics ← hashed data”: not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple "interpretations" of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - "semantics → hashed data": always given
  - "semantics ← hashed data": not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple "interpretations" of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - "semantics → hashed data": always given
  - "semantics ← hashed data": not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple “interpretations” of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - “semantics → hashed data”: always given
  - “semantics ← hashed data”: not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

LibrePGP v5 - v3 Signature Aliasing

- Signature aliasing:
  - hashed data is ambiguous
  - → multiple “interpretations” of what was signed
- requirement:
  - injective / one-to-one mapping of protocol semantics to hashed data
  - “semantics → hashed data”: always given
  - “semantics ← hashed data”: not necessarily
    - hashed data needs to be uniquely parseable
    - from front, rear, or both
- example: hash first and last name together:
  - maxi + müller → maximüller
  - maxim + üller → maximüller

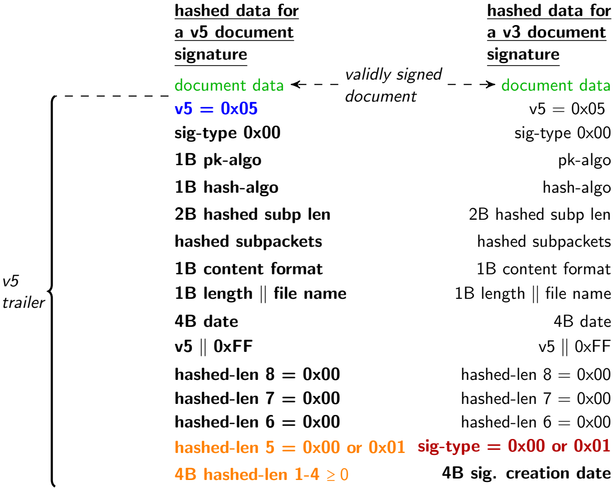
LibrePGP v5 - v3 Signatures Aliasing

hashed data for a v5 document signature	hashed data for a v3 document signature
document data	document data
v5 = 0x05	v5 = 0x05
sig-type 0x00	sig-type 0x00
1B pk-algo	pk-algo
1B hash-algo	hash-algo
2B hashed subp len	2B hashed subp len
hashed subpackets	hashed subpackets
1B content format	1B content format
1B length || file name	1B length || file name
4B date	4B date
v5 || 0xFF	v5 || 0xFF
hashed-len 8 = 0x00	hashed-len 8 = 0x00
hashed-len 7 = 0x00	hashed-len 7 = 0x00
hashed-len 6 = 0x00	hashed-len 6 = 0x00
hashed-len 5 = 0x00 or 0x01	**sig-type = 0x00 or 0x01**
4B hashed-len 1-4 ≥ 0	**4B sig. creation date**

LibrePGP v5 - v3 Signatures Aliasing

hashed data for a v5 document signature		**hashed data for a v3 document signature**
document data ←	*validly signed document*	→ document data
v5 = 0x05		v5 = 0x05
sig-type 0x00		sig-type 0x00
1B pk-algo		pk-algo
1B hash-algo		hash-algo
2B hashed subp len		2B hashed subp len
hashed subpackets		hashed subpackets
1B content format		1B content format
1B length || file name		1B length || file name
4B date		4B date
v5 || 0xFF		v5 || 0xFF
hashed-len 8 = 0x00		hashed-len 8 = 0x00
hashed-len 7 = 0x00		hashed-len 7 = 0x00
hashed-len 6 = 0x00		hashed-len 6 = 0x00
hashed-len 5 = 0x00 or 0x01		**sig-type = 0x00 or 0x01**
4B hashed-len 1-4 ≥ 0		**4B sig. creation date**

LibrePGP v5 - v3 Signatures Aliasing

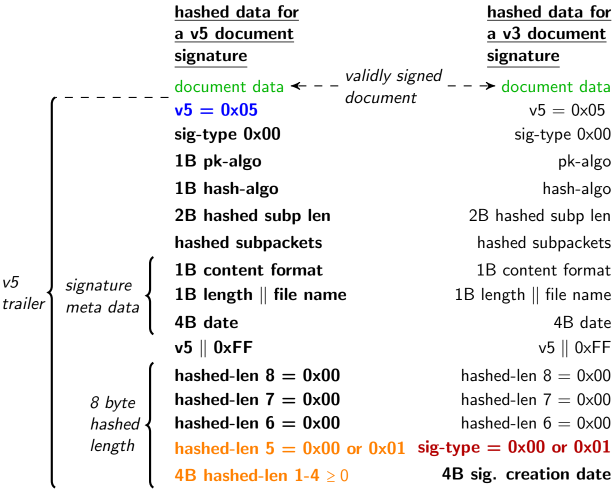


hashed data for a v5 document signature		hashed data for a v3 document signature
document data	validly signed document	document data
v5 = 0x05		v5 = 0x05
sig-type 0x00		sig-type 0x00
1B pk-algo		pk-algo
1B hash-algo		hash-algo
2B hashed subp len		2B hashed subp len
hashed subpackets		hashed subpackets
1B content format		1B content format
1B length ‖ file name		1B length ‖ file name
4B date		4B date
v5 ‖ 0xFF		v5 ‖ 0xFF
hashed-len 8 = 0x00		hashed-len 8 = 0x00
hashed-len 7 = 0x00		hashed-len 7 = 0x00
hashed-len 6 = 0x00		hashed-len 6 = 0x00
hashed-len 5 = 0x00 or 0x01		sig-type = 0x00 or 0x01
4B hashed-len 1-4 ≥ 0		4B sig. creation date

LibrePGP v5 - v3 Signatures Aliasing

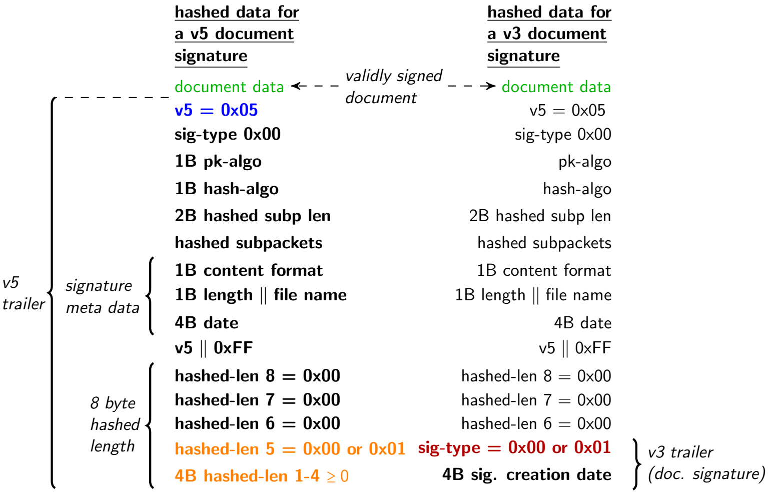
		hashed data for a v5 document signature		**hashed data for a v3 document signature**
		document data	← – – *validly signed document* – – →	document data
v5 trailer		**v5 = 0x05**		v5 = 0x05
		sig-type 0x00		sig-type 0x00
		1B pk-algo		pk-algo
		1B hash-algo		hash-algo
		2B hashed subp len		2B hashed subp len
		hashed subpackets		hashed subpackets
	signature meta data	**1B content format**		1B content format
		1B length ‖ file name		1B length ‖ file name
		4B date		4B date
		v5 ‖ 0xFF		v5 ‖ 0xFF
		hashed-len 8 = 0x00		hashed-len 8 = 0x00
		hashed-len 7 = 0x00		hashed-len 7 = 0x00
		hashed-len 6 = 0x00		hashed-len 6 = 0x00
		hashed-len 5 = 0x00 or 0x01		**sig-type = 0x00 or 0x01**
		4B hashed-len 1-4 ≥ 0		**4B sig. creation date**

LibrePGP v5 - v3 Signatures Aliasing



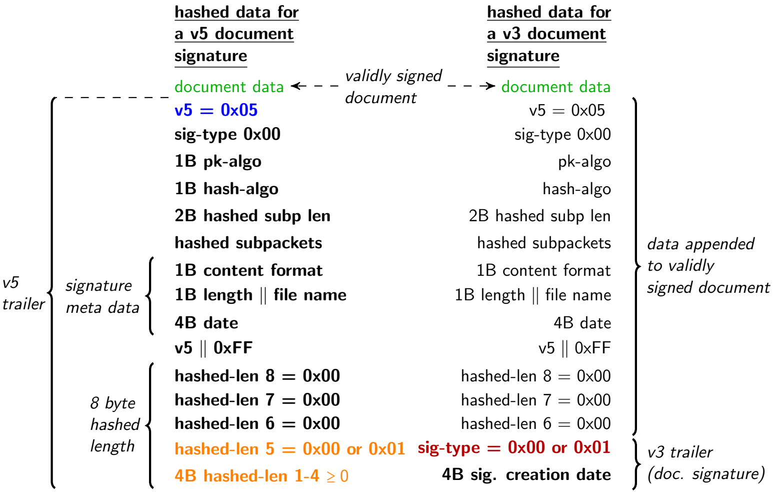
		hashed data for a v5 document signature		**hashed data for a v3 document signature**
		document data	← validly signed document →	document data
v5 trailer		**v5 = 0x05**		v5 = 0x05
		sig-type 0x00		sig-type 0x00
		1B pk-algo		pk-algo
		1B hash-algo		hash-algo
		2B hashed subp len		2B hashed subp len
		hashed subpackets		hashed subpackets
	signature meta data	**1B content format**		1B content format
		1B length ‖ file name		1B length ‖ file name
		4B date		4B date
		v5 ‖ 0xFF		v5 ‖ 0xFF
	8 byte hashed length	**hashed-len 8 = 0x00**		hashed-len 8 = 0x00
		hashed-len 7 = 0x00		hashed-len 7 = 0x00
		hashed-len 6 = 0x00		hashed-len 6 = 0x00
		hashed-len 5 = 0x00 or 0x01		**sig-type = 0x00 or 0x01**
		4B hashed-len 1-4 ≥ 0		**4B sig. creation date**

LibrePGP v5 - v3 Signatures Aliasing



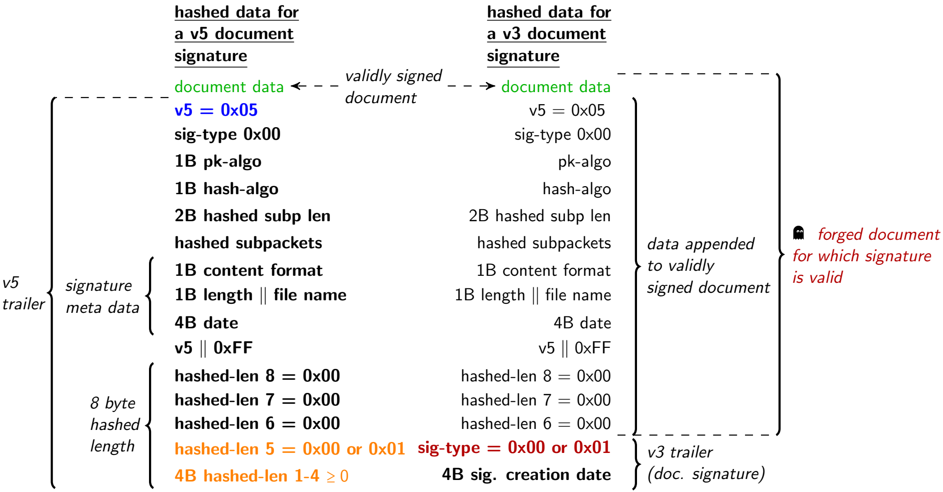
		hashed data for a v5 document signature		**hashed data for a v3 document signature**	
		document data	*validly signed document*	document data	
v5 trailer		**v5 = 0x05**		v5 = 0x05	
		sig-type 0x00		sig-type 0x00	
		1B pk-algo		pk-algo	
		1B hash-algo		hash-algo	
		2B hashed subp len		2B hashed subp len	
		hashed subpackets		hashed subpackets	
	signature meta data	**1B content format**		1B content format	
		1B length || file name		1B length || file name	
		4B date		4B date	
		v5 || 0xFF		v5 || 0xFF	
	8 byte hashed length	**hashed-len 8 = 0x00**		hashed-len 8 = 0x00	
		hashed-len 7 = 0x00		hashed-len 7 = 0x00	
		hashed-len 6 = 0x00		hashed-len 6 = 0x00	
		hashed-len 5 = 0x00 or 0x01		**sig-type = 0x00 or 0x01**	*v3 trailer (doc. signature)*
		4B hashed-len 1-4 ≥ 0		**4B sig. creation date**	

LibrePGP v5 - v3 Signatures Aliasing



		hashed data for a v5 document signature		hashed data for a v3 document signature	
		document data	validly signed document	document data	
v5 trailer		**v5 = 0x05**		v5 = 0x05	*data appended to validly signed document*
		sig-type 0x00		sig-type 0x00	
		1B pk-algo		pk-algo	
		1B hash-algo		hash-algo	
		2B hashed subp len		2B hashed subp len	
		hashed subpackets		hashed subpackets	
	signature meta data	**1B content format**		1B content format	
		1B length ‖ file name		1B length ‖ file name	
		4B date		4B date	
		v5 ‖ 0xFF		v5 ‖ 0xFF	
	8 byte hashed length	**hashed-len 8 = 0x00**		hashed-len 8 = 0x00	
		hashed-len 7 = 0x00		hashed-len 7 = 0x00	
		hashed-len 6 = 0x00		hashed-len 6 = 0x00	
		hashed-len 5 = 0x00 or 0x01		**sig-type = 0x00 or 0x01**	*v3 trailer (doc. signature)*
		4B hashed-len 1-4 ≥ 0		**4B sig. creation date**	

LibrePGP v5 - v3 Signatures Aliasing



hashed data for a v5 document signature	**hashed data for a v3 document signature**
document data	document data
v5 = 0x05	v5 = 0x05
sig-type 0x00	sig-type 0x00
1B pk-algo	pk-algo
1B hash-algo	hash-algo
2B hashed subp len	2B hashed subp len
hashed subpackets	hashed subpackets
1B content format	1B content format
1B length ‖ file name	1B length ‖ file name
4B date	4B date
v5 ‖ 0xFF	v5 ‖ 0xFF
hashed-len 8 = 0x00	hashed-len 8 = 0x00
hashed-len 7 = 0x00	hashed-len 7 = 0x00
hashed-len 6 = 0x00	hashed-len 6 = 0x00
hashed-len 5 = 0x00 or 0x01	**sig-type = 0x00 or 0x01**
4B hashed-len 1-4 ≥ 0	**4B sig. creation date**

validly signed document
v5 trailer
signature meta data
8 byte hashed length
data appended to validly signed document
v3 trailer (doc. signature)
👻 *forged document for which signature is valid*

LibrePGP v5 - v3 Signatures Aliasing

- v5 → v3 aliasing possible
  - v3 signatures may still be verified
- v3 → v5 aliasing not possible:
  - creation of v3 signatures disallowed (OpenPGP, LibrePGP)
  - signature creation time would have to match the hashed length
- also other signatures types (e.g., signature over key) can be interpreted as v3 document signature

LibrePGP v5 - v3 Signatures Aliasing

- v5 → v3 aliasing possible
  - v3 signatures may still be verified
- v3 → v5 aliasing not possible:
  - creation of v3 signatures disallowed (OpenPGP, LibrePGP)
  - signature creation time would have to match the hashed length
- also other signatures types (e.g., signature over key) can be interpreted as v3 document signature

LibrePGP v5 - v3 Signatures Aliasing

- v5 → v3 aliasing possible
  - v3 signatures may still be verified
- v3 → v5 aliasing not possible:
  - creation of v3 signatures disallowed (OpenPGP, LibrePGP)
  - signature creation time would have to match the hashed length
- also other signatures types (e.g., signature over key) can be interpreted as v3 document signature

LibrePGP v5 - v3 Signatures Aliasing

- v5 → v3 aliasing possible
  - v3 signatures may still be verified
- v3 → v5 aliasing not possible:
  - creation of v3 signatures disallowed (OpenPGP, LibrePGP)
  - signature creation time would have to match the hashed length
- also other signatures types (e.g., signature over key) can be interpreted as v3 document signature

LibrePGP v5 - v3 Signatures Aliasing

- v5 → v3 aliasing possible
  - v3 signatures may still be verified
- v3 → v5 aliasing not possible:
  - creation of v3 signatures disallowed (OpenPGP, LibrePGP)
  - signature creation time would have to match the hashed length
- also other signatures types (e.g., signature over key) can be interpreted as v3 document signature

LibrePGP v5 - v3 Signatures Aliasing

- v5 → v3 aliasing possible
  - v3 signatures may still be verified
- v3 → v5 aliasing not possible:
  - creation of v3 signatures disallowed (OpenPGP, LibrePGP)
  - signature creation time would have to match the hashed length
- also other signatures types (e.g., signature over key) can be interpreted as v3 document signature

Countermeasure in OpenPGP RFC 9580

- crypto-refresh had the same problem as now v5
- was reported by Demi Marie Obenour in GitHub
- fix in RFC 9580:
  - revert from 8-byte hashed length to 4-byte
  - prohibit signature type 0xFF

v3:	...	...	...	...	...	...**msg**	sig-type	4B crea. time
v4:	...	...	...	...	**hashed-subp bd**	0x04	0xFF	4B hashed length
v6:	...	...	...	...	**hashed-subp bd**	0x06	0xFF	4B hashed length
v5:	...	0x05	0x0FF	hl8	hl7	hl6	$\underbrace{\text{hl5}}_{0 \text{ or } 1}$	low 4B of hashed len.

Countermeasure in OpenPGP RFC 9580

- crypto-refresh had the same problem as now v5
- was reported by Demi Marie Obenour in GitHub
- fix in RFC 9580:
  - revert from 8-byte hashed length to 4-byte
  - prohibit signature type 0xFF

v3:	...	...	...	...	...	...**msg**	sig-type	4B crea. time
v4:	...	...	...	...	**hashed-subp bd**	0x04	0xFF	4B hashed length
v6:	...	...	...	...	**hashed-subp bd**	0x06	0xFF	4B hashed length
v5:	...	0x05	0x0FF	hl8	hl7	hl6	$\underbrace{\text{hl5}}_{0 \text{ or } 1}$	low 4B of hashed len.

Countermeasure in OpenPGP RFC 9580

- crypto-refresh had the same problem as now v5
- was reported by Demi Marie Obenour in GitHub
- fix in RFC 9580:
  - revert from 8-byte hashed length to 4-byte
  - prohibit signature type 0xFF

v3:	...	...	...	...	...	...**msg**	sig-type	4B crea. time
v4:	...	...	...	...	**hashed-subp bd**	0x04	0xFF	4B hashed length
v6:	...	...	...	...	**hashed-subp bd**	0x06	0xFF	4B hashed length
v5:	...	0x05	0x0FF	hl8	hl7	hl6	$\underbrace{\text{hl5}}_{0 \text{ or } 1}$	low 4B of hashed len.

Countermeasure in OpenPGP RFC 9580

- crypto-refresh had the same problem as now v5
- was reported by Demi Marie Obenour in GitHub
- fix in RFC 9580:
  - revert from 8-byte hashed length to 4-byte
  - prohibit signature type 0xFF

v3:	...	...	...	...	...	...**msg**	sig-type	4B crea. time
v4:	...	...	...	...	**hashed-subp bd**	0x04	0xFF	4B hashed length
v6:	...	...	...	...	**hashed-subp bd**	0x06	0xFF	4B hashed length
v5:	...	0x05	0x0FF	hl8	hl7	hl6	$\underbrace{\text{hl5}}_{\text{0 or 1}}$	low 4B of hashed len.

Countermeasure in OpenPGP RFC 9580

- crypto-refresh had the same problem as now v5
- was reported by Demi Marie Obenour in GitHub
- fix in RFC 9580:
  - revert from 8-byte hashed length to 4-byte
  - prohibit signature type 0xFF

v3:	...	**...**	**...**	**...**	**...**	**...msg**	sig-type	4B crea. time
v4:	...	**...**	**...**	**...**	**hashed-subp bd**	0x04	0xFF	4B hashed length
v6:	...	**...**	**...**	**...**	**hashed-subp bd**	0x06	0xFF	4B hashed length
v5:	...	0x05	0x0FF	hl8	hl7	hl6	$\underbrace{\text{hl5}}_{0 \text{ or } 1}$	low 4B of hashed len.

Introduction

CMS: Old EUF-CMA Violations

CMS: (generalized) EUF-CMA Problem in Current Proposal for Composite Signatures

OpenPGP: Natural Strong Non-Separability of Composite Signatures

LibrePGP: EUF-CMA Violation through Signature Version Aliasing

OpenPGP: Unsigned Packet Meta Data

Other Aspects of Post Quantum Signatures in Protocols

Unsigned Packet Meta Data in RFC 9580

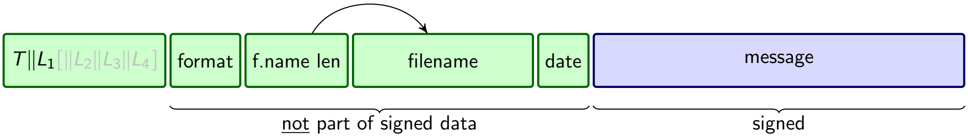
- v4 signatures: Literal Data Packet header not signed
- RFC 9580: no change for v6

Unsigned Packet Meta Data in RFC 9580

- v4 signatures: Literal Data Packet header not signed
- RFC 9580: no change for v6

Unsigned Packet Meta Data in RFC 9580

- v4 signatures: Literal Data Packet header not signed
- RFC 9580: no change for v6



$T||L_1[||L_2||L_3||L_4]$

format

f.name len

filename

date

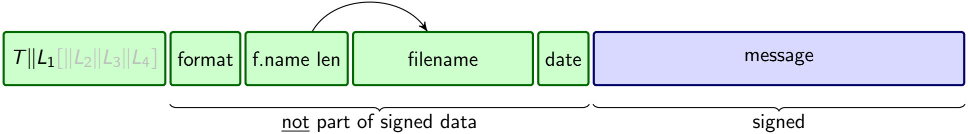
message

not part of signed data

signed

Unsigned Packet Meta Data in RFC 9580

- v4 signatures: Literal Data Packet header not signed
- RFC 9580: no change for v6



- LibrePGP: header added to signed data for v5 signatures

Introduction

CMS: Old EUF-CMA Violations

CMS: (generalized) EUF-CMA Problem in Current Proposal for Composite Signatures

OpenPGP: Natural Strong Non-Separability of Composite Signatures

LibrePGP: EUF-CMA Violation through Signature Version Aliasing

OpenPGP: Unsigned Packet Meta Data

Other Aspects of Post Quantum Signatures in Protocols

ML-DSA and SLH-DSA security assumptions

- neither scheme follows hash-then-sign paradigm
  - internal hashing
  - internal-hash(salt || len(ctx) || ctx || message)

ML-DSA and SLH-DSA security assumptions

- neither scheme follows hash-then-sign paradigm
    - internal hashing
    - internal-hash(salt || len(ctx) || ctx || message)

ML-DSA and SLH-DSA security assumptions

- neither scheme follows hash-then-sign paradigm
  - internal hashing
  - internal-hash(salt || len(ctx) || ctx || message)

ML-DSA Design

- $\mu \leftarrow H(H(\text{public-key})||M)$ $\quad$ *// contributes to collision resistance*
- uses per-key constant salt
- is one-pass over the message

ML-DSA Design

- $\mu \leftarrow H(H(\text{public-key})||M)$ *// contributes to collision resistance*
- uses per-key constant salt
- is one-pass over the message

ML-DSA Design

- $\mu \leftarrow H(H(\text{public-key})||M)$ // *contributes to collision resistance*
- uses per-key constant salt
- is one-pass over the message

SLH-DSA Design

MTG

- choice
  - opt_rand ← rnd
  - opt_rand ← PK.seed
- $R \leftarrow \mathrm{PRF}_{\mathrm{msg}}(SK.prf, \mathrm{opt\_rand}, \mathrm{PK.root}, M)$ *// M to hedge against RNG failure*
- $\mathrm{digest} \leftarrow H_{\mathrm{msg}}(R, \mathrm{PK.seed}, \mathrm{PK.root}, M)$ *// improved collision resistance*
- compute signature based on *digest* . . .

SLH-DSA Design

- choice
  - opt_rand ← rnd
  - opt_rand ← PK.seed
- $R \leftarrow \mathrm{PRF}_{\mathrm{msg}}(SK.prf, \mathrm{opt\_rand}, \mathrm{PK.root}, M)$ *// M to hedge against RNG failure*
- $\mathrm{digest} \leftarrow H_{\mathrm{msg}}(R, \mathrm{PK.seed}, \mathrm{PK.root}, M)$ *// improved collision resistance*
- compute signature based on *digest* . . .

SLH-DSA Design

- choice
  - opt_rand ← rnd
  - opt_rand ← PK.seed
- $R \leftarrow \mathrm{PRF}_{\mathrm{msg}}(SK.prf, \mathrm{opt\_rand}, \mathrm{PK.root}, M)$ *// M to hedge against RNG failure*
- $\mathrm{digest} \leftarrow H_{\mathrm{msg}}(R, \mathrm{PK.seed}, \mathrm{PK.root}, M)$ *// improved collision resistance*
- compute signature based on *digest* . . .

SLH-DSA Design

- choice
  - opt_rand ← rnd
  - opt_rand ← PK.seed
- $R \leftarrow \mathrm{PRF}_{\mathrm{msg}}(SK.prf, \mathrm{opt\_rand}, \mathrm{PK.root}, M)$ *// M to hedge against RNG failure*
- $\mathrm{digest} \leftarrow H_{\mathrm{msg}}(R, \mathrm{PK.seed}, \mathrm{PK.root}, M)$ *// improved collision resistance*
- compute signature based on *digest* ...

SLH-DSA Design

- choice
  - opt_rand ← rnd
  - opt_rand ← PK.seed
- $R \leftarrow \mathrm{PRF_{msg}}(SK.prf, \text{opt\_rand}, \mathrm{PK.root}, M)$ *// M to hedge against RNG failure*
- $\text{digest} \leftarrow H_{\mathrm{msg}}(R, \mathrm{PK.seed}, \mathrm{PK.root}, M)$ *// improved collision resistance*
- compute signature based on *digest* . . .

SLH-DSA Design

- choice
  - opt_rand ← rnd
  - opt_rand ← PK.seed
- $R \leftarrow \mathrm{PRF}_{\mathrm{msg}}(SK.prf, \mathrm{opt\_rand}, \mathrm{PK.root}, M)$ *// M to hedge against RNG failure*
- $\mathrm{digest} \leftarrow H_{\mathrm{msg}}(R, \mathrm{PK.seed}, \mathrm{PK.root}, M)$ *// improved collision resistance*
- compute signature based on *digest* . . .

SLH-DSA Design

- choice
    - opt_rand ← rnd
    - opt_rand ← PK.seed
- $R \leftarrow \mathrm{PRF}_{\mathrm{msg}}(SK.\mathit{prf}, \mathrm{opt\_rand}, \mathrm{PK.root}, M)$ *// M to hedge against RNG failure*
- $\mathrm{digest} \leftarrow H_{\mathrm{msg}}(R, \mathrm{PK.seed}, \mathrm{PK.root}, M)$ *// improved collision resistance*
- compute signature based on *digest* . . .

⟶ pure SLH-DSA: signing, verification not on-line, no streaming

SLH-DSA Design

- choice
  - opt_rand ← rnd
  - opt_rand ← PK.seed
- $R \leftarrow \mathrm{PRF}_{\mathrm{msg}}(SK.prf, \mathrm{opt\_rand}, \mathrm{PK.root}, M)$ *// M to hedge against RNG failure*
- $\mathrm{digest} \leftarrow H_{\mathrm{msg}}(R, \mathrm{PK.seed}, \mathrm{PK.root}, M)$ *// improved collision resistance*
- compute signature based on *digest* . . .

⟶ pure SLH-DSA: signing, verification not on-line, no streaming

⟶ need to resort to pre-hash variant for memory efficiency

SLH-DSA Design

- choice
  - opt_rand ← rnd
  - opt_rand ← PK.seed
- $R \leftarrow \mathrm{PRF}_{\mathrm{msg}}(SK.prf, \mathrm{opt\_rand}, \mathrm{PK.root}, M)$ // *M to hedge against RNG failure*
- $\mathrm{digest} \leftarrow H_{\mathrm{msg}}(R, \mathrm{PK.seed}, \mathrm{PK.root}, M)$ // *improved collision resistance*
- compute signature based on *digest* ...

⟶ pure SLH-DSA: signing, verification not on-line, no streaming

⟶ need to resort to pre-hash variant for memory efficiency

⟶ no increased collision resistance at all

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn't define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn't define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with "pure" variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn't define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn't define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn't define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn't define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn't define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn’t define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn’t define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn’t define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn’t define PQC signatures yet
- CMS:
  - no random salt on protocol level

ML-DSA and SLH-DSA in OpenPGP

- OpenPGP: bound to hash-then-sign
  - compute hash
  - sign the digest with “pure” variant (current proposal)
  - → no two-pass-problem with SLH-DSA
- RFC 9580: v6 signatures
  - prefixes random salt on protocol level
  - → no dependence on collision resistance of hash function
- LibrePGP: v5 signatures
  - no random salt on protocol level
  - doesn’t define PQC signatures yet
- CMS:
  - no random salt on protocol level

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - "fixes" the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - “fixes” the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
    - compute hash in the application
    - provide hash value to the signature function
- pre-hash variant ensures:
    - domain separation from pure variant
    - prevents digest substitution attacks
        - this attack requires finding 2nd preimage
        - “fixes” the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
    - OpenPGP already has hash-algo in signature meta data → domain separation
    - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - "fixes" the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - "fixes" the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - “fixes” the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - "fixes" the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - “fixes” the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - “fixes” the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - “fixes” the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Pre-hash variant

- FIPS 204, 205 for ML-DSA and SLH-DSA define pre-hash-variant
  - compute hash in the application
  - provide hash value to the signature function
- pre-hash variant ensures:
  - domain separation from pure variant
  - prevents digest substitution attacks
    - this attack requires finding 2nd preimage
    - "fixes" the pre-hash hash algorithm with the strength of the internal hash
- neither is needed for OpenPGP:
  - OpenPGP already has hash-algo in signature meta data → domain separation
  - PQC signature draft defines only single hash algorithm → no digest substitution

Conclusion

- old and new (generalized) EUF-CMA problems in CMS
- also in LibrePGP
- problems with signature meta data
  - CMS: missing totally
  - LibrePGP: not uniquely parseable
- CMS: composite signatures give rise to “signature combiner”
  - currently with (generalized) EUF-CMA problem

Conclusion

- old and new (generalized) EUF-CMA problems in CMS
- also in LibrePGP
- problems with signature meta data
  - CMS: missing totally
  - LibrePGP: not uniquely parseable
- CMS: composite signatures give rise to “signature combiner”
  - currently with (generalized) EUF-CMA problem

Conclusion

- old and new (generalized) EUF-CMA problems in CMS
- also in LibrePGP
- problems with signature meta data
  - CMS: missing totally
  - LibrePGP: not uniquely parseable
- CMS: composite signatures give rise to “signature combiner”
  - currently with (generalized) EUF-CMA problem

Conclusion

- old and new (generalized) EUF-CMA problems in CMS
- also in LibrePGP
- problems with signature meta data
  - CMS: missing totally
  - LibrePGP: not uniquely parseable
- CMS: composite signatures give rise to “signature combiner”
  - currently with (generalized) EUF-CMA problem

Conclusion

- old and new (generalized) EUF-CMA problems in CMS
- also in LibrePGP
- problems with signature meta data
  - CMS: missing totally
  - LibrePGP: not uniquely parseable
- CMS: composite signatures give rise to “signature combiner”
  - currently with (generalized) EUF-CMA problem

Conclusion

- old and new (generalized) EUF-CMA problems in CMS
- also in LibrePGP
- problems with signature meta data
  - CMS: missing totally
  - LibrePGP: not uniquely parseable
- CMS: composite signatures give rise to “signature combiner”
  - currently with (generalized) EUF-CMA problem

Conclusion

- old and new (generalized) EUF-CMA problems in CMS
- also in LibrePGP
- problems with signature meta data
  - CMS: missing totally
  - LibrePGP: not uniquely parseable
- CMS: composite signatures give rise to “signature combiner”
  - currently with (generalized) EUF-CMA problem

Thank you for your attention

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

MTG AG
www.mtg.de