信頼される安心を、社会へ。

ISOC-JP&JPNIC IETF120報告会

AEAD to non-AEAD Downgrade AttackとCOSEでの対策案

セコム株式会社　IS研究所
髙山献

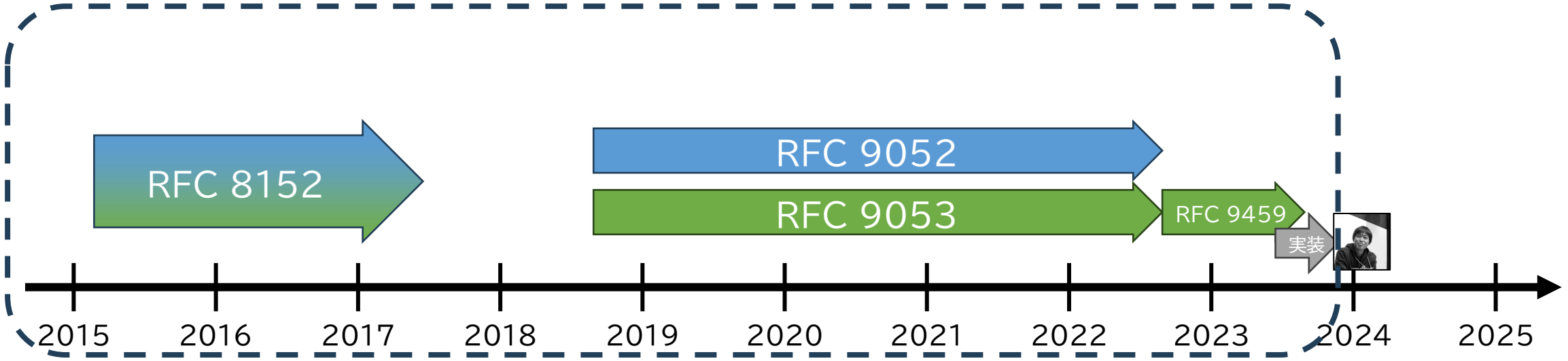
自己紹介

- 名前：髙山 献（たかやま けん）
- 経歴
  - 1991年　出生
  - 2019年　東京農工大学大学院卒業
    - 専門はOSやVMMなどシステムソフトウェア
  - 2019年　セコム入社　IS研究所 所属
  - 2020年　機密情報保護とデータ利活用を行うための基盤技術を研究
    - キーワード：TEE、Confidential Computing、…
  - 2020年　標準化団体IETFの109回会議出席、以降年3回参加
    - 下書き段階の標準を検証する立場、執筆する立場で参加
    - 主にTEEP, SUIT, COSEといったセキュリティ系のグループで活動中

目次

- COSEの話
  - IETFで標準化されている、暗号ペイロード格納フォーマット
  - 暗号化アルゴリズムにはAES-GCMやAES-CTRなどをサポート
- AEAD-to-CBC Downgrade Attack関連の話
  - 受信者Bobが特定の挙動をすると、平文が漏れる
  - 受信者Bobが検査を漏らすと、改ざんされた平文を受け取ってしまう
- 対策の話
  - COSEライブラリの工夫で、意図せずnon-AEADで復号しないよう変更
  - 送信者Aliceが作る暗号ペイロードを工夫して、攻撃成立条件をつぶす

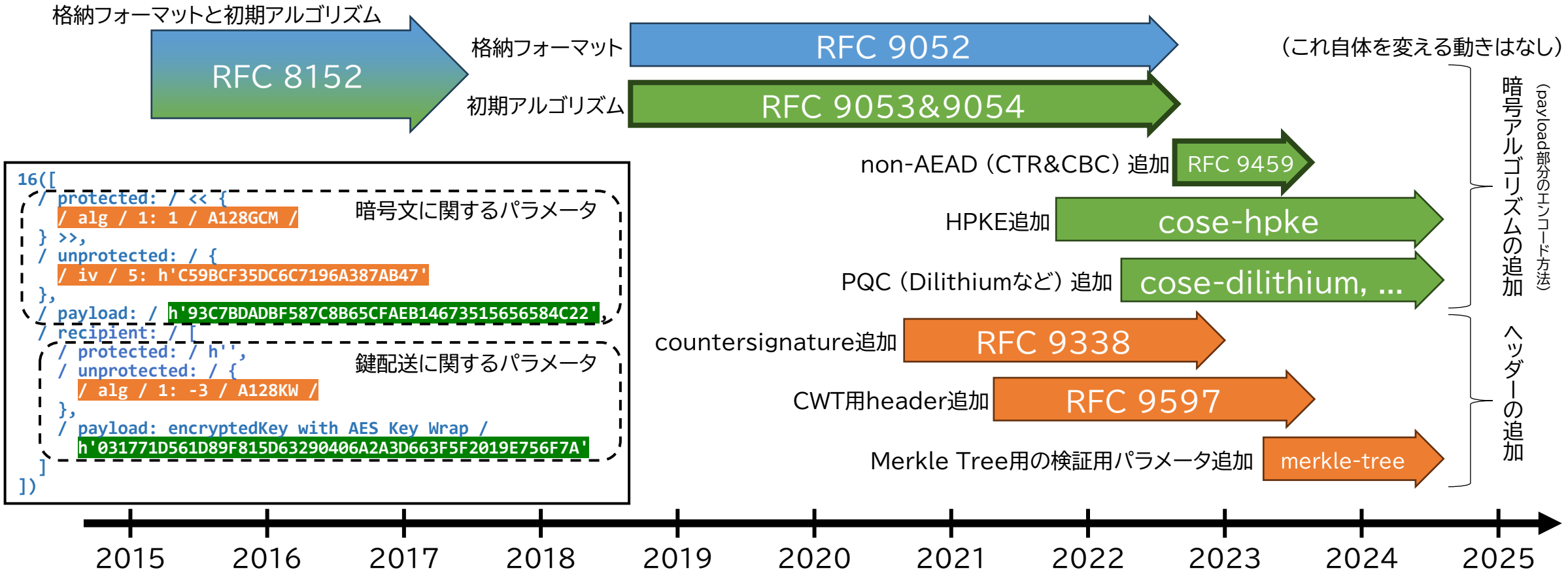
COSEの話



COSE (CBOR Object Signing and Encryption)

信頼される安心を、社会へ。 SECOM

- IETFが2015年頃から標準化している暗号ペイロード格納フォーマット
  - RFC 8152の再編以降、暗号アルゴリズムやヘッダーの追加が継続中



IETF120でのドキュメント状況

- RFC化間近
  - COSE Key Thumbprint: COSE_KeyのIDをハッシュ値から導出
- WG Last Call間近
  - TSA TST Header: COSEメッセージにタイムスタンプ付与
  - Merkle Tree Proof: 受領時点のハッシュツリー状態を格納

サプライチェーン管理を扱うSCITT WGから移動してきたドキュメント

- 熟成中

(PQ/T = Post Quantum Traditional Hybrid)

  - COSE HPKE, PQ/T: HPKEと、PQなKEMも含めた暗号スート追加
  - COSE ML-KEM, ML-DSA, SLH-DSA , *FN-DSA*:
    - PQなCRYSTALS-{Kyber, Dilithium}, SPHINCS+, *FALCON*追加

COSEでもPQ対応

- 新規Internet Draft
  - **CEK HKDF**: AEAD to non-AEADへの対策 ←髙山共著者

セキュリティ強化

詳細はこちらから：https://datatracker.ietf.org/wg/cose/documents/

暗号文のCOSEへの格納方法

- 暗号化アルゴリズム: AES-GCM (AEAD), AES-CTR (non-AEAD)など
- 鍵配送アルゴリズム: AES-KW, ECDH+AES-KWなど
  - 将来的にHPKEやML-KEMを含む暗号スートも追加される見込み

AES-KW+AES-GCMの例

```
96([
  / protected: / << {
    / alg / 1: 1 / A128GCM /
  } >>,
  / unprotected: / {
    / iv / 5: h'C59BCF35DC6C7196A387AB47'
  },
  h'93C7BDADBF587C8B65CFAEB1467351565658 4C22',
  / recipients: / [
    [
      / protected: / h'',
      / unprotected: / {
        / alg / 1: -3 / A128KW /
      },
      / encryptedKey with AES Key Wrap /
      h'031771D561D89F815D63290406A2A3D6 63F5F2019E756F7A'
    ]
  ]
])
```

暗号文　GCM用タグ(完全性＋認証)

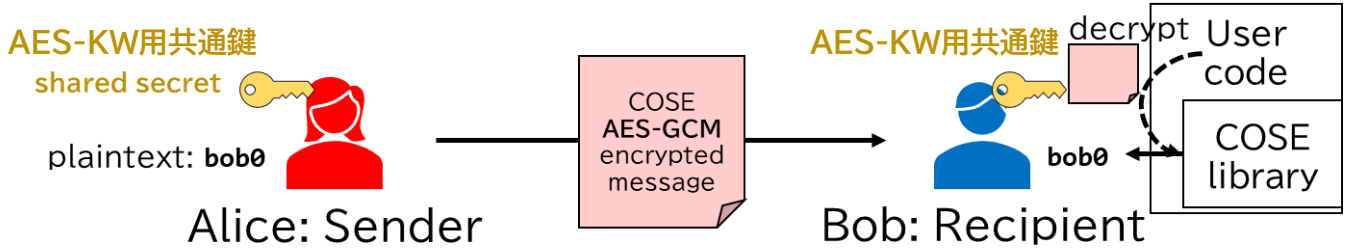
wrapされた暗号鍵　　AES-KW用タグ

【事前に共有するAES Key Wrap用共通鍵】
h'849B57219DAE48dE646D07DBB533566E'

【Aliceが生成しBobが得るAES-GCM用共通鍵】
h'E6620CB79D6338F97C544D80B614A160'

【平文ペイロード】
'bob0'

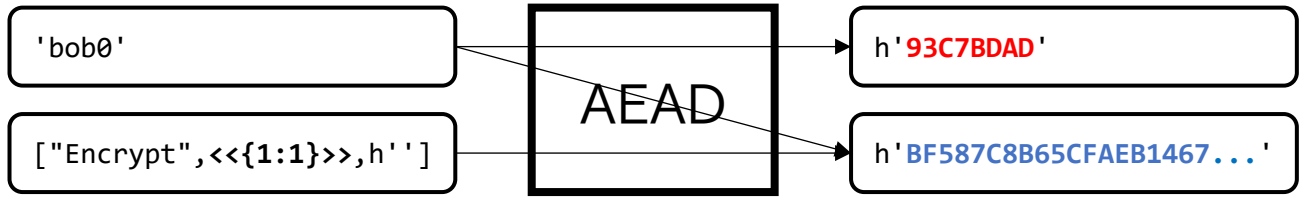
【暗号化済みペイロード】
h'93C7BDAD'



AEADとは

- Authenticated Encryption and Associated Data
  - AES-GCM、AES-CCMなど（RFC 9053で定義）
  - タグの検証対象に暗号文以外のデータ(alg=A128GCM等)を含められる

➢いずれかが改ざんされても、
Bobは検証することが可能



AES-GCMの
128bit鍵で
暗号文を作ります

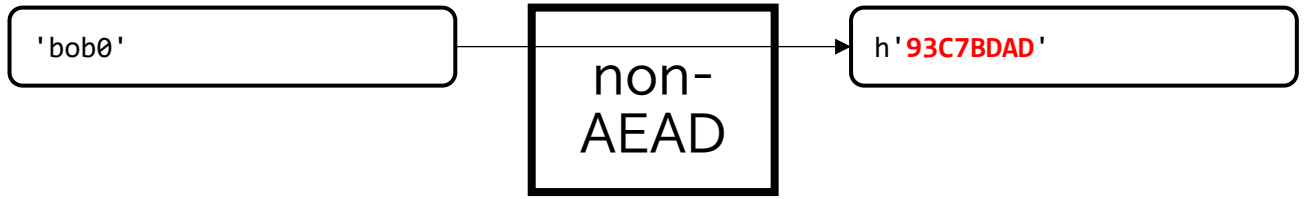
送信者Aliceが作るCOSEメッセージ

AES-KW+AES-GCMの例

```
96([
  / protected: / << {
    / alg / 1: 1 / A128GCM /
  } >>,
  / unprotected: / {
    / iv / 5: h'C59BCF35DC6C7196A387AB47'
  }, 暗号文 GCM用タグ(完全性＋認証)
  h'93C7BDADBF587C8B65CFAEB14673515656584C22',
  / recipients: / [
    [
      / protected: / h'',
      / unprotected: / {
        / alg / 1: -3 / A128KW /
      },
      / encryptedKey with AES Key Wrap /
      h'031771D561D89F815D63290406A2A3D663F5F2019E756F7A'
    ]
  ]
])
```

non-AEADとは

- タグなどによる完全性や認証の検査が無い暗号アルゴリズム
  - AES-CTR、AES-CBCなど（RFC 9459で定義）
  - 受信者Bobは、復号した平文、暗号パラメータ(alg=A128CTR)について、送信者が送ろうとしたものかわからない
  - ➢特別な注意を払った場合のみ使うべき
    - 電子署名などと組み合わせろ（RFC9459）



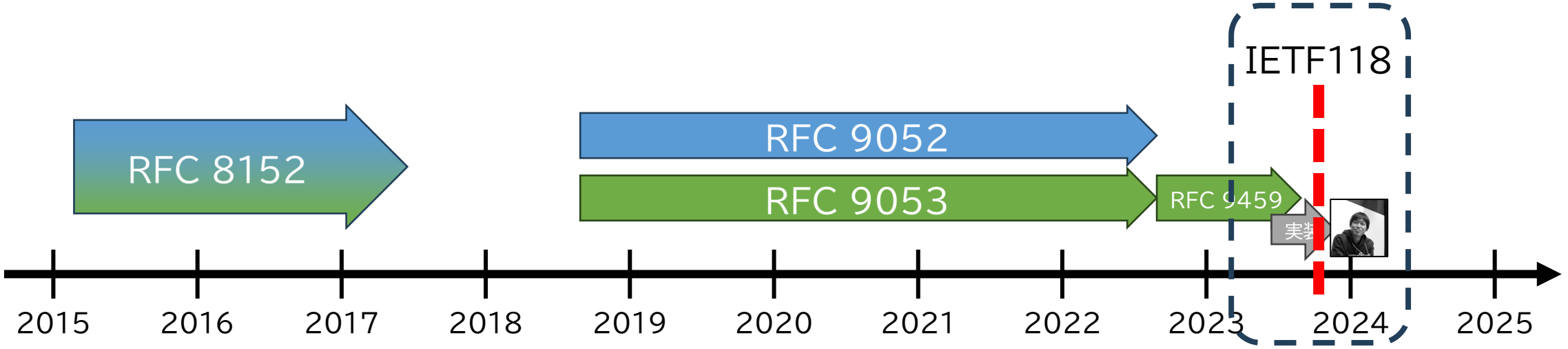
送信者Aliceが作るCOSEメッセージ

IETF118の頃に、髙山がCOSEライブラリに機能追加

AES-KW+AES-CTRの例

```
96([
  / protected: / h'',

  / unprotected: / {
    / alg / 1: -65534 / A128CTR /,
    / iv / 5: h'C59BCF35DC6C7196A387AB4700000002'
  },暗号文
  h'93C7BDAD',
  / recipients: / [
    [
      / protected: / h'',
      / unprotected: / {
        / alg / 1: -3 / A128KW /
      },
      / encryptedKey with AES Key Wrap /
      h'031771D561D89F815D63290406A2A3D663F5F2019E756F7A'
    ]
  ]
])
```

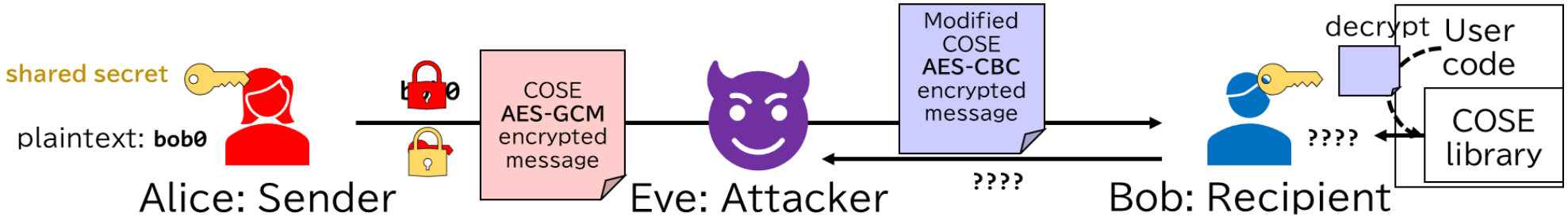
AEAD-to-CBCの話



AEAD-to-CBC Downgrade Attack

- IETF118 LAMPS WGにて報告された（2023年11月）
  - by Johannes Roth and Falko Strenzke
- 攻撃成功条件と起こることを大雑把に説明すると
  - Aliceがテンプレートのある平文ブロックPに対してAES-GCMの暗号文Cを作り
  - AES-CBCにも対応したBobが暗号文を復号し、”異変”を感じた復号文を送り返す（Decryption Oracleである）場合に
  - 平文テンプレートp_jを知っている**攻撃者Eve**が以下のペイロードをBobに送ると
    - $C \oplus p_0 || C \oplus p_1 || \cdots || C \oplus p_t || \cdots || C \oplus p_n$
  - Aliceが送った平文の一部を、Eveは推測できる

C：Eveが狙っている暗号文ブロック
p_t：Eveが的中させたAliceの平文P



https://datatracker.ietf.org/meeting/118/materials/slides-118-lamps-attack-against-aead-in-cms-00

COSEではどうなのか？

- 秘密が漏れるAEAD-to-CBCは、COSEではそこまで心配がない
  - Decryption Oracleになる受信者は一般的ではない
    （元の攻撃対象は暗号化メールのS/MIMEだったため、より起こりえた）
  - CBCのパディングチェックが必須で、ここで弾かれる可能性もある
  - S/MIME v4.1ではCBC実装必須、COSEではCBC”廃止済み”であるため
    - ”廃止済み”として登録したのは、意図しない利用を防ぐため
    - 一部条件下でユースケースがあるため、CBCに対応するCOSEライブラリも存在する
- 攻撃者が意図した暗号文に書き換えやすいAEAD-to-CTRには要注意
  - Decryption Oracleでなくても、受信者は改変された平文を受け取ってしまう
  - GCMやCCMは内部でCTRを使っていることが要因で、
    攻撃者は意図して細工した暗号ペイロードを作りやすい
  - ”廃止済み”CTRに対応するCOSEライブラリがあると、簡単に成立しうる

実はさっき紹介したCOSEバイナリは

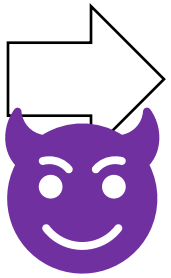
（＝AEADからnon-AEADにDowngrade）

- Eveが簡単にGCM to CTRに　変換　できることを示している
  - アルゴリズムIDを変えて、タグを消して、IVにh'00000002'を足すだけ
  - この時点では、Aliceが送ろうとしたのと同じ'bob0'が復号される

```
96([
  / protected: / << {
    / alg / 1: 1 / A128GCM /
  } >>,
  / unprotected: / {
    / iv / 5: h'C59BCF35DC6C7196A387AB47'
  },
  h'93C7BDADBF587C8B65CFAEB14673515656584C22', b'0
  / recipients: / [
    [
      / protected: / h'',
      / unprotected: / {
        / alg / 1: -3 / A128KW /
      },
      / encryptedKey with AES Key Wrap /
      h'031771D561D89F815D63290406A2A3D663F5F2019E756F7A'
    ]
  ]
])
```

暗号文　GCM用タグ（完全性＋認証）

wrapされた暗号鍵　　AES-KW用タグ



```
96([
  / protected: / h'',
  / unprotected: / {
    / alg / 1: -65534 / A128CTR /,
    / iv / 5: h'C59BCF35DC6C7196A387AB4700000002'
  },
  h'93C7BDADBF587C8B65CFAEB14673515656584C22',
  / recipients: / [
    [
      / protected: / h'',
      / unprotected: / {
        / alg / 1: -3 / A128KW /
      },
      / encryptedKey with AES Key Wrap /
      h'031771D561D89F815D63290406A2A3D663F5F2019E756F7A'
    ]
  ]
])
```

暗号アルゴリズムを変える

暗号文　　IVを変える

GCM用タグを消す

wrapされた暗号鍵　　AES-KW用タグ

CTRの暗号文には意図的な改ざんをしやすい

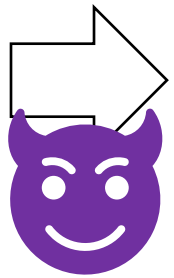
信頼される安心を、社会へ。 SECOM

- 暗号文のビットフリップが、復号された平文のビットフリップになる
  - AES-CTRにはMalleabilityがあるため
  - 例）Aliceの平文'bob0'、暗号文の最終ビットフリップでBobの復号文は'bob1'

```
96([
  / protected: / << {
    / alg / 1: 1 / A128GCM /
  } >>,
  / unprotected: / {
    / iv / 5: h'C59BCF35DC6C7196A387AB47'
  },
  h'93C7BDADBF587C8B65CFAEB14673515656584C22', b'0
  / recipients: / [
    [
      / protected: / h'',
      / unprotected: / {
        / alg / 1: -3 / A128KW /
      },
      / encryptedKey with AES Key Wrap /
      h'031771D561D89F815D63290406A2A3D663F5F2019E756F7A'
    ]
  ]
])
```

暗号文　GCM用タグ（完全性＋認証）

wrapされた暗号鍵　AES-KW用タグ



```
96([
  / protected: / h'',
  / unprotected: / {
    / alg / 1: -65534 / A128CTR /,
    / iv / 5: h'C59BCF35DC6C7196A387AB4700000002'
  },
  h'93C7BDAC[illegible]',
  / recipients: / [
    [
      / protected: / h'',
      / unprotected: / {
        / alg / 1: -3 / A128KW /
      },
      / encryptedKey with AES Key Wrap /
      h'031771D561D89F815D63290406A2A3D663F5F2019E756F7A'
    ]
  ]
])
```

暗号アルゴリズムを変える

暗号文（ビットフリップ済み）

IVを変える

GCM用タグを消す

wrapされた暗号鍵　AES-KW用タグ

受信者が意図せずCTRを使ってしまう実装例

信頼される安心を、社会へ。 SECOM

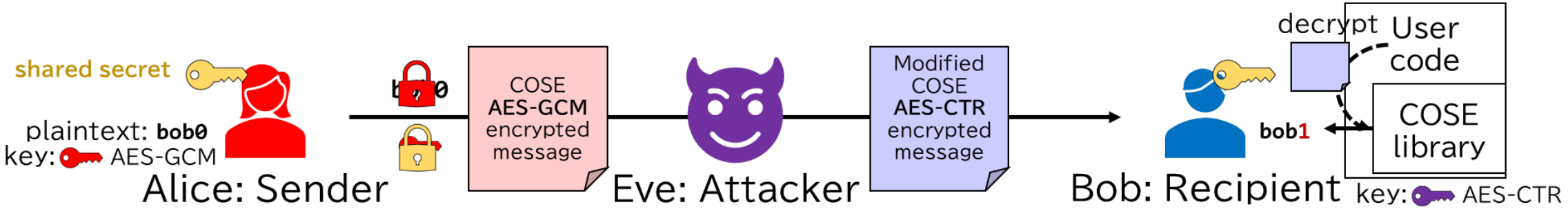
ライブラリがCTRもサポートしていて、受容アルゴリズムに制限がないと

```
# pre-shared KEK for A128KW
wrapping_key = new COSE_Key(
  kty=Symmetric, k=h'849B...', alg=A128KW)

decrypted_plaintext = COSE_Encrypt.decrypt(
  cose=alice_cose_message, key=wrapping_key)
# decrypted_plaintext = 'bob0'
```

受信者BobはAES-KWに使われる鍵を指定。
送信者AliceのCOSEメッセージをパースすると、
ライブラリ内でAES-GCMの鍵が導出され復号される。

```
# pre-shared KEK for A128KW
wrapping_key = new COSE_Key(
  kty=Symmetric, k=h'849B...', alg=A128KW)

decrypted_plaintext = COSE_Encrypt.decrypt(
  cose=eve_cose_message, key=wrapping_key)
# decrypted_plaintext = 'bob1'
```

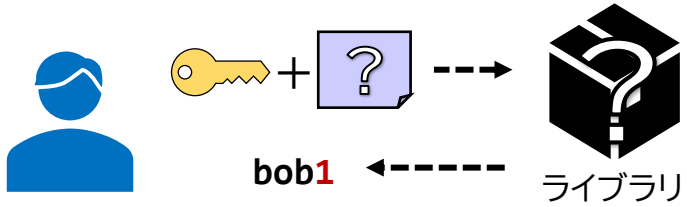
受信者BobはAES-KWに使われる鍵を指定。
攻撃者EveのCOSEメッセージをパースすると、
意図せずAES-CTRの鍵が導出され復号される。



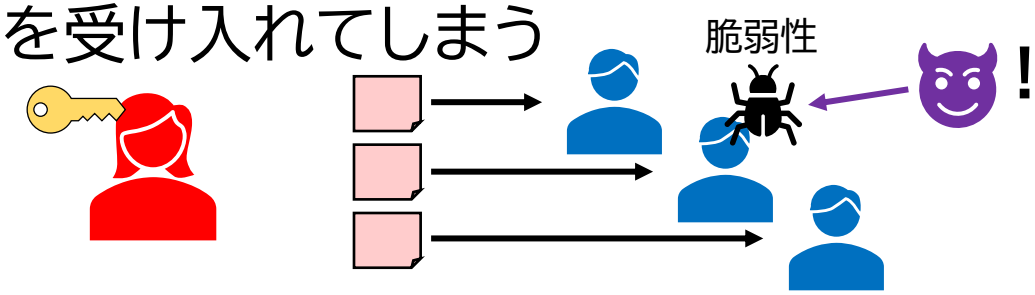
この問題の難しいところ

A) 受信者BobがCOSEライブラリを使う場合はチェックが漏れやすい
- COSEバイナリの中身を見たくないからライブラリを使うので
  暗号化アルゴリズムがnon-AEADかどうか意識しないかもしれない
- そのライブラリがnon-AEADをサポートする場合に、RFC9459を読み、
  non-AEADで何が起こりうるかまで知っていることには期待できない

B) 送信者Aliceにできる対策がない
- 受信者のうち1つでもAEAD-to-CBCに脆弱な実装になっていると
  送信者が送ろうとした秘密が漏れる
- AEAD-to-CTRに脆弱な実装になっている受信者は
  任意の位置でビットフリップされた復号文を受け入れてしまう

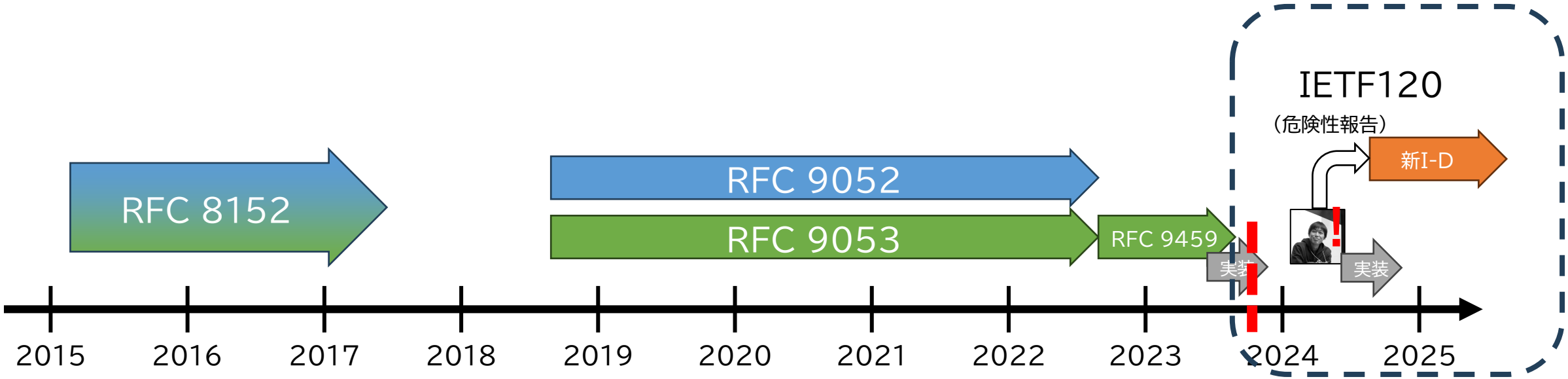


A) COSEライブラリがブラックボックスになっている場合など



B) 送信者が全受信者を管理しきれない場合など

対策の話



危険性の認識から対策まで

信頼される安心を、社会へ。
SECOM

- 適宜しかるべき相手に開示をしてきました

時期	人	できごと
2023/10	髙山 & Laurence	t_coseにAES-CTRとAES-CBC実装
2023/11	Falko & Johannes→IETF118	AEAD-to-CBC Downgrade Attack発表
2023/11	髙山	育児休業開始（IETFのこと忘れた）
2024/04	髙山	職場復帰（攻撃のことを思い出して調査）
2024/05	髙山 & セコムIS研究所メンバー	攻撃の亜種発見、一部COSEでも起こることを実証
2024/06	髙山→Hannes & Russ	RFC 9459著者に攻撃コードを報告
2024/06	髙山→Laurence	t_cose作者に報告、対策を提示、相談
2024/07	Hannes→IETF120	抜本的な対策案を新規I-Dに（共著Russ & 髙山）
2024/07	髙山 & Laurence	t_coseでnon-AEADデフォルト無効を実装

A）デフォルトでnon-AEAD無効

信頼される安心を、社会へ。
SECOM

AEADのみ許容（デフォルト）

```
# pre-shared KEK for A128KW
wrapping_key = new COSE_Key(
  kty=Symmetric, k=h'849B...', alg=A128KW)


decrypted_plaintext = COSE_Encrypt.decrypt(
  cose=cose_message, key=wrapping_key)
# decrypted_plaintext = "bob0" for AES-GCM
# decrypted_plaintext = NULL for AES-CTR
```

明示的にnon-AEADも許容

```
if verify_authentication_and_integrity(cose_message) == false
  return false   # RFC 9459に違反しないよう呼び出し側に検査させる

# pre-shared KEK for A128KW
wrapping_key = new COSE_Key(
  kty=Symmetric, k=h'849B...', alg=A128KW)

# decrypt_non_aeadのAPIマニュアルに「検査せよ」と注意書きをする
decrypted_plaintext = COSE_Encrypt.decrypt_non_aead(
  cose=cose_message, key=wrapping_key)
# decrypted_plaintext = "bob0" for AES-GCM
# decrypted_plaintext = "bob0" for AES-CTR
```

（t_coseの場合は関数を分ける以外の方法があったため、実際とは異なります）

https://github.com/laurencelundblade/t_cose/pull/284

B）安全な暗号ペイロードの作り方をする 新I-D

信頼される安心を、社会へ。 SECOM

- （補足）Bobは以下のように復号し、Eveに返すことで秘密を漏らす

$$
\begin{aligned}
d_j &= D_K(\boldsymbol{C} \oplus \boldsymbol{p_j}) \oplus ctr_j \quad \text{同じになったら相殺} \\
&= D_K(\boldsymbol{E_K(CTR_t)} \oplus \boldsymbol{P} \oplus p_j) \oplus ctr_j \\
&= \boldsymbol{D_K(E_K(CTR_t))} \oplus ctr_j \\
&= CTR_t \oplus ctr_j \quad \text{逆関数なので相殺}
\end{aligned}
$$

Eveが細工した暗号ブロック

Aliceが作った暗号ブロック$\boldsymbol{C = E_K(CTR_t) \oplus P}$

Eveの推測平文$\boldsymbol{p_j}$と、Aliceの平文$\boldsymbol{P}$が一致したら

EveがBobからもらうと$\boldsymbol{p_j = P}$だったと分かる値

- 攻撃を成立させる肝になっている箇所：「逆関数なので相殺」
  - BobはAES-GCMと、AES-CBCまたはAES-CTRで同じ復号鍵を使う
  - BobはAliceが送ろうとしたものを知らないので、意図せず使ってしまう

➢KDFを使って、暗号アルゴリズム別の復号鍵を導出すれば良いのでは？

- LAMPS WGでは、Russがこの対策を提案しS/MIMEに対して標準化中
- COSEライブラリでも実装可能なため、髙山含めて新規標準ドキュメント執筆中

B）復号側でも変更が必要

新I-D

信頼される安心を、社会へ。 SECOM

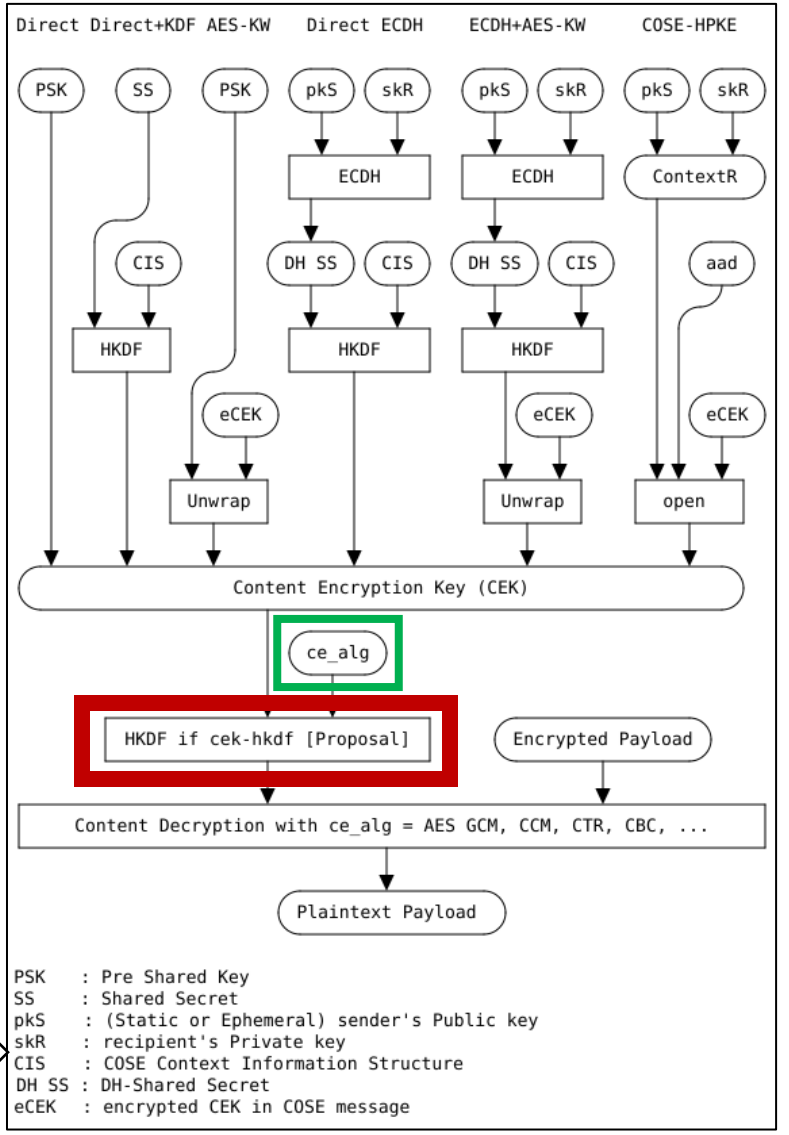
- AES-KWやECDHなどの鍵配送の後、復号する直前でKDFを挟んでK'を導出
  - KDFの入力には、復号アルゴリズムのIDを含める

$$= \boxed{D_{K'}(E_K(CTR_t))} \oplus ctr_j \quad (K' = HKDF(K, ce\_alg))$$

$$\neq CTR_t \oplus ctr_j$$

逆関数ではなく相殺されない

- 有効＆現実的な方法について議論中
  - draft-tschofenig-cose-cek-hkdf-sha256で提案中
  - まだまだv01版、今後に乞うご期待！



RFC 9052, 9053等を読みながら苦労して作った図

目次

- COSEの話
  - IETFで標準化されている、暗号ペイロード格納フォーマット
  - 暗号化アルゴリズムにはAES-GCMやAES-CTRなどをサポート
- AEAD-to-CBC Downgrade Attack関連の話
  - 受信者Bobが特定の挙動をすると、平文が漏れる
  - 受信者Bobが検査を漏らすと、改ざんされた平文を受け取ってしまう
- 対策の話
  - COSEライブラリの工夫で、意図せずnon-AEADで復号しないよう変更
  - 送信者Aliceが作る暗号ペイロードを工夫して、攻撃成立条件をつぶす

信頼される安心を、社会へ。

セキュリティ技術への標準化等を通して
安心で安全な社会を作ることに貢献していきます

Appendix

RFC 8152, 9052, 9053

（P.5補足）

信頼される安心を、社会へ。
SECOM

- COSEのRFCは少々複雑、STDとInformationalが混在している

RFC 8152 (STD 96)
Jim Schaad, **Standards Track**, 2017
デジタル署名・MAC・暗号文の、エンコード方法 と 暗号アルゴリズムを定義
暗号文用のアルゴリズム ・Key distribution methods: direct, AES-KW, ECDH-ES+AES-KWなど ・Content encryption algorithms: AES-GCM, CCM (いずれも認証付きのAEAD)

再編

RFC 9052 (STD 96)
Jim Schaad, **Standards Track**, 2022
エンコード方法を定義

RFC 9053
Jim Schaad, **Informational**, 2022
暗号アルゴリズムを定義
暗号文用のアルゴリズム ・Key distribution methods ・Content encryption algorithms

RFC 9459

（P.5補足）

信頼される安心を、社会へ。
SECOM

- non-AEADのAES-CTRとAES-CBCを追加
  - 特にSUIT用（IoT機器のためのファームウェアアップデート指示）
    - SUIT Manifestはデジタル署名/MACのレイヤーと、暗号化のレイヤーの2層構造
      - 完全性の検査と認証は前者で、後者では秘匿性だけを担保しても良い
      - ➢ AES-GCMだけでなく、AES-CTRやAES-CBCも使いたい（がRFC 9053にはない）
  - ”Implementations **MUST** use AES-CTR in conjunction with an authentication and integrity mechanism, such as a digital signature.” と警告。

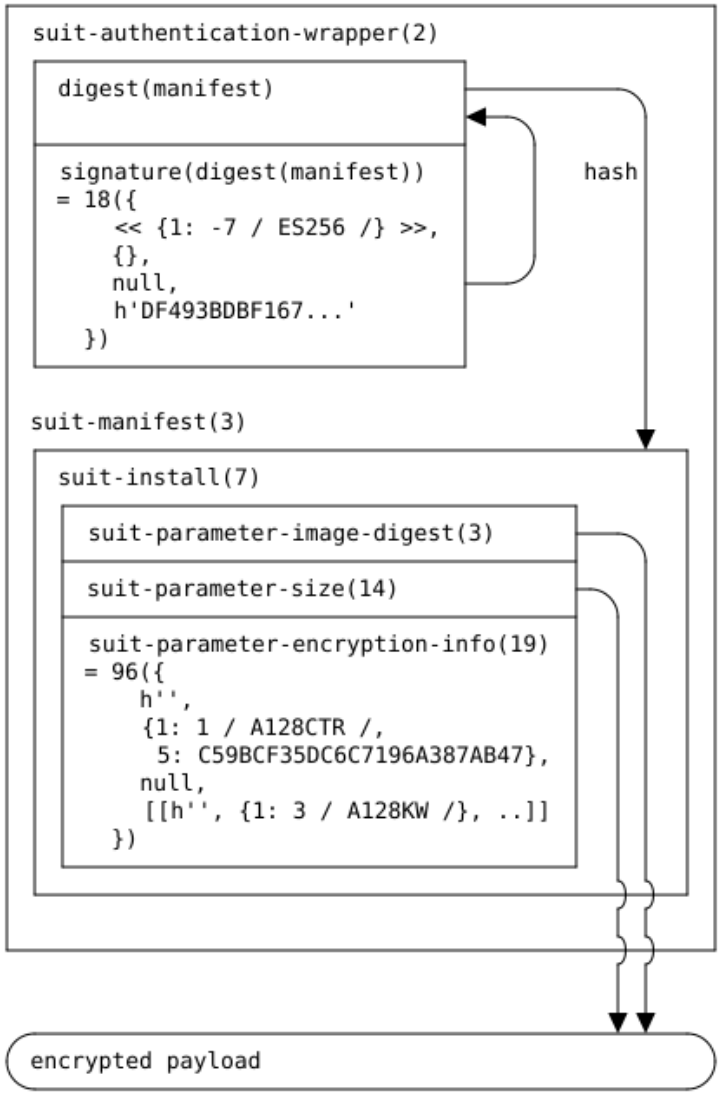
RFC 9053より9459の方が”強い”

RFC 9053		RFC 9459
Jim Schaad, **Informational**, 2022		Russ and Hannes, **Standards Track**, 2023
暗号アルゴリズムを定義	＋	暗号文用アルゴリズムを追加
暗号文用のアルゴリズム ・Key distribution methods ・Content encryption algorithms		・AES-CTR ・AES-CBC

髙山は呼び出し側コードでチェックしていた

- 髙山のlibcsuitがSUIT Manifestをパース
  1. suit-authentication-wrapperにあるCOSE_Sign1を使ってauthentication+integrityを確認
     - Laurence作のt_coseライブラリのverify関数を呼ぶ
  2. 暗号化されたペイロードのサイズ＆ハッシュ値を確認
  3. suit-install内にあるCOSE_Encryptを使って暗号化されたペイロード（ファームウェア等）を復号
     - Laurence作のt_coseライブラリの復号関数を呼ぶ
     - 「A128GCMがあるから、ここにA128CTRを足そう」

➢RFC 9459に準拠してAES-CTRを利用していた
- それ以外のt_cose使用者を危険にさらしてしまった



COSEの復号関数に期待されていたこと

信頼される安心を、社会へ。
SECOM

- RFC 8152, 9053だけを読んだCOSEライブラリユーザーは AEADのみを想定して復号してもおかしくない
  - 特にRFC 9459で後からnon-AEADが追加されたことを知らない場合
  - 復号処理に成功したということは、完全性＋認証も検証されたと思いかねない

➢non-AEADが実際に使われていた場合、
RFC 9459が要求する注意事項を、Bobが守らない可能性がある

- ”Implementations **MUST** use AES-CTR in conjunction with an authentication and integrity mechanism, such as a digital signature.”

この状況をどうやって解決するか？

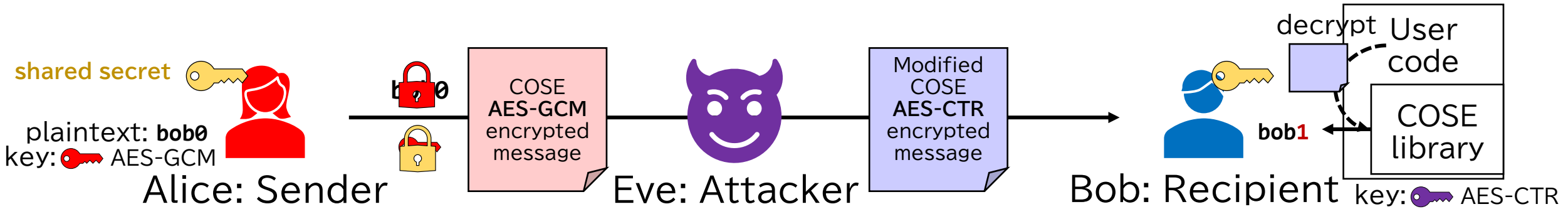
（P.19補足）

A）復号関数実行前に、ライブラリ呼び出し側が気を付ける
- non-AEADである場合は必ず、別途完全性＋認証をチェック（RFC9459遵守）

B）Bob側がnon-AEAD（AES-CTRとAES-CBC）を実装しない
- RFC 9459は”廃止済み”として登録しているため、実装しない理由の方が強い

C）COSEライブラリはデフォルトでnon-AEADを無効にする
- 有効にするためのインタフェースを別で用意する

D）Alice側で暗号ペイロードの作り方に新しい仕組みを導入する



この状況をどうやって解決するか？

（P.19補足）

信頼される安心を、社会へ。
SECOM

A）復号関数実行前に、ライブラリ呼び出し側が気を付ける
- 👍ライブラリ呼び出し側の裁量でRFC 9459を守ることができる
- 👎そもそもCOSEバイナリの中身を見たくないからライブラリを使ってるのに…

B）Bob側がnon-AEAD（AES-CTRとAES-CBC）を実装しない
- 👍最も基本的で確実
- 👎実装しなければならない場合（例：髙山が標準化に携わるSUIT）もある

C）COSEライブラリはデフォルトでnon-AEADを無効にする
- 👍COSEライブラリ側の工夫でRFC 9459を守るよう誘導できる
- 👎理解せずnon-AEADを有効にしてしまうBobまでは守れない

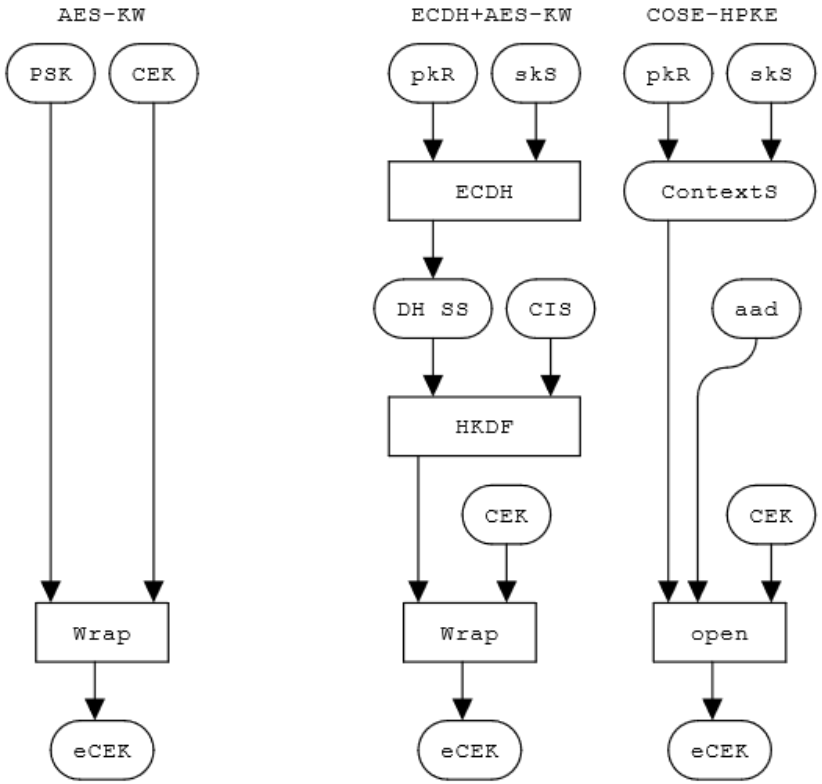
D）Alice側で暗号ペイロードの作り方に新しい仕組みを導入する
- 👍秘密を守ってほしいAlice側の裁量でconfidentialityを強化できる
- 👎AliceもBobも追加の実装が必要になる

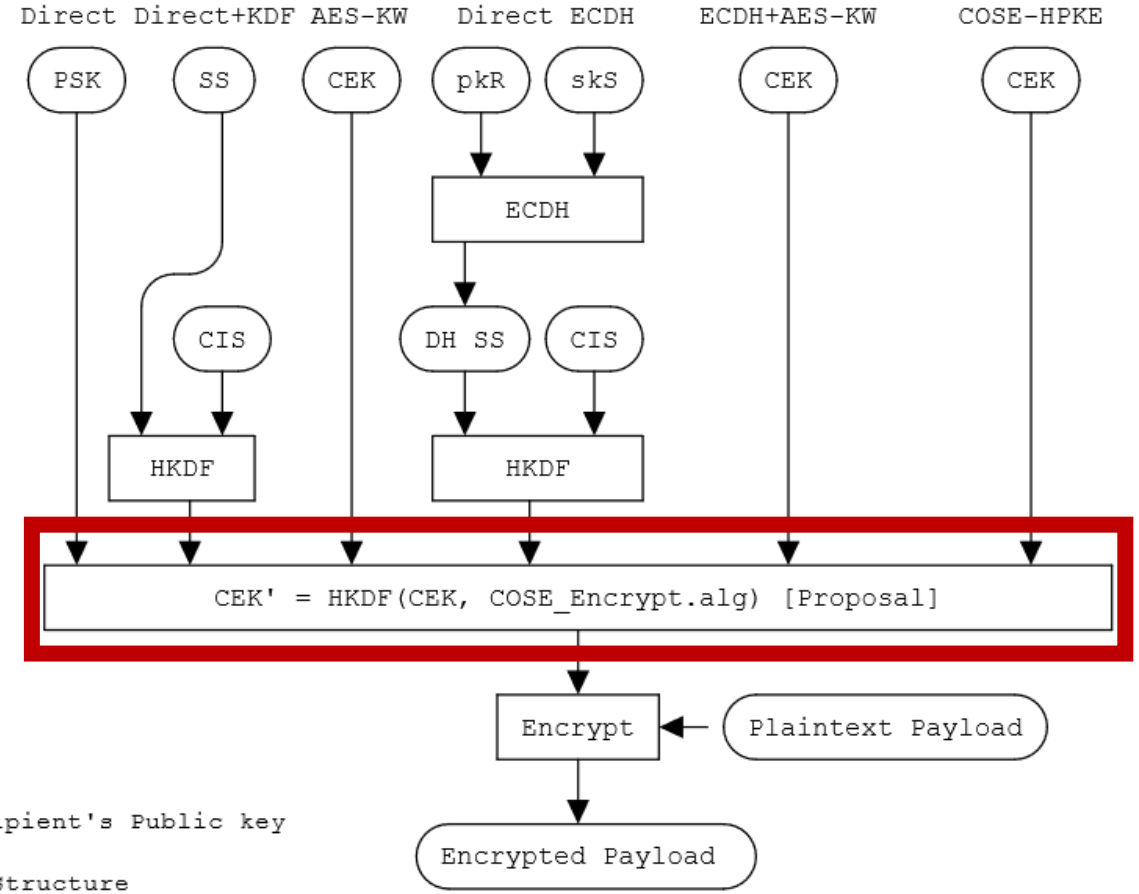
新I-Dで変更する暗号化フロー案

（P.21補足）

信頼される安心を、社会へ。
SECOM

- 鍵配送アルゴリズムには変化なし、暗号化部分に変化






```
CEK    : Content Encryption Key
PSK    : Pre Shared Key
SS     : Shared Secret
pkR    : (Static or Ephemeral) recipient's Public key
skS    : sender's Private key
CIS    : COSE Context Information Structure
DH SS  : DH-Shared Secret
eCEK   : encrypted CEK in COSE message
```