JP AAWG JAPAN ANTI-ABUSE WORKING GROUP

マニアックセッション ～次世代のメールプロトコルの斜め読み（2026年版）

Hornetsecurity

平野善隆

2026/11/06

自己紹介

名前　　平野 善隆

所属　　Hornetsecurity株式会社 (旧Vade Japan)
Principal Messaging Engineer

好きな技術　　メール、DNS、Python、Go
AWS、Serverless

趣味　　長距離の自転車大会(1,200kmとか、2,000kmとか)
バンド演奏

主な活動　　M³AAWG
JPAAWG
迷惑メール対策推進協議会
Audax Randonneurs Nihonbashi





メールとの関わり

1990	パソコン通信などでメールに触れる
199x	ドメインを取得して近所のISPに個人のサーバーを置かせてもらって運用開始
2000	外人さんの多い会社に転職したのでメールの漢字にふりがなを付けたりして遊ぶ (のちのhiragana.jp)
	個人のサーバーをちゃんとしたデータセンターに移動。 imail.ne.jpというドメインを取って一攫千金を夢見るが挫折
2004	メールの会社に入社
以降	スパムフィルタ、誤送信防止製品の開発やサービスの立ち上げ。 PPAPの礎を築く。
2023	Vade Japan(現Hornetsecurity)に入社

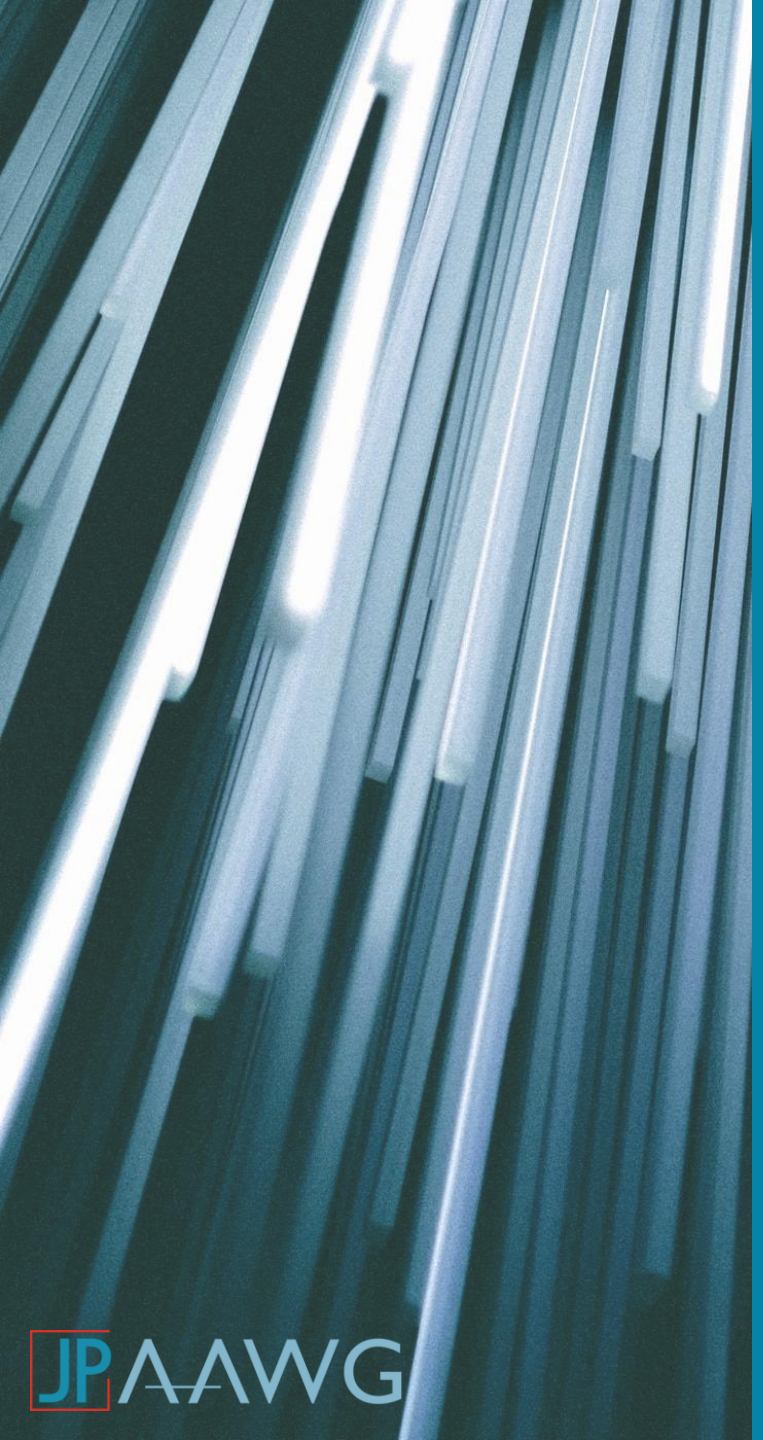
www.vadesecure.com

注意事項

- あくまで斜め読みです
- 間違ってたらごめんなさい
- 深い質問をされても、たぶん困ります

お題

- **DMARC**
  - DMARCbis (RFC 9989～9991)
  - ARC を Historic に (arc-to-historic)
- **新しいRFC**
  - Expires ヘッダ (RFC 10080)
- **DKIM2**
  - DKIM2 のドラフトの全体像
  - DKIM2 本体 (spec-09)
  - Message-Instance と Recipe
  - DSN (バウンス)
  - 鍵の DNS レコード (dns-03)
  - DKIM2 Sender Policy
  - デバッグ用ヘッダ (debug-header)
- **そのほかのドラフト**
  - HMTP (HTTPS でメールを転送)
  - VEIF (送信者の身元の保証)
  - APRF (配信結果の集約レポート)
  - Unobtrusive Signatures (E2E 署名)
  - Mail Action Protocol (MAP)
  - Structured Quoted Content (引用)
  - AI 処理の希望を伝えるヘッダ
  - Secure SMTP/TLS SRV Announcement
  - RFC5321bis / RFC5322bis
  - SMTP Client Identity
  - LIMITS 拡張 (RFC 9422)



RFC9989 (DMARCbis)

次世代のDMARC

https://www.rfc-editor.org/rfc/rfc9989

→ 2026年5月に RFC 9989 として発行

JPAAWG

DMARCbis

- 今のDMARCの嫌なところを解決しています
- 曖昧なところを詳しく書き直しています
- 今のDMARCのRFC、RFC7489が廃止されました
- DMARCのPublic Suffix Domainに関するRFC9091も廃止されました

→ 2026年5月に RFC 9989 として発行！ (Proposed Standard)
→ 名前は「DMARCbis」ではなく普通の「DMARC」。v=DMARC1 のまま

DMARCは3つのRFCに分かれた

- **RFC 9989**　DMARC 本体
- **RFC 9990**　集約レポート (rua)
- **RFC 9991**　失敗レポート (ruf)
- 3本とも 2026年5月発行の Proposed Standard
  - RFC 7489 は Informational だったので、初めて Standards Track に
- RFC 9989 の中身は draft-41 からほぼ編集上の修正だけ
  → 昨年のこの講演の内容はそのまま有効
- 分けたのは、今後それぞれを個別に改訂できるようにするため

新しく追加されたタグ

- 新しく追加されたタグ
  - np, psd, t

- 削除されたタグ
  - pct, ri, rf

新しいタグ (np)

- 存在しないサブドメインに対するポリシー

- 存在しないとは?
  - sub.example.jpが存在するかどうか
  - _dmarc.sub.example.jpがあるかどうかは関係ない

example.jp　v=DMARC1; p=none; sp=quarantine; np=reject

sub.example.jpが存在しない → reject

sub.example.jpは存在する

_dmarc.sub.example.jpが存在しない → quarantine

_dmarc.sub.example.jpがp=none → none

p, sp, npの関係

親ドメイン			Subdomain		参照ポリシー
p	sp	np	存在する?	_dmarc	
あり	なし	なし	存在しない	-	親ドメインのp
あり	なし	なし	存在する	なし	親ドメインのp
あり	なし	なし	存在する	あり	subドメインのp
あり	あり	なし	存在しない	-	親ドメインのsp
あり	あり	なし	存在する	なし	親ドメインのsp
あり	あり	なし	存在する	あり	subドメインのp
あり	なし	あり	存在しない	-	親ドメインのnp
あり	なし	あり	存在する	なし	親ドメインのp
あり	なし	あり	存在する	あり	subドメインのp
あり	あり	あり	存在しない	-	親ドメインのnp
あり	あり	あり	存在する	なし	親ドメインのsp
あり	あり	あり	存在する	あり	subドメインのp

新しいタグ (psd)

- Public Suffix Domainを表すタグ
  - y, n, u が書ける

現行のDMARCの仕組み

- a.b.c.hornetsecurity.com
  - 1. _dmarc.a.b.c.hornetsecurity.com を調べる
    - → あれば使う
  - 2. なければ
    _dmarc.b.c.hornetsecurity.com を調べる?
    _dmarc.hornetsecurity.com を調べる?
  - 3. _dmarc.hornetsecurity.comにもなければ
    _dmarc.com を調べる?

組織ドメインを調べる

調べない

組織ドメインより上は調べない

Public Suffix Domainとは

- jp, co.jp, com, net などの組織ドメインの1つ上の階層のこと
- もともとは、webのクッキーの境界のため

company1.net
company2.net

お互いクッキーが見えるとまずい

sub1.example.net
sub2.example.net

お互いクッキーが見えても問題ない

website1.cloudfront.net
website2.cloudfront.net

こちらも、まずい

cloudfront.netもPublic Suffix Domain

- Public Suffix Listというものが(いくつか?)ある
  が、管理が曖昧

新しいタグ (psd)

- 既存の曖昧なPublic Suffix Listを使わずに、
  自分で境界を宣言するためのタグ
  - y: 自分はPublic Suffix Domainです。ここより上は探索しないでね。
    例: jp, co.jp, cloudfront.net
  - n: 自分は組織ドメインまたはそのサブドメインです。
    ここより上は探索しなくていいよ。
  - u: 不明 (デフォルト)
    組織ドメインかPSDになるまで上位のドメインを探索する。
    → レコードがない場合もこれ

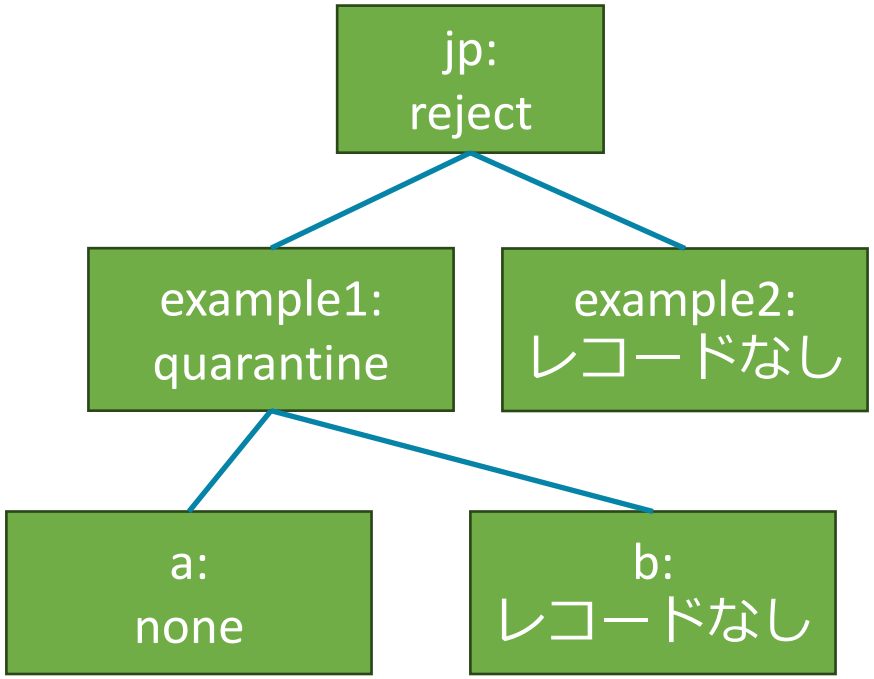
a.9.8.7.6.5.4.3.2.com

辿るのは上位8階層だけ

こんな感じ?

- _dmarc.jp p=reject; psd=y
- _dmarc.example1.jp p=quarantine; psd=n
- _dmarc.a.example1.jp p=none; psd=n
- _dmarc.b.example1.jp レコードなし
- _dmarc.example2.jp レコードなし

- example1.jp → quarantine
- a.example1.jp → none
- b.example1.jp → quarantine
- example2.jp → reject



現在のDMARCではPSDまで辿らないので
DMARCは「なし」になる

新しいタグ (t)

- テストモード
- 1段階低いポリシーが適用される
  - rejectの場合はquarantine
  - quarantineの場合はnone
  - noneの場合は変化なし
- 値はyとn
- デフォルトはn
- 詳しくは付録A6で議論

t=y と pct=0

- pctは0～100の任意の値を設定できたが、受信側は事実上0と100しか実装していなかった。
- 「p=none」と「p=quarantine; pct=0」に違いがある?!
- メーリングリストなどでヘッダFromを転送サービスのドメインに書き換える機能がある。
  → p=quarantine, rejectの場合には書き換わる
  → p=noneでは書き換わらない
- 0, 100しかないのにpctはおかしい
- tタグを新たに導入、pctは廃止

廃止されたタグ

- pct: tに置き換え
- ri: レポートの送信間隔。　誰も実装せず無視して日次で送信。
  → RFC 9990 では「日次 (またはそれ以上の頻度)、期間は通常 UTC の1日」
- rf: 失敗レポートのフォーマット afrf(ARF形式)とiodefが指定できたが、ARFに固定。
- rua / ruf の URI 末尾の「!10m」のようなサイズ指定も廃止
  → RFC 9990では、あっても無視される

レポートを外部ドメインに送るときの例のバグ修正

- 現行DMARCのRFC7489

  example.com._report._dmarc　IN　TXT　"v=DMARC1"

- RFC 9990 の4章

  example.com._report._dmarc　IN　TXT　"v=DMARC1;"

ここの「;」

- RFC 9990 では「末尾の ; は省略可」と明記された
  → どちらでもOK

用語集

- RFC9989には詳しい用語集が付いている (3章)

- Monitoring Mode ← p=noneの状態
- Enforcement ← p=quarantineやrejectの状態

ドメイン所有者のアクション

- SPFをヘッダFromのドメインで公開
- DKIMをヘッダFromのドメインで署名するように設定
- DMARCレポートを受けられるように準備
- p=noneとruaタグを書いたDMARCレコードを公開
- レポートを集めて分析
- 認証に失敗している原因を修正
- ポリシーを厳しくするかどうかを決める

必ずしもp=rejectにするのがいいとは限らない

慎重になるべきポイント

- 卒業生向けメール転送サービス、部署名のエイリアス、メーリングリストなどに送る場合
- かつ
- 中継するメールサーバーがヘッダFromアドレスを書き換えない場合

- SPFに依存してはならない(MUST NOT)
- 有効なDKIM署名が必要 (MUST)

慎重になるべきポイント2

- 一般ユーザーが日常的にメールを送るようなドメインで、インターネット上のメーリングリストに送るような場合

- p=rejectにするべきではない (SHOULD NOT)
- p=rejectにしたい場合は
  - 1ヶ月間p=noneでレポートを分析
  - さらに1ヶ月間p=quarantineでレポートを分析、noneの時と比較する
  - p=rejectにする場合は、ユーザーに
    - メーリングリストに投稿させないようにするか
    - メーリングリストの参加が妨げられる可能性があることを周知すべき (SHOULD)

メール受信者のアクション

- ヘッダFromからドメインを抽出
- DMARCが宣言されているか確認
- SPF、DKIMのチェック
- アラインメントを確認
- DMARCのPASS, FAILを決定
- ポリシーを適用する
- 結果を保存する
- Aggregateレポートを送る
- 失敗レポートを送ってもいい

ポリシー強制の懸念

- DMARC PASSはヘッダFromドメインが正当なものであることを示すにすぎず、メッセージの価値判断をするものではない
- DMARC PASSでも隔離したり、Rejectしてもよい

- p=rejectでDMARCが失敗しても、メールを受け入れてもよい
- p=rejectだけを理由にRejectすべきではない (SHOULD NOT)

- p=noneの場合は既存の処理プロセスを変更してはならない (MUST NOT)

慎重になるべきポイント

- メーリングリスト管理ソフトは、メールが届かない場合、リストから購読解除する
- メーリングリスト経由のメールをp=rejectだからといって拒否すると、送信者がメーリングリストから購読解除されてしまう
- この場合、受信サーバーはp=rejectのみに基づいて拒否してはならない(MUST NOT)

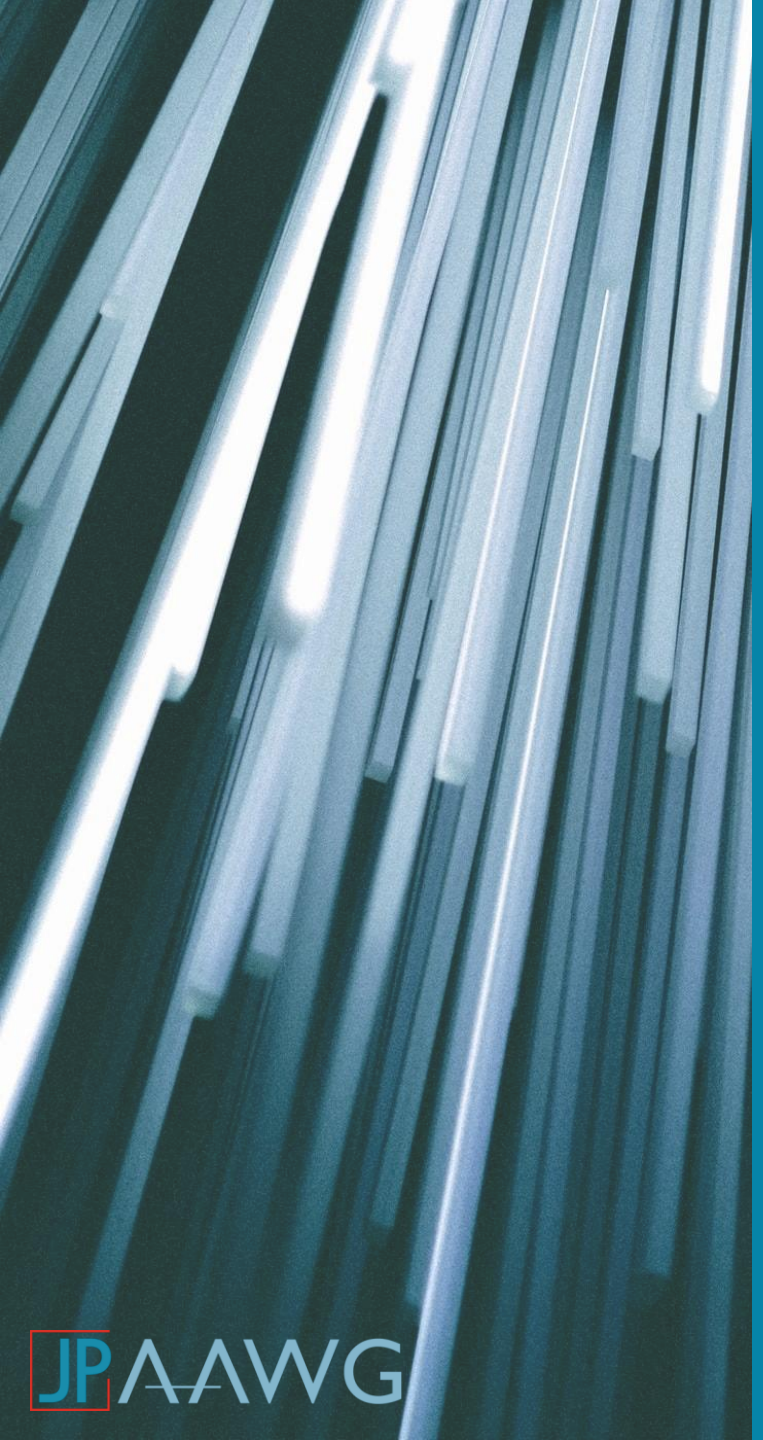
ARCよりはヘッダFromの書き換え

- ここ10年でメーリングリストソフトはすでに
  bob@example.com を
  bob=example.com@user.somelist.example
  のように書き換えるような修正をおこなった
- 理想とはほど遠いがメーリングリスト業者は受け入れている

- ARCも解決策のひとつで研究は現在も進行中であるが、
  この文書の発行時点では、広く普及するには至っていない
- 将来、広く採用されるようになった場合には、
  本文書は更新される
- → そのARCは、DMARC WGが2026年4月に再チャーターされ
  Historic へ移す作業中

RFC9990 集約レポートの変更点

- XML名前空間はurn:ietf:params:xml:ns:dmarc-2.0
- policy_publishedにnp, testing (t タグ), discovery_method が入る
  - discovery_methodは psl (RFC 7489方式) か treewalk (RFC 9989方式)
- DKIM の結果には selector が必須
- 上書き理由にpolicy_test_mode が追加
  (pct がなくなったので sampled_out は消えた)
- extension要素で拡張できる
- 1つのレポートには1つの DMARC Policy Domain だけ
- レポートは日次 (またはそれ以上の頻度)。
- 期間は通常 UTC の1日



Reclassifying ARC as Historic

draft-ietf-dmarc-arc-to-historic-01 (2026-10-21)

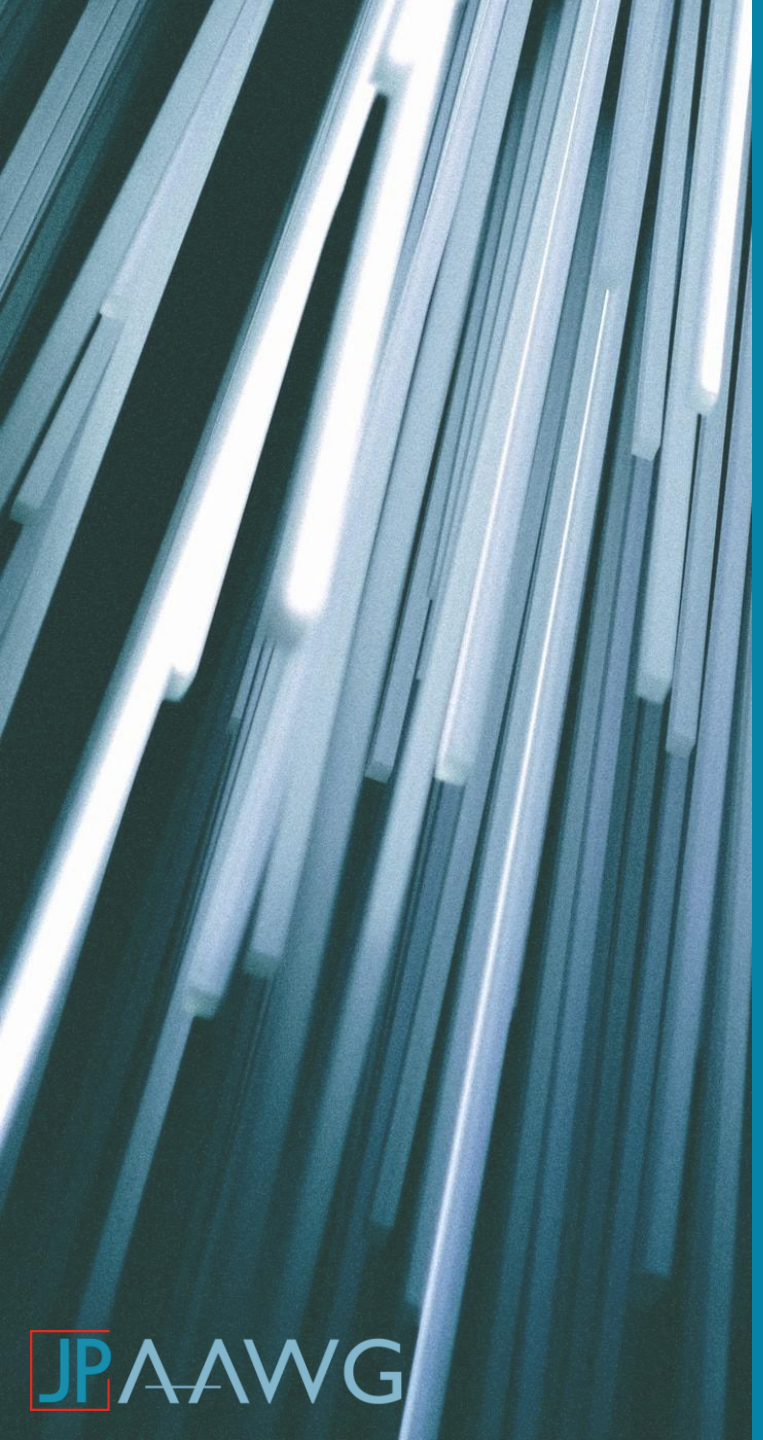
JP AAWG

ARCの実験は終わり

- RFC 8617 (ARC, 2019年, Experimental) を Historic にする
  - DMARC WG が2026年4月に再チャーター
  - ARC で分かるのは「誰が中継したか」だけ
  - 「何を変えたか」は分からない
- 署名が検証できても、それは信頼ではない
  - 中継者ごとのレピュテーションが必要だが、10年で実現しなかった
  - 実際の運用は allow list 止まり
- 転送による認証の失敗は今も一番多い問題

実装者・運用者へのガイダンス

- 新しくARCを導入しない
- 今のARCは廃止するか、組織内の閉じた用途だけにする
- ARCの検証結果で配送の判断をしない
  - 調査・診断のために検証するのはかまわない
- ARCの教訓は DKIM2 に引き継がれる
  - 送信元・宛先を署名 (Chain of Custody)
  - 何を変えたかを Message-Instance に記録
- ARCに興味がある人は DKIM2 を見よう



Updated Use of the Expires Message Header Field

RFC10080 (2026-10-30)

JPAAWG

RFC 10080 の概要

- メールのヘッダ Expires: を標準化
  - 昔から使われていたが、正式な仕様がなかった
- 「このメールはいつ役に立たなくなるか」を示す
- 期限が過ぎたら、表示の優先度を下げたり、隠したりできる
- 意味を細かく決めすぎない。受け取った側の動作も強制しない
  - 既存の使い方を壊さないため
- 書式は RFC 5322 の 3.3 節の日時と同じ

```
Expires: Thu, 16 Jul 2026 11:28:03 +0000
```

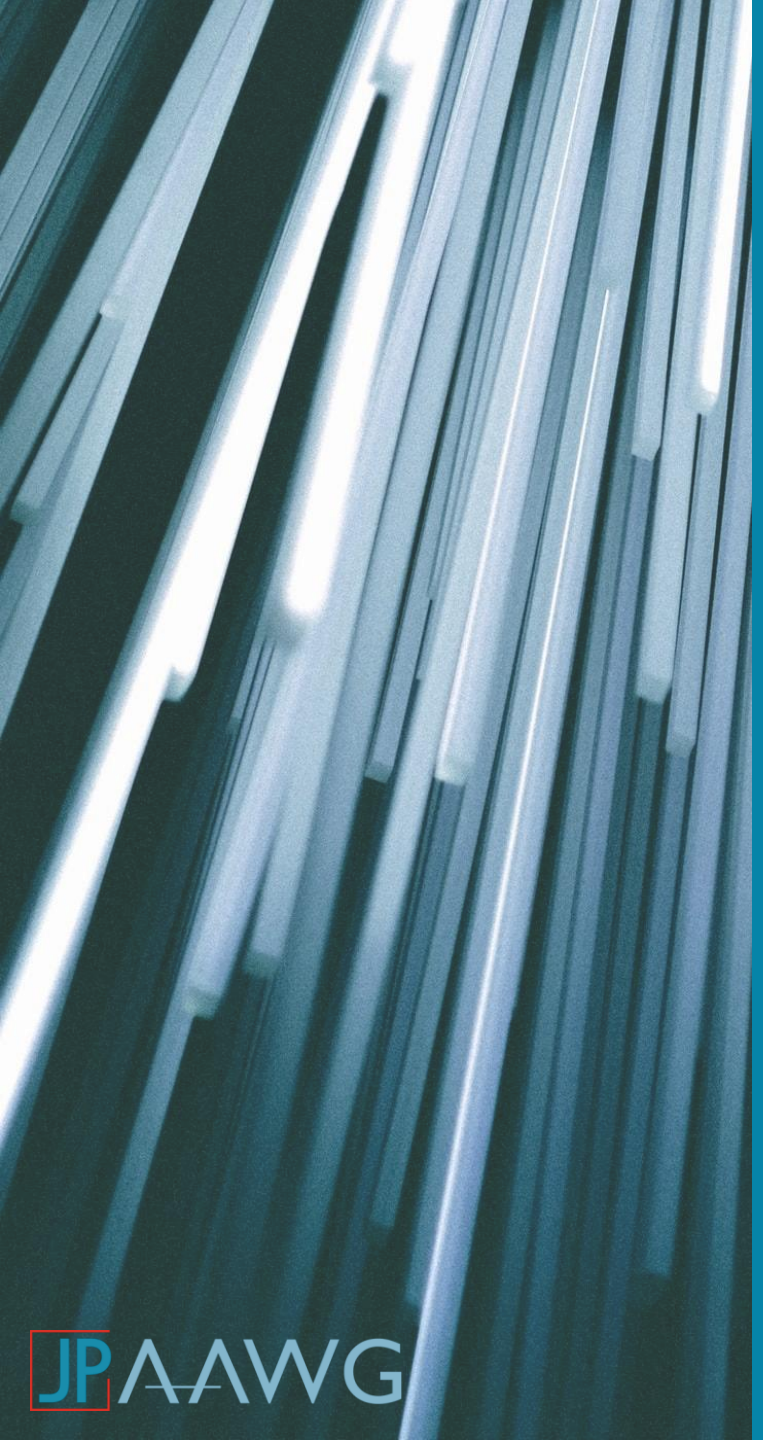
※ 内容は S. Bortzmeyer 氏のブログの解説をもとに作成

例: 期間限定セールのお知らせ

- セールの最終日の夜に期限を設定。過ぎたらメーラーが目立たなくする
- (補足) Expires: を DKIM の署名対象に入れておけば、途中で書き換えられても分かる

```
From: Example Shop <news@shop.example>
To: user@example.jp
Subject: 週末限定セール (10/31 まで)
Date: Mon, 26 Oct 2026 09:00:00 +0900
Expires: Sat, 31 Oct 2026 23:59:59 +0900
DKIM-Signature: v=1; a=rsa-sha256; d=shop.example; s=sel1;
    h=from:to:subject:date:expires:message-id; ...

  (本文)
```



DKIM2

次世代のDKIM
https://datatracker.ietf.org/doc/draft-ietf-dkim-dkim2-spec/

JPAAWG

DKIM2のRFCドラフト

- **2026年11月時点のWGドラフトは3本**

- DomainKeys Identified Mail Signatures v2 (DKIM2)
  (draft-ietf-dkim-dkim2-spec-09, 2026-11-01)
- Domain Name Specification for DKIM2
  (draft-ietf-dkim-dkim2-dns-03, 2026-11-02)
- DKIM2 Best Practices
  (draft-ietf-dkim-dkim2-bcp-01, 2026-09-09)

- まだ RFC ではない

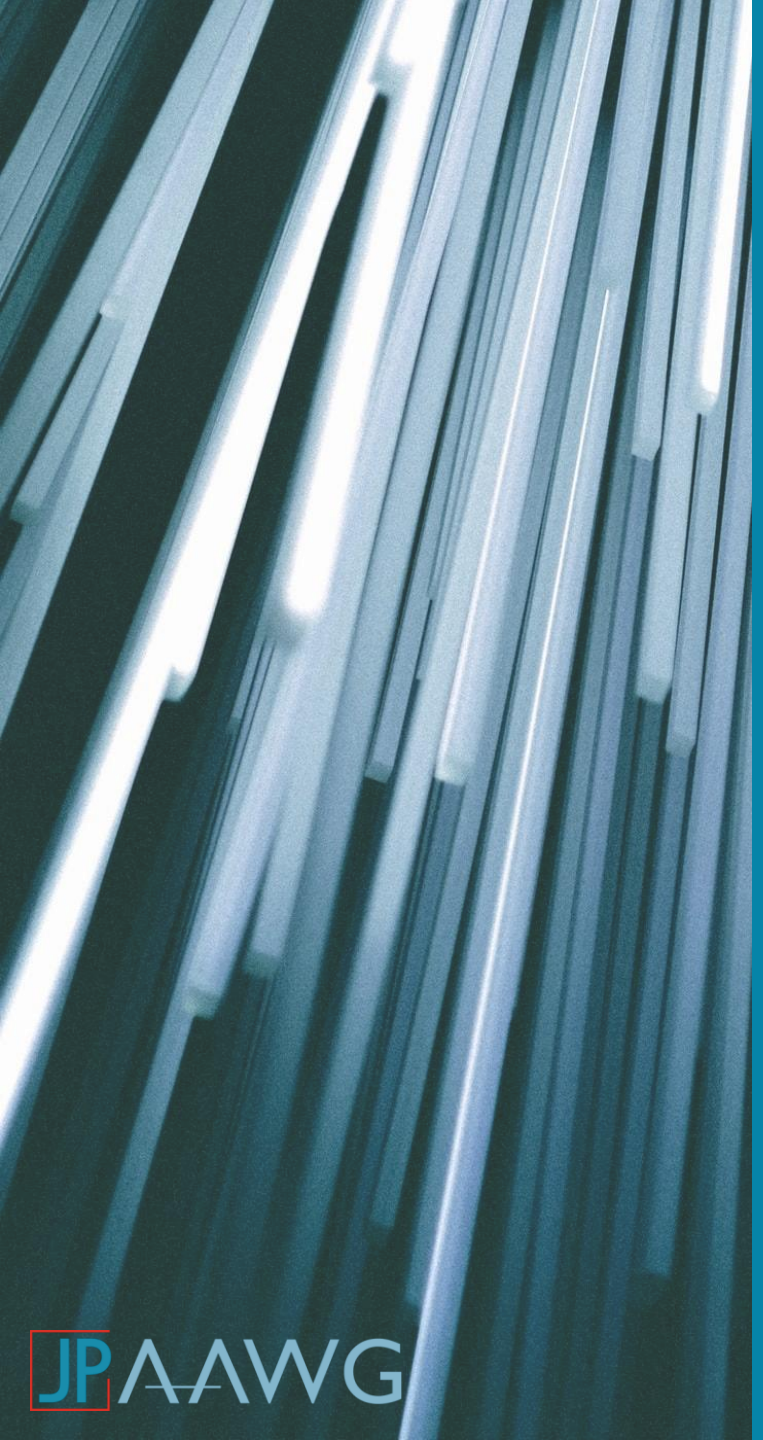
DKIM2のRFCドラフト (2)

- **WG外のドラフト**

- DKIM2 Sender Policy
  (draft-chuang-dkim2-sender-policy-04, 2026-10-18)
- A Diagnostic Header Field for DKIM2 Implementations
  (draft-gondwana-dkim2-debug-header-03, 2026-11-02)

前回紹介したドラフトのその後

- DKIM2 - signing the source and destination of every email
  (motivation-02) → 2026年5月に Expired。考え方は spec の Introduction へ
- DomainKeys Identified Mail Signatures v2 (DKIM2)
  (draft-clayton-dkim2-spec) → 2026年3月に WG文書 draft-ietf-dkim-dkim2-spec へ
- DKIM2 Header Definitions / A method for describing changes
  (header, modification-algebra) → spec に統合 (Message-Instance + JSON Recipe)
- Domain Name Specification for DKIM2
  (draft-chuang-dkim2-dns) → 2026年7月に WG文書 draft-ietf-dkim-dkim2-dns へ
- Bounce processing / Message Examples
  (robinson) → DSN の扱いは spec の16章で規定



DomainKeys Identified Mail Signatures v2 (DKIM2)

draft-ietf-dkim-dkim2-spec-09 (2026-11-01)

JPAAWG

spec の概要

- DKIM (DKIM1) で解決できなかった問題に取り組む。
- ARC の実験の教訓も取り入れた
  - DKIM Replay 攻撃 / 転送や改変で署名が壊れる / バウンスのバックスキャッター
- DKIM2 の考え方
  - MAIL FROM / RCPT TOも署名し、配送経路をつなぐ (Chain of Custody)
  - 変更したら、元に戻す手順 (Recipe) を記録する。
  - 変更の内容から中継者のレピュテーションを判断できる
  - バウンス (DSN) は、配送に関わったところにだけ返る
  - 署名者はフィードバックを求めることもできる
- ヘッダは2種類
  - DKIM2-Signature: 中継ごとの署名 (i=1, 2, 3 …)
  - Message-Instance: その時点のハッシュと変更履歴 (m=1, 2, 3 …)
- 鍵の置き方は別ドラフト (dns)、運用は BCP に

DKIM2-Signature ヘッダのタグ

- i: 署名の連番 (必須)
- m: その時点で最大の Message-Instance の番号 (必須)
- t: タイムスタンプ (UNIX時間、10^12 まで想定) (必須)
- d: 署名ドメイン。mf= のドメインと同じか、その親 (必須)
- s: セレクタ:アルゴリズム:署名値 の組み合わせ (必須)
- mf: MAIL FROM を < > 込みで base64 (パラメータは含めない)
- rt: RCPT TO を < > 込みで base64 (複数可)
- nd: 次に署名するドメイン (mf / rt の代わり)
  ※ mf + rt か、nd のどちらかが必須
- f: フラグ
- n: nonce (64文字まで) 署名者が自由に使っていい
- 同じタグが2回出てきたら、タグリスト全体が無効

i=タグ

- 中継毎にカウンターを増加
- 上限の規定はなくなった
- BCPでは「ローカルで上限を決めて超えたらPERMERROR」を推奨
- mf= / rt= は < > 込みでbase64
  - PGFAb3JpZ2luLmV4YW1wbGU+ = <a@origin.example>

```
MAIL FROM: <a@origin.example>
RCPT TO: <b@alias.example>
DATA
DKIM2-Signature: i=1; m=1; t=1792713600; d=origin.example;
    mf=PGFAb3JpZ2luLmV4YW1wbGU+; rt=PGJAYWxpYXMuZXhhbXBsZT4=;
    s=key1:rsa-sha256:K7xQ...
Message-Instance: m=1; h=sha256:9fWb...:Lx3A...
From: <a@origin.example>

A Message!
```

```
MAIL FROM: <b@alias.example>
RCPT TO: <c@destination.example>
DATA
DKIM2-Signature: i=2; m=1; t=1792713660; d=alias.example;
    mf=PGJAYWxpYXMuZXhhbXBsZT4=; rt=PGNAZGVzdGluYXRpb24uZXhhbXBsZT4=;
    s=key4321:rsa-sha256:p0Zr...
DKIM2-Signature: i=1; m=1; t=1792713600; d=origin.example;
    mf=PGFAb3JpZ2luLmV4YW1wbGU+; rt=PGJAYWxpYXMuZXhhbXBsZT4=;
    s=key1:rsa-sha256:K7xQ...
Message-Instance: m=1; h=sha256:9fWb...:Lx3A...
From: <a@origin.example>

A Message!
```

JPAAWG

t=タグ

- 署名のタイムスタンプ
  - UNIX時間 (1970/1/1 00:00:00 UTC からの秒数)
  - 例: t=1792713600
- 配送時点で14日より古い署名がある → 無効
- 1日以上未来の署名がある → 無効
  - Replay検知やDSN用のキャッシュを安心して捨てられるように
- BCP: 署名サーバの時計はNTPで合わせよう

f=タグ

- すでにやったこと・受信サーバにやってほしいこと
- exploded
  - 他の人(複数の宛先)に送った。i=1 には付けてはいけない
- feedback
  - Feedbackがほしい (中身や送り先はまだ未定義)
- feedhere （新)
  - Feedbackはこのホップ経由で。転送先を知られたくない転送者が付ける
- ~~modifiedbody, modifiedheader~~ ← 廃止
- ~~donotmodify, donotexplode~~ ← 廃止

s=タグ

- s=セレクタ:アルゴリズム:署名値
- 「,」でand、「|」でor条件を書ける
- 署名はMessage-InstanceとDKIM2-Signatureヘッダだけに対して計算
  - 本文やヘッダはハッシュ経由で間接的に署名される

```
DKIM2-Signature: i=1; m=1; t=1792713600; d=example.jp;
  mf=PGFAZXhhbXBsZS5qcD4=; rt=PGJAZXhhbXBsZS5uZXQ+;
  s=sel1:rsa-sha256:abcdefg,sel2:ed25519-sha256:hijklmn|sel3:rsa-sha512:opqrtu
Message-Instance: m=1; h=sha256:abc:def
From: <a@example.jp>

A Message!
```

ハッシュ (旧 bh= タグ)

- 本文とヘッダのハッシュは Message-Instance の h= に入る
  - h=sha256:<ヘッダのハッシュ>:<本文のハッシュ>
  - sha256 と sha512 が使える
- 本文: DKIM1 の simple と同じ (末尾の空行だけ無視)
  - 2025年版の「relaxed 固定」から変わった
- ヘッダ: 名前を小文字化・unfold・空白を圧縮し、名前順に並べてハッシュ

nd=タグ

- next domain
- 受け取ったドメインと、転送に使うドメインが違うとき用
  - 例: example.com 宛に届いたメールを example.net から転送
- nd= があるときは mf= / rt= を書かない
- 次の署名の d= は nd= と完全一致しなければならない
- nd= を使わず「架空のホップ」の署名を作ってつなぐ方法もある

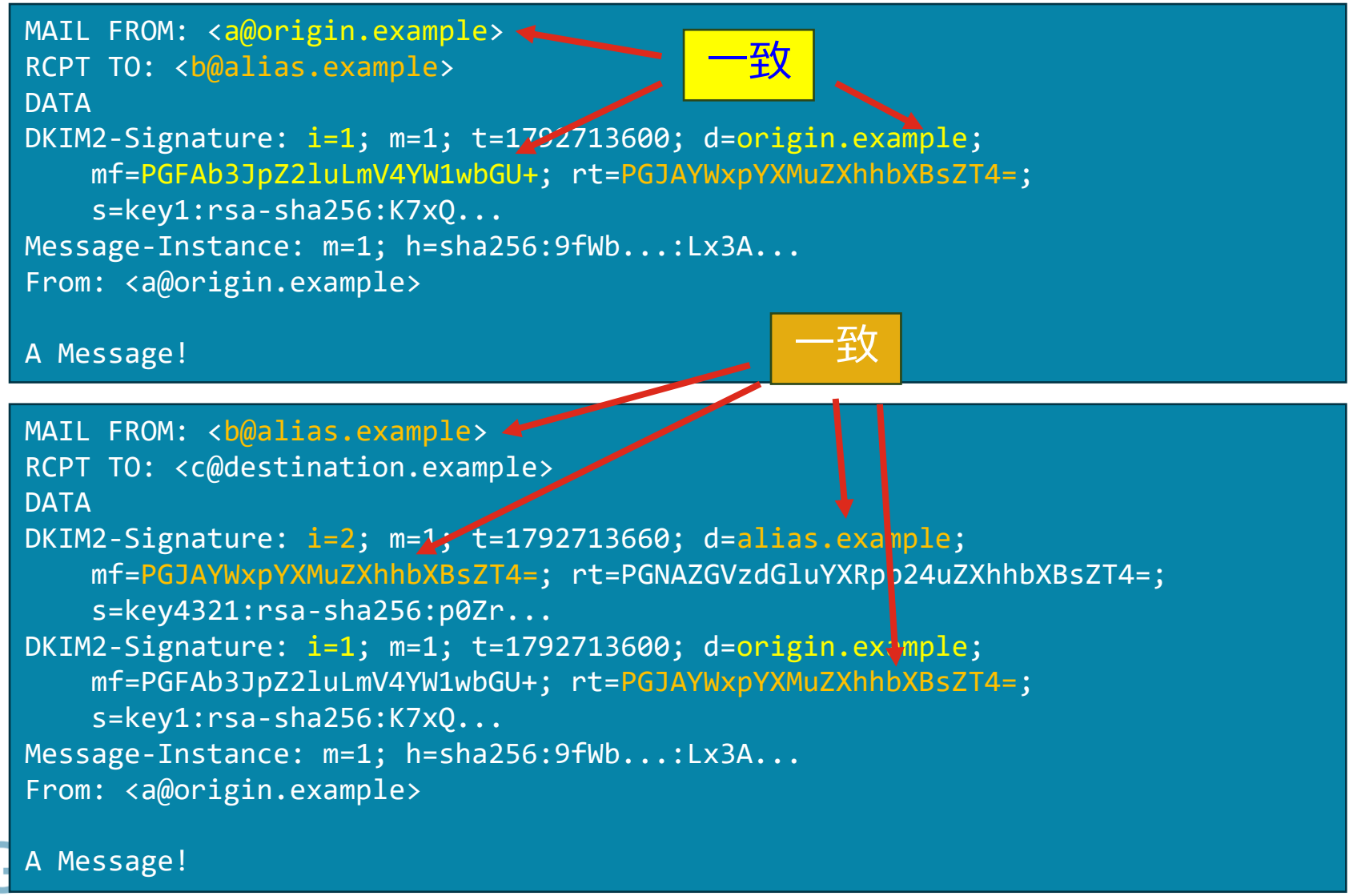
m=タグ

- Message-Instance (旧 MailVersion) の番号と連動
- 署名した時点で一番大きい Message-Instance の m= を書く
- どの署名者がどの変更をしたのかが分かる
- どの署名からも参照されていない Message-Instance があったら無効

Chain of Custody

- 各ホップの mf= / rt= で配送経路を鎖のようにつなぐ
- 受信時: 実際の MAIL FROM / RCPT TO と最新の署名の mf= / rt= が完全一致
  - local part も一致が必要。rt= に余分な宛先があるのはOK
- i=n の mf= と i=n-1 の rt= が relaxed domain match
- どの署名も d= と mf= が relaxed domain match
- relaxed domain match: 左のラベルを1つずつ削りながら一致を探す
  - bounce.mail.example.com は example.com に一致
  - Public Suffix List は不要
- 何も変えずに経路どおり転送するだけなら、署名を足さなくてもよい

Chain of Custody



From は Authentic であること

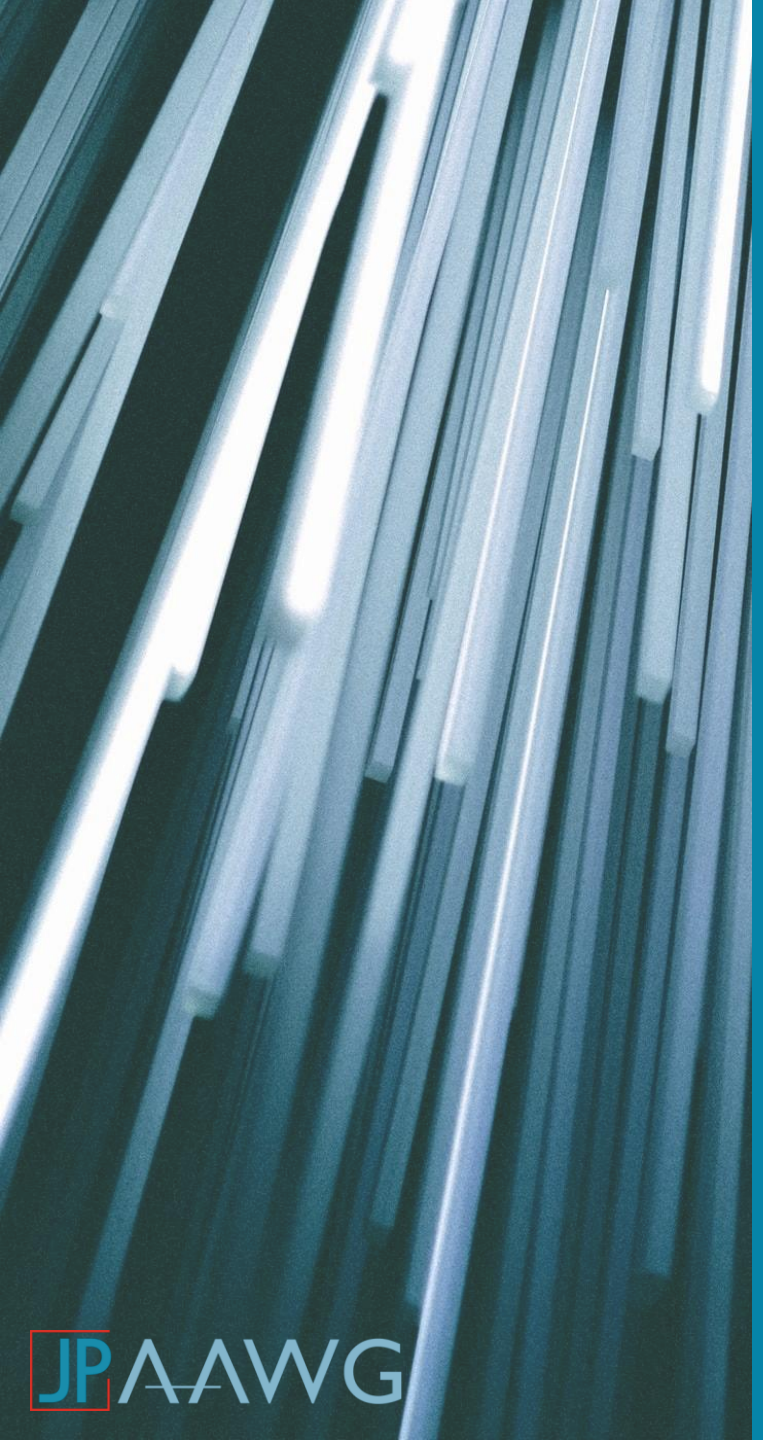
- ヘッダFromのドメインと署名のd= が relaxed domain match
  - From がそのままなら i=1 の署名と比べる
  - 途中で From を書き換えたら、書き換え後の最初の署名と比べる
- DMARC のアラインメントに近い考え方がDKIM2自体に入った
  - ただし DMARC (RFC 9989) で DKIM2 をどう扱うかは未定

検証結果と Provisionally Verified

- 結果は4種類: PASS / FAIL / PERMERROR / TEMPERROR
  - エラー文言まで決められている

    ```
    FAIL: DKIM2-Signature i=<x> RCPT TO <value> did not match
    ```

  - DKIM2ヘッダがない場合はNONE
- Provisionally Verified (仮検証)
  - 最新の署名と最新の Message-Instance だけ見て受け取ってもよい
  - 後でダメと分かったら DSN を返せる
  - 仮検証すら通らないものを受け取ったら、後から DSN を出してはいけない
- SMTP応答: 拒否は 550/5.7.x、DNSの一時エラーは 451/4.7.5
- 正しく署名されていない DKIM2 メッセージは他へ転送してはいけない



Message-Instance ヘッダと Recipe

draft-ietf-dkim-dkim2-spec-09 5章・9章

旧 draft-gondwana-dkim2-modification-alegbra-04 / MailVersion

JP AAWG

変更の記録方法

- ヘッダや本文の変更を記録する方法
- MailVersion ヘッダ → **Message-Instance** ヘッダに改名
- 独自記法のレシピ → JSON の Recipe を base64 にして格納
- Recipe は「受け取った状態から、1つ前の状態を再現する手順」
- 変更した人 (Reviser) が Message-Instance を1つ追加する
- 戻せない本文の変更は null Recipe で宣言
  - 契約に基づくURL書き換えやDLPでの削除など
  - ヘッダの変更は null にできない (必ず戻せること)

Message-Instance ヘッダのタグ

- m: リビジョン番号。1 から連番で、抜けや重複は不可 (必須)
- h: ハッシュアルゴリズム:ヘッダのハッシュ:本文のハッシュ (必須・組み合わせ可)
- r: Recipe。1つ前の状態に戻す手順の JSON を base64 で (option)
  - 戻せない本文の変更は "b": null (null Recipe)
- 知らないタグは無視。同じタグが2回出てきたら無効
- どの DKIM2-Signature の m= からも参照されない Message-Instance があると無効

例: `Message-Instance: m=2; h=sha256:9fWb...:Lx3A...; r=eyJiIjpb...`

本文のレシピ (戻し方)

- {"b": [ステップ, …]}　本文の行は上から 1, 2, 3 …
- {"c": [start, end]}　範囲の行をコピーする
- {"d": ["行", …]}　UTF-8 の行を挿入
- {"u": ["base64", …]}　UTF-8 でない行を decode して挿入
- "b": null　差分は保存しない。元に戻せない

本文501行目にあったURLの行を削除したときのヘッダ

```
{"b":[{"c":[1,500]},
      {"d":["<a href=¥"https://www.example.com¥">Example</a>"]},
      {"c":[501,702]}]}
Message-Instance: m=2; h=sha256:...:...;
    r=eyJiIjpbeyJjIjpbMSw1MDBdfSx7ImQiOlsiPGEgaHJlZj1cImh0dHBz...
```

JSON全体を base64 にして r= に入れる

ヘッダのレシピ (戻し方)

- {"h": {"ヘッダ名(小文字)": [ステップ, …]}}
- ヘッダは下から数える (最後に出てくるものが 1)
- c: [start, end] 番目をコピー / d: 値を挿入 / u: base64 の値を挿入
- 空配列 [] → そのヘッダを全部削除
- ヘッダの変更は null にできない

A replacement for DKIMというSubjectを変更した
Reply-Toヘッダを追加した

```
{"h":{"subject":[{"d":["A replacement for DKIM"]}],
      "reply-to":[]}}
Message-Instance: m=3; h=sha256:...:...;
    r=eyJoIjp7InN1YmplY3QiOlt7ImQiOlsiQSByZXBsYWNl...
```

複数アルゴリズムとハイブリッド

- ハッシュ: sha256 / sha512
- 署名: rsa-sha256 / ed25519-sha256 / mldsa44
- 検証側は5つ全部を実装 MUST
- s= と h= は組み合わせて書ける: 「,」= AND、「|」= OR (「,」が優先)

```
s=k1:rsa-sha256:AAA,k2:mldsa44:BBB|k3:ed25519-sha256:CCC
```

  → (k1 かつ k2) または k3

- 新しい方式は「|」で段階導入、信用しきれないものは「,」でハイブリッドに
- ML-DSA-44: pure ML-DSA、コンテキスト文字列は "DKIM2"
- RSA: 1024bit 以上 (MUST)、2048bit 以上 (SHOULD)
- 未実装・安全でないアルゴリズムや短すぎる鍵は PERMERROR

署名の例 (転送された後)

- origin.example → alias.example → destination.example
- i=1 は「key1 (RSA) か key2 (Ed25519) のどちらかが PASS すればよい」という書き方

```
MAIL FROM: <b@alias.example>
RCPT TO: <c@destination.example>
DATA
DKIM2-Signature: i=2; m=1; t=1792713660; d=alias.example;
    mf=PGJAYWxpYXMuZXhhbXBsZT4=; rt=PGNAZGVzdGluYXRpb24uZXhhbXBsZT4=;
    s=key4321:rsa-sha256:p0Zr...
DKIM2-Signature: i=1; m=1; t=1792713600; d=origin.example;
    mf=PGFAb3JpZ2luLmV4YW1wbGU+; rt=PGJAYWxpYXMuZXhhbXBsZT4=;
    s=key1:rsa-sha256:K7xQ...|key2:ed25519-sha256:Hk2w...
Message-Instance: m=1; h=sha256:9fWb...:Lx3A...
From: <a@origin.example>

A Message!
```

DKIM Replay への対策

- DKIM Replay: 正しい署名の付いた悪いメールを、第三者が大量に送り直す攻撃
  - 署名者が送る数を絞っていても、その制限をすり抜けられてしまう
- Chain of Custody があるので、送り直した側は「自分が中継した」証拠を残すことになる
- 大量のメールボックスを持つ受信者は、同じメッセージが中継経由で何度も届くのを数えられる
  - 「同じ」= m=1 のヘッダと本文のハッシュが一致。署名が2つ以上なら中継経由
  - exploded が無いのに何通も届く → 怪しい
  - exploded があれば、どこが「複数に送った」と言っているのかが分かる
- 予想外の経路 (第三者経由) で届いたかどうかで、普通の一斉配信と見分けられる
- 複数の受信者がそれぞれ転送して、同じところに集まる場合もあるので注意

DSN (バウンス) の作り方

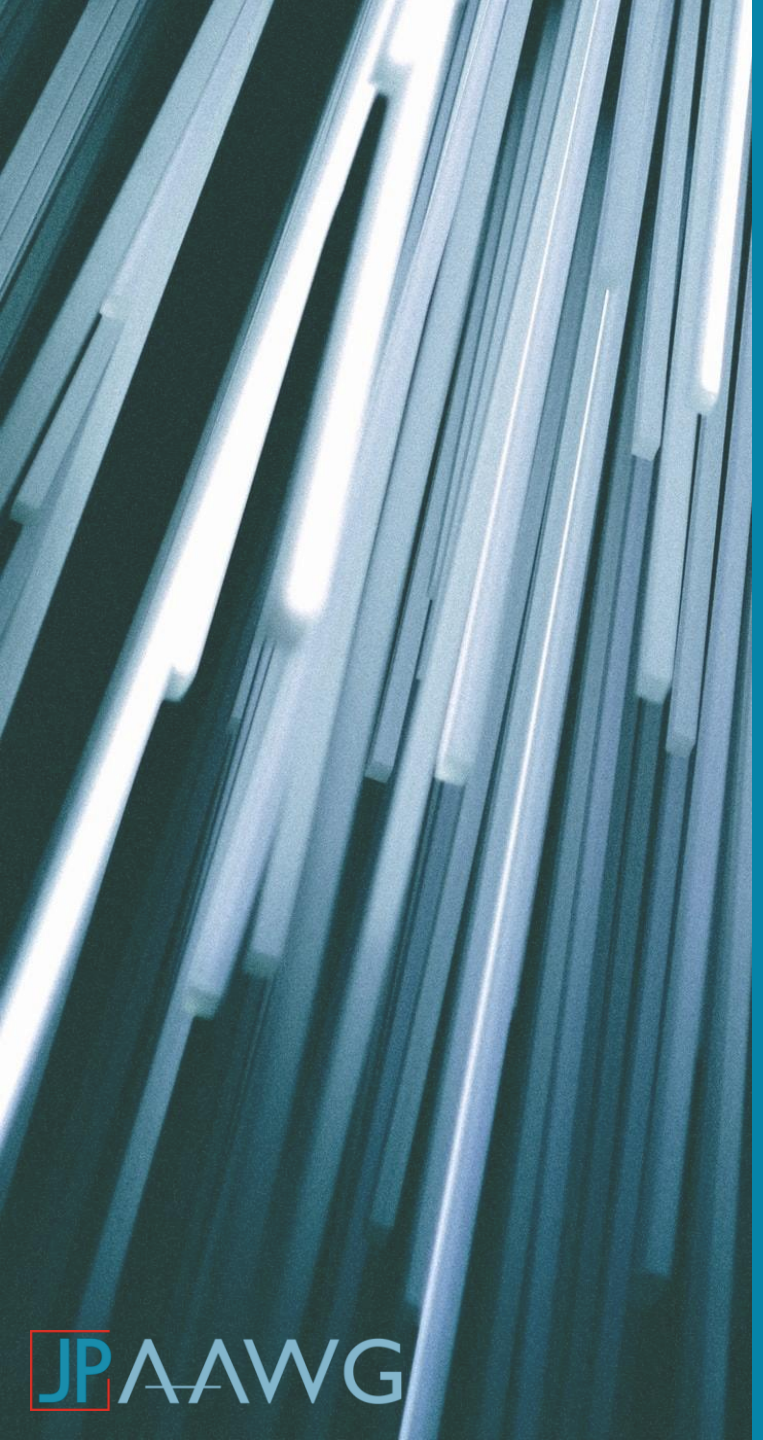
- 配送できないときは SMTP の応答コードで返す。セッション後なら DSN を作る
- 送り先は最新の DKIM2-Signature の mf=。<> なら送ってはいけない (MUST NOT)
  - d= と mf= は一致しているので、仮検証を通ったメッセージの DSN はバックスキャッターにならない
- DSN は必ず3つのパート
  - 人が読む文 / message/delivery-status / 元のメッセージ (message/rfc822) かヘッダだけ (text/rfc822-headers)
- DSN 自身にも Message-Instance と DKIM2-Signature を付ける。MAIL FROM は <>
- DSN の d= は、元のメッセージの最新の rt= の宛先と relaxed domain match (MUST)
- 責任を持てるところまで、来た道を1ホップずつ戻していく

DSN を戻すときと受け取ったとき

- **転送者が DSN を前のホップへ戻すとき (MAY)**
  - 1つ目と2つ目のパートから、転送先の情報を消す (MUST)
  - 3つ目のパートは届いたときの状態に戻す: 自分の変更を元に戻し、その Message-Instance と自分の署名 (nd= 付きも) を消す
  - 自分が足した Received などもできるだけ消す (SHOULD)。null Recipe で本文を戻せなければヘッダだけにする
  - 戻す DSN は新しいメッセージ (署名も Message-Instance も1つだけ)
- **DSN を受け取ったとき (SHOULD)**
  - DSN の i=1 の d= と、返ってきたメッセージの最新の rt= が一致
  - 返ってきたメッセージの最新の署名が自分のものか、ヘッダのハッシュが合うか
  - 失敗したらそれ以上戻さない (MUST NOT)。受け付ける前なら 550/5.7.x、一時的な問題なら 4xx

DKIM1 との違い

- 署名するヘッダ: h= で選ぶ → 原則すべて (例外は固定リスト)
- 正規化: c= で選ぶ → 固定 (本文は simple 相当)
- アルゴリズム: 1つずつ → AND / OR で組み合わせ、ML-DSA-44 も
- 経路: 署名しない → mf= / rt= で署名し、中継ごとにつなぐ
- 改変: 署名が壊れる → Recipe で元に戻して検証
- Replay: 防げない → 宛先が一致しないものは失敗
- バウンス: 署名なし → 署名付きで来た道を戻る
- From との関係: DMARC に任せる → From が Authentic であることを要求
- DNS の鍵: v=DKIM1 のレコードをそのまま使う



DKIM2 における DSN (バウンス)

draft-ietf-dkim-dkim2-spec-09 16章

旧 draft-robinson-dkim2-bounce-processing-01

JPAAWG

DKIM2 における DSN の処理

- DSN は最新の DKIM2-Signature の mf= 宛に送る (<> なら送らない)
- DSN を出せるのは仮検証を通ったメッセージだけ
  - d= と mf= の一致が保証される → **バックスキャッター防止**
- 転送者は受け取った DSN を1つ前のホップへ戻してよい
  - 転送先の情報を消し、自分の変更を戻し、自分の署名を外す
  - 戻す DSN は新しいメッセージ (署名も Message-Instance も1つ)
- 受け取った DSN も、元メッセージが改ざんされていないか検証

例: 転送

- a.comからb.comにメールを送信　※ mf= / rt= は読みやすさのためデコードして表記

```
MAIL FROM: <original@a.com>
RCPT TO: <dest@b.com>
DATA
DKIM2-Signature: i=1; m=1; d=a.com; mf=<original@a.com>; rt=<dest@b.com>; s=...
Message-Instance: m=1; h=sha256:...
From: Sender <original@a.com>
To: <dest@b.com>

I hope this email reaches its destination
```

- b.comがc.comに転送

```
MAIL FROM: <something@b.com>
RCPT TO: <newdest@c.com>
DATA
DKIM2-Signature: i=2; m=1; d=b.com; mf=<something@b.com>; rt=<newdest@c.com>; s=...
DKIM2-Signature: i=1; m=1; d=a.com; mf=<original@a.com>; rt=<dest@b.com>; s=...
Message-Instance: m=1; h=sha256:...
From: Sender <original@a.com>
To: <dest@b.com>

I hope this email reaches its destination
```

例: 転送先から Bounce (c → b)

```
MAIL FROM: <>
RCPT TO: <something@b.com>
DATA
DKIM2-Signature: i=1; m=1; d=c.com; mf=<>; rt=<something@b.com>; s=...
Message-Instance: m=1; h=sha256:...
From: <postmaster@c.com>
To: <something@b.com>
Subject: DSN for ...
Content-Type: multipart/report; boundary="divider43541325151"

--divider43541325151
Content-Type: text/plain

This message is being returned.

--divider43541325151
Content-Type: message/delivery-status

--divider43541325151
Content-Type: message/rfc822

DKIM2-Signature: i=2; m=1; d=b.com; mf=<something@b.com>; rt=<newdest@c.com>; s=...
DKIM2-Signature: i=1; m=1; d=a.com; mf=<original@a.com>; rt=<dest@b.com>; s=...
Message-Instance: m=1; h=sha256:...
From: Sender <original@a.com>
To: <dest@b.com>

I hope this email reaches its destination
--divider43541325151--
```

例: 元の送り先から Bounce (b → a)

```
MAIL FROM: <>
RCPT TO: <original@a.com>
DATA
DKIM2-Signature: i=1; m=1; d=b.com; mf=<>; rt=<original@a.com>; s=...
Message-Instance: m=1; h=sha256:...
From: <postmaster@b.com>
To: <original@a.com>
Subject: DSN for ...
Content-Type: multipart/report; boundary="divider43541325151"

--divider43541325151
Content-Type: text/plain

This message is being returned.

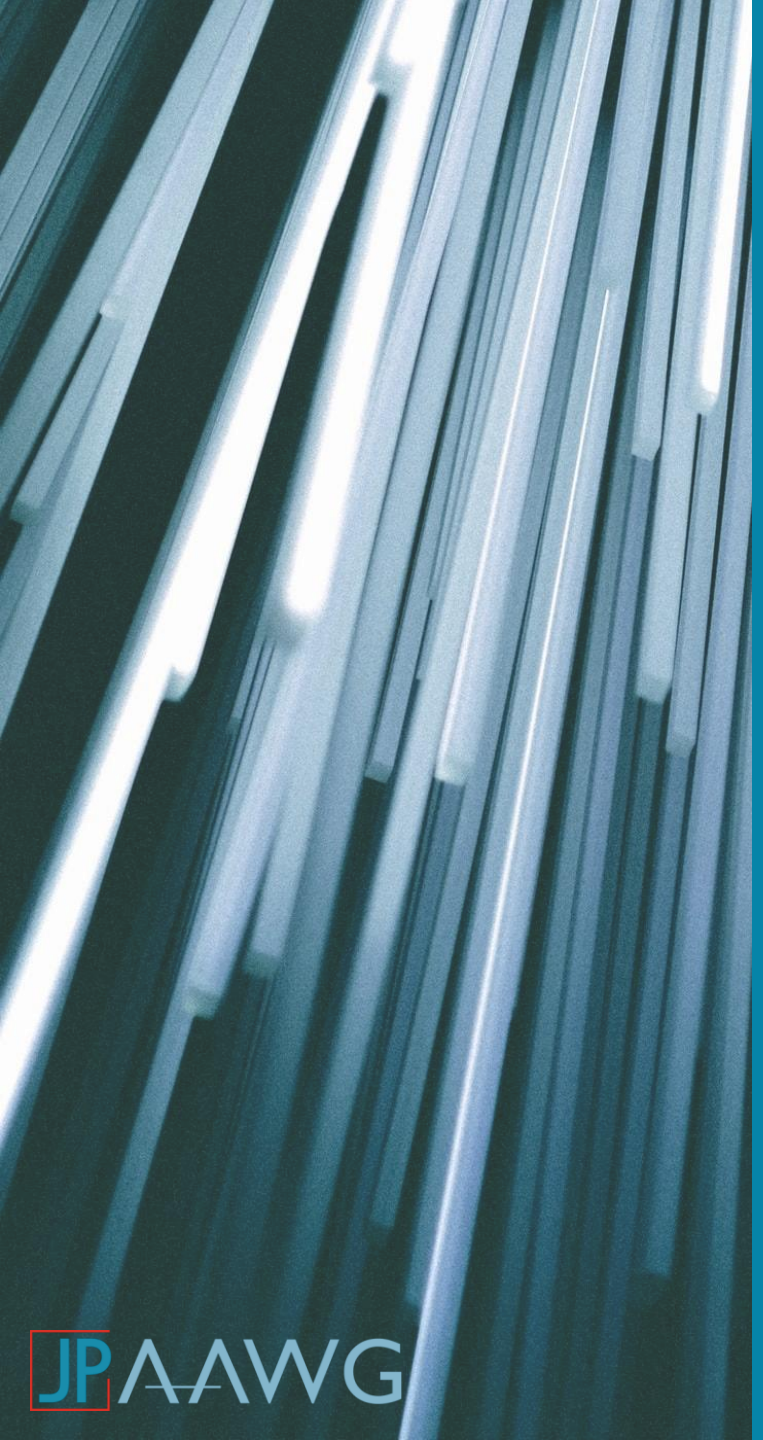
--divider43541325151
Content-Type: message/delivery-status

--divider43541325151
Content-Type: message/rfc822

DKIM2-Signature: i=1; m=1; d=a.com; mf=<original@a.com>; rt=<dest@b.com>; s=...
Message-Instance: m=1; h=sha256:...
From: Sender <original@a.com>
To: <dest@b.com>

I hope this email reaches its destination
--divider43541325151--
```

b.com が付けた i=2 の署名は外して返す



Domain Name Specification for DKIM1 and DKIM2

draft-ietf-dkim-dkim2-dns-03 (2026-11-02)

JP AAWG

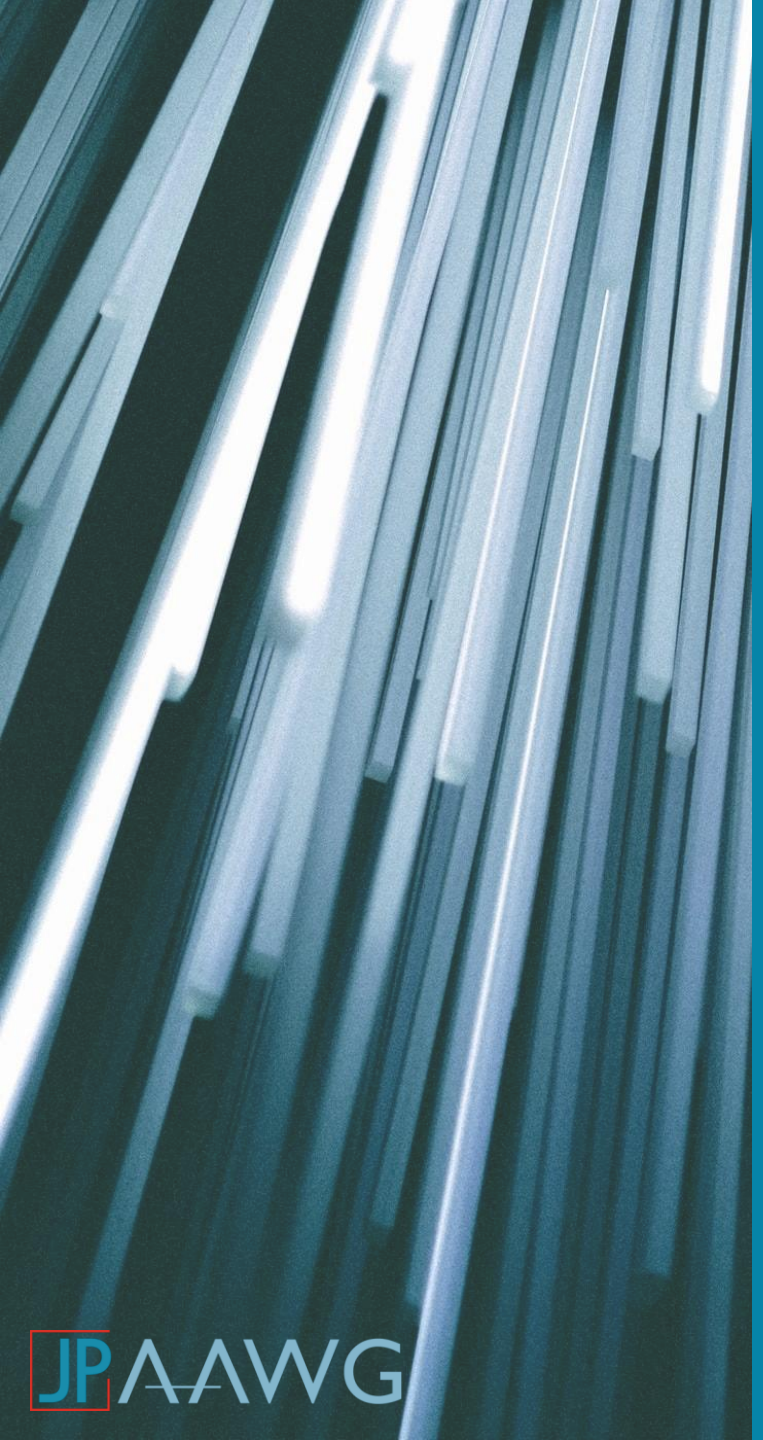
鍵レコード

- <セレクタ>._domainkey.<ドメイン> の TXT
  - v=DKIM1 を推奨 (書くなら先頭)。
- k=: rsa (省略時) / ed25519 / mldsa44。
  - DKIM2 の検証側は3つ全部を実装 (MUST)
- p=: rsa は RSAPublicKey か SPKI、ed25519 は32オクテット、mldsa44 は1312オクテット (ラッパーなし)
  - 長さが違えば不正。空なら失効
- h=, n=, s=, t= は非推奨。DKIM1 でも使わない (SHOULD NOT)、DKIM2 は無視

```
key1._domainkey.example.com.  IN TXT  "v=DKIM1; k=rsa; p=MIIBIjANBgkq..."
key2._domainkey.example.com.  IN TXT  "v=DKIM1; k=ed25519; p=11qYAYKx..."
key3._domainkey.example.com.  IN TXT  "v=DKIM1; k=mldsa44; p=(base64 で 1752 文字)"
```

Domain Policy Record ほか

- **Domain Policy Record (DKIM2 専用、新規)**
  - v=DKIM1 が必須 (先頭)
  - f=strictを書くと、サブドメインがFromをAuthenticにできない
  - そのドメインにだけ効く。上にも下にも継承しない
- **RFC6376(DKIM) の更新**
  - h=, n=, s=, t= を非推奨に
  - rsa の鍵は RSAPublicKey か SPKI のどちらでもよい (errata 3017, 6674, 7001 を解決)
  - DKIM1 以外の v= で始まる TXT は無視。残りが複数ならエラー
- IANA: mldsa44 を登録



DKIM2 Sender Policy

draft-chuang-dkim2-sender-policy-04 (2026-10-19)

JPAAWG

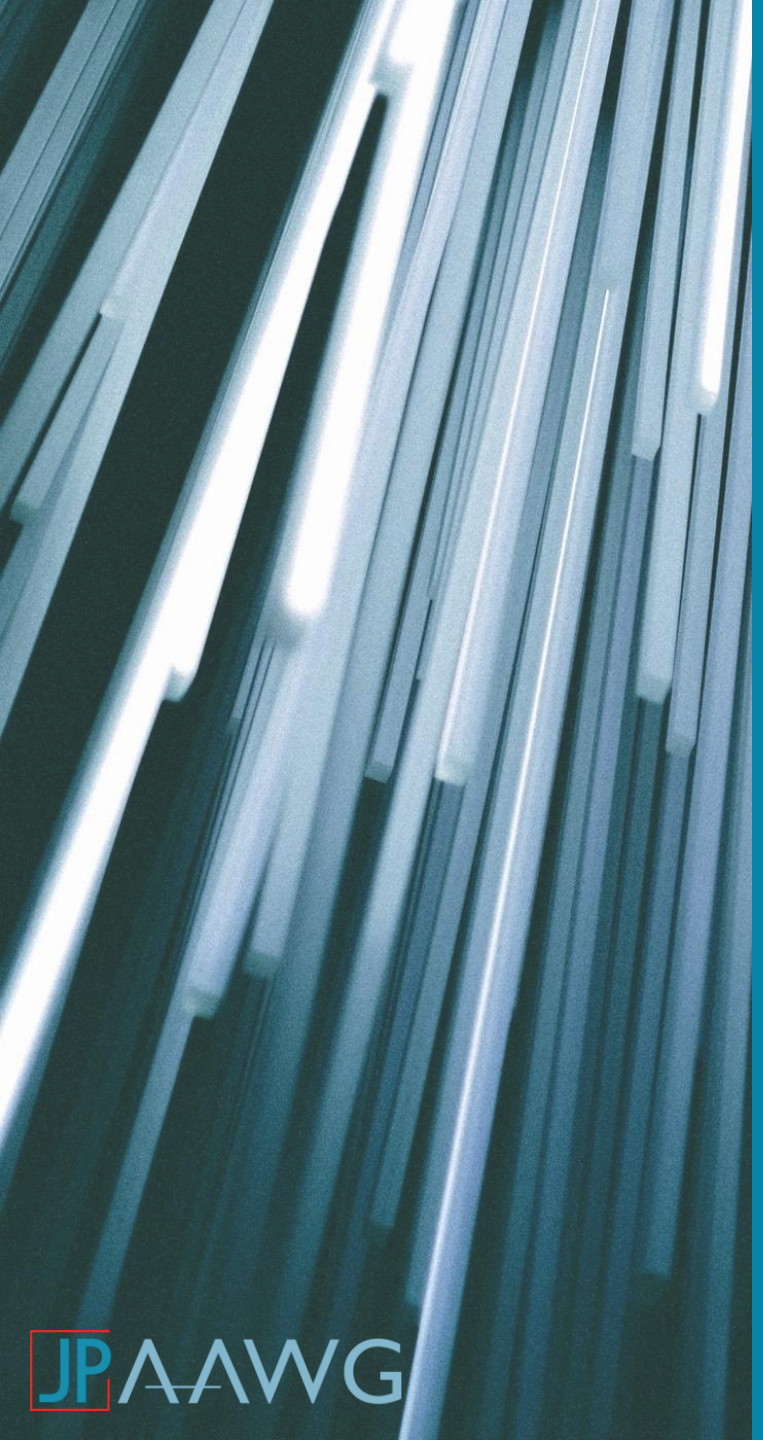
概要

- DMARC (RFC 9989) を DKIM2 に対応させる
  - RFC 9989 の 4.3 節の認証方式に DKIM2 を加える
  - DMARC は送信元しか考えておらず、途中で転送する中継者を想定していない
- ポリシーを2つに分ける
  - ドメイン単位 (処理方針・使う認証方式) → DMARC の DNS レコード
  - メッセージ単位の制約 → DKIM2 署名の f= フラグ
  - DNS はメッセージの外の経路なので、ダウングレードされない
- p= / sp= (reject / quarantine / none) を、アラインメント以外の制約の失敗にも使えるよう一般化
- 参照する DMARC レコードは、i=1 時点の From (Recipe で復元) のドメインのもの (SHOULD)
  - その場合はチェーン全体を検証する (SHOULD)

DMARC レコードの auth= タグ

- 送信・受信の両方で使う認証方式を宣言 (dkim, dkim2, spf のカンマ区切り。省略可)
- 省略時は dkim と spf とみなしてよい。dkim2 は明示的に書く → DKIM2 対応の宣言になる
- 書かれていない方式は評価しない (SHOULD NOT)

```
_dmarc.originator.example:
    v=DMARC1; p=quarantine; auth=dkim2,dkim,spf; rua=mailto:report@originator.example
_dmarc.forwarder.example:
    v=DMARC1; p=reject; auth=dkim2,dkim,spf; rua=mailto:report@forwarder.example
```



A Diagnostic Header Field for DKIM2 Implementations

draft-gondwana-dkim2-debug-header-03
(2026-11-02)

RFC にはならない見込み

JPAAWG

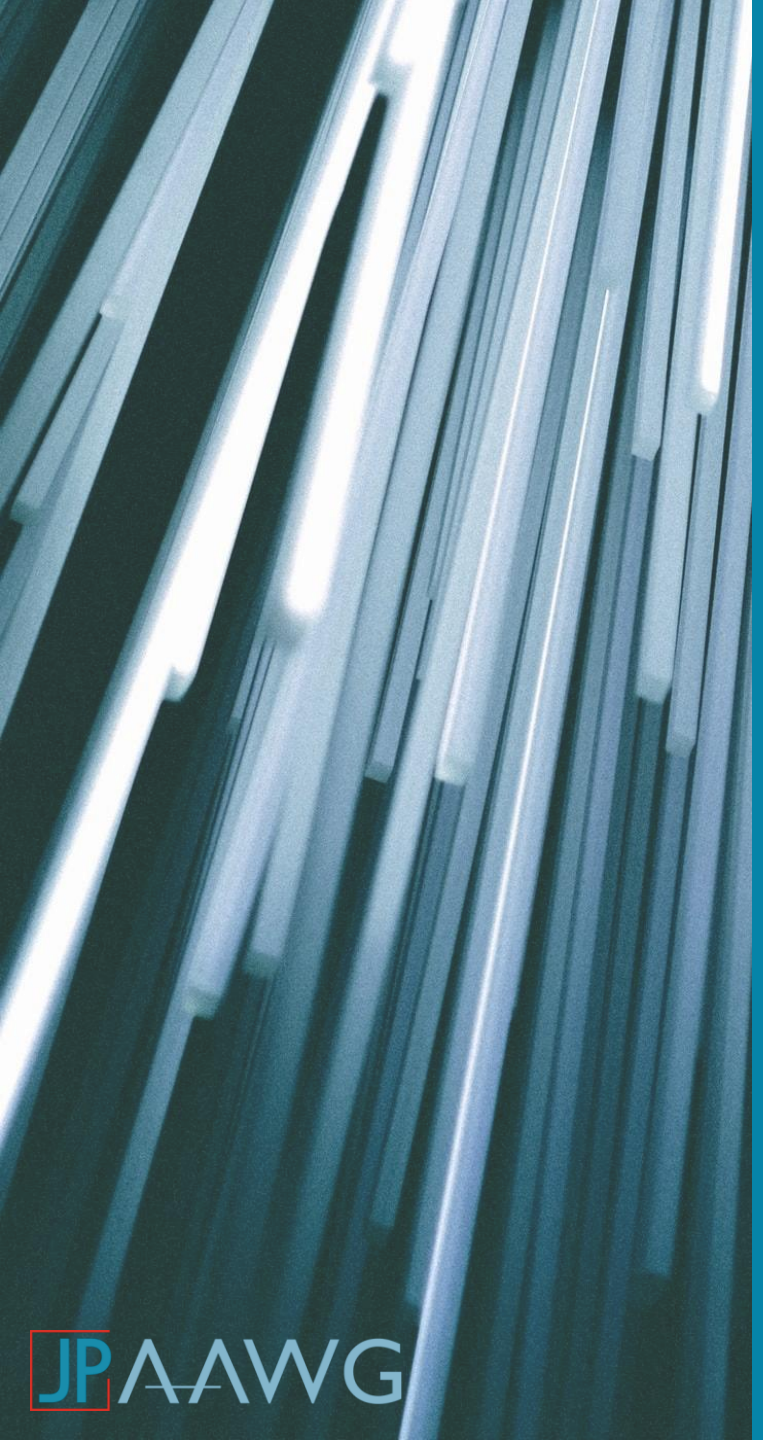
X-DKIM2-Info ヘッダ

- 相互接続テストのためのデバッグ用ヘッダ
  - いくつかのテスト実装がすでに付けているものを、書き方を揃えるために文書化
  - 規範ではない。DKIM2 はこのヘッダに依存しない
- X- で始まるので DKIM2 のヘッダハッシュの対象外
  - いつ誰が足しても、Message-Instance も署名も壊れない
- 検証結果ではない。結果は Authentication-Results に書く
  - 署名もハッシュもされないので、これを見て判断してはいけない (MUST NOT)
- ソフト名やリポジトリ、ヘッダ名が漏れるので、外向きの境界で消してよい

例: メーリングリスト経由

- 受信フィルタ → メーリングリスト → 送信フィルタ の順に追加されていく (上が新しい)

```
X-DKIM2-Info: draft=ietf-dkim-dkim2-spec-09; repo=git.example.net/dkim2;
  date=2026-08-28; sw=outbound-filter; action=sign d=list.example.org a=rsa-sha256;
DKIM2-Signature: i=2; d=list.example.org; ...
X-DKIM2-Info: draft=ietf-dkim-dkim2-spec-09; repo=git.example.org/listmanager;
  date=2026-08-28; sw=listmanager; action=mi-m=2; hc=18;
  hn=archived-at,content-type,date,feedback-id,from,list-archive,...;
Message-Instance: m=2; ...
X-DKIM2-Info: draft=ietf-dkim-dkim2-spec-09; repo=git.example.net/dkim2;
  date=2026-08-28; sw=inbound-filter; action=mi-m=1; hc=8;
  hn=content-type,date,from,message-id,mime-version,subject,to,to;
Message-Instance: m=1; ...
X-DKIM2-Info: ... sw=inbound-filter; action=verify=none (no DKIM2-Signature headers found);
Authentication-Results: mx.example.net; dkim2=none
```



HTTP Mail Transfer Protocol

draft-tishkin-hmtp-01 (2026-10-17)

JP AAWG

HMTP の概要

- HTTPS でメールの投稿と転送をする新しいプロトコル
  - SMTP の拡張ではない。個人提出 (D. Tishkin), Standards Track 志望
- TLS は必須。平文モードはない
- 転送リクエストは送信ドメインがHTTP Message Signatures (RFC 9421) で署名
  - 公開鍵は DKIM と同じく <セレクタ>._domainkey.<ドメイン> に置く
- メッセージ本体(RFC 5322)は改変せずJSONのエンベロープに入れる
  - DKIM 署名や S/MIME, OpenPGP はそのまま有効
- Transfer Identifierで再送しても二重配送しない
- エラーは宛先ごとにProblem Details(RFC 9457)
- 大きな添付は URL + サイズ + ハッシュで参照渡し

SMTP との対応

- MX → SVCB レコード _hmtp.<ドメイン>
- EHLO → /.well-known/hmtp の Capabilities Document
- STARTTLS (任意) → HTTPS (必須)
- ポート 25/587 → 443
- SMTP AUTH → HTTP 認証 (Basic / Bearer)
- MAIL FROM / RCPT TO → エンベロープの "from" / "to"
- DATA / BDAT → "message" (インライン or 参照)
- 応答コード → Problem Details + "smtpStatus"
- Received の "with ESMTP" → "with HMTP"
- ホップごとに SMTP / HMTP を選べる。SMTP へのフォールバックも可
  - ダウングレードを防ぐポリシーも公開できる

HMTP の例: 入口を探す

- DNS の SVCB レコード _hmtp.<ドメイン> で入口のホストを探す (MX の代わり)
  - alpn に h2 が必須。ポートを書かなければ 443。優先度の扱いは MX と同じ
- そのホストの /.well-known/hmtp から Capabilities Document を取得 (認証なし)

```
_hmtp.example.com.  3600 IN SVCB 1 hmtp.provider.example. alpn=h2,h3

GET /.well-known/hmtp HTTP/1.1
Host: hmtp.provider.example

HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: max-age=86400

{ "versions": { "1":{
      "endpoints": {
            "transfer":   { "uri": "/hmtp/v1/transfer", "reports": "/hmtp/v1/reports" },
            "submission": { "uri": "/hmtp/v1/submission",
                      "authentication": ["basic", "bearer"] } } } },
  "limits": { "maxMessageSize": 107374182400, "maxRecipients": 1000 },
  "policy": { "mode": "enforce", "maxAge": 604800 } }
```

HMTP の例: 転送リクエスト

- JSON の envelope (送信元・宛先) と message (RFC 5322 のメッセージを base64) を POST
- Content-Digest と HTTP Message Signatures で送信ドメインが署名。鍵は keyid と domain で _domainkey から
- id は再送しても二重配送しないための Transfer Identifier (128bit 以上の乱数)

```
POST /hmtp/v1/transfer HTTP/1.1
Host: mx.example.net
Content-Type: application/json
Content-Digest: sha-256=:Cw2CRUGnwBRSPpU7qzJnP6Fk2zBdJ8u8mXh7sQe1zYI=:
Signature-Input: hmtp=("@method" "@target-uri" "content-type" "content-digest");
  created=1767268800;expires=1767269100;
  keyid="hmtp2026";domain="example.com";alg="ed25519";tag="hmtp"
Signature: hmtp=:3iXk1Lw3P1pZq9xH0M2eR8vUe6bJ4yD7nT5aC0fK2gS9...:

{ "envelope": {
    "id": "3q2-7wEXAMPLEd9Kc1fQ0g",
    "from": "bounces+7f3a@example.com",
    "to": [ {"address": "bob@example.net"},
            {"address": "carol@example.net"} ] },
  "message": { "data": "RnJvbTogQWxpY2UgPGFsaWNlQGV4YW1wbGUuY29tPg0K..." } }
```

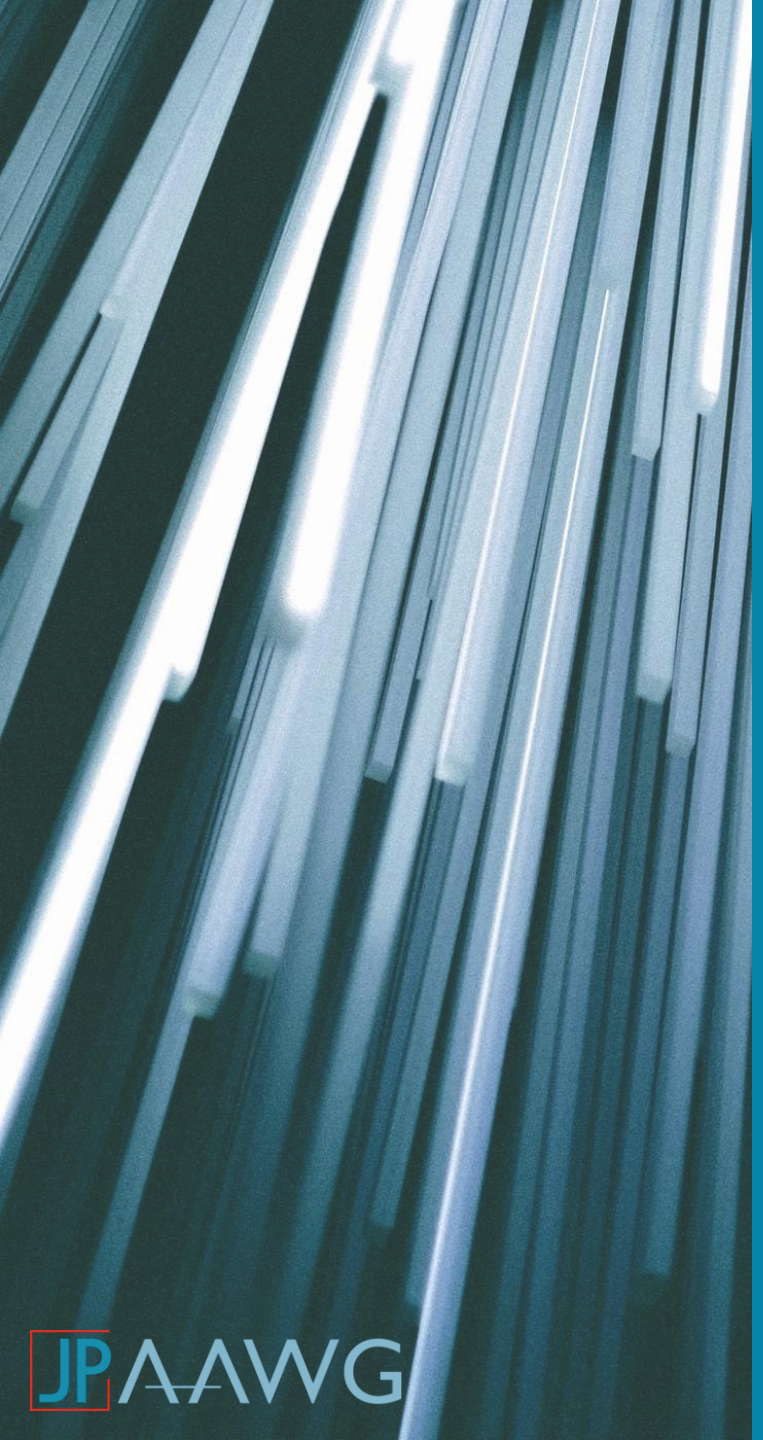
HMTP の例: 応答と Received

- 結果は宛先ごとに accepted / deferred / rejected
- エラーは Problem Details (RFC 9457) で、smtpStatus / smtpReply も付く
- 受け取ったサーバはReceivedを追加 (with HMTP、SubmissionはWith HMTPA)

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "queueId": "4Y7mKq2ZtR",
  "recipients": [
    { "address": "bob@example.net", "result": "accepted" },
    { "address": "carol@example.net", "result": "rejected",
      "problem": {
        "type": "urn:ietf:params:hmtp:error:unknown-recipient",
        "title": "Unknown recipient",
        "smtpStatus": "5.1.1", "smtpReply": 550 } } ] }

Received: from example.com (hmtp-out.example.com [192.0.2.25])
        by mx.example.net with HMTP id 4Y7mKq2ZtR;
        Thu, 01 Jan 2026 12:00:01 +0000
```



Verified Email Identity Framework (VEIF)

draft-uimonen-veif-02 (2026-10-15)

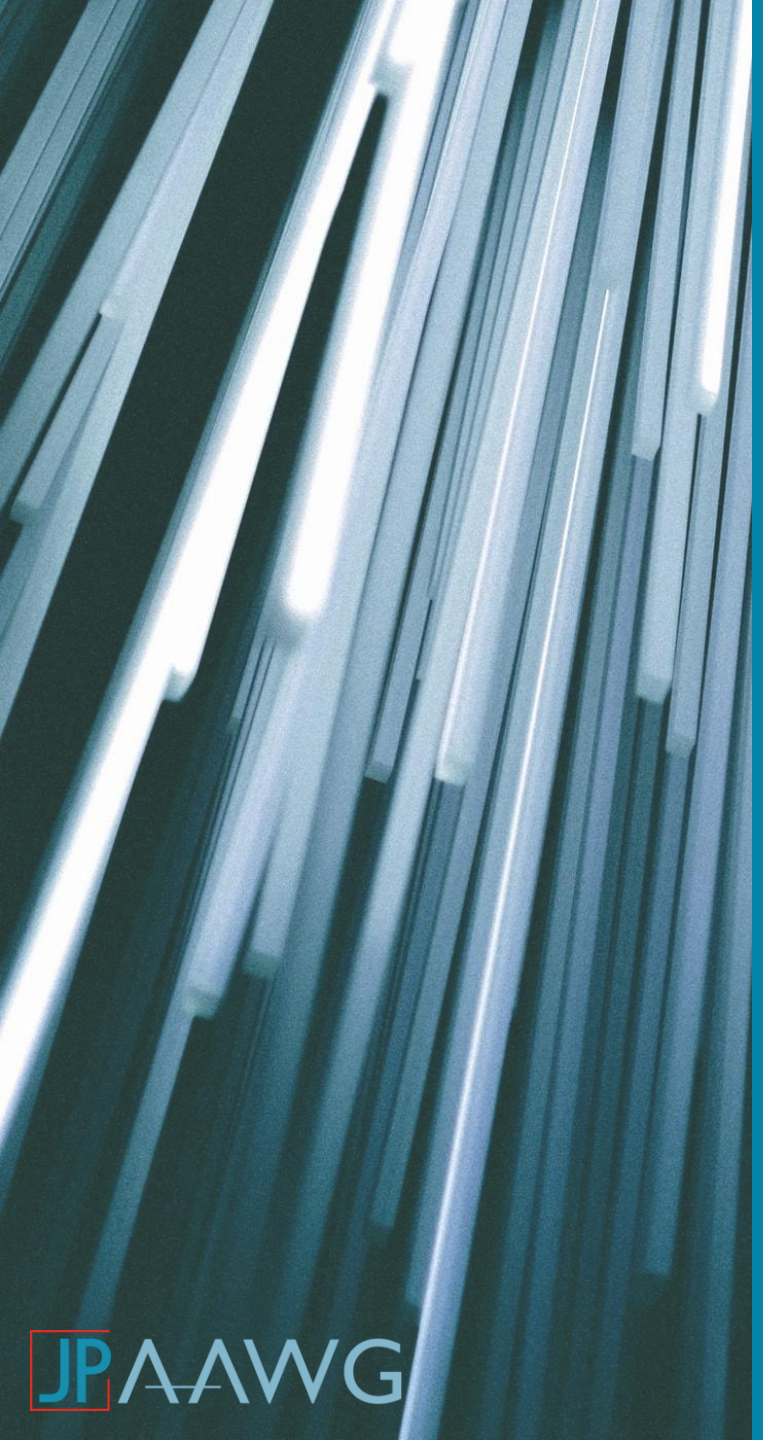
JPAAWG

VEIF の概要

- メールに「実在の人・組織」の身元を暗号的に結びつける
  - SPF / DKIM / DMARC の上に乗る、身元保証のレイヤ
- 身元は IdP (政府機関、通信事業者、認定機関など) が確認
  - IdP が VIA (署名付きの身元アサーション) を発行し、送信者が付ける

```
VEIF-Identity: v=1; org="..."; regid="..."; domain=...; sig=...
```

- 受信側: DKIM 検証 → DMARC アラインメント
  → VIA の署名・失効・トラストアンカーを確認



Aggregate Performance Reporting

draft-ietf-mailmaint-aprf-00 (2026-10-09)

JP AAWG

APRF の概要

- 受け取った後どうなったか (受信箱か迷惑メールか、開封などの統計情報) を集約レポートで送る
- DMARCと違いFromではなく、DKIM署名のd=とセレクタに紐づく
- レポートは JSON、UTC の1日単位、メールで送る (gzip可)
- 少ない件数は丸めるなど、プライバシーに配慮

```
sel1._aprf._domainkey.example.org.  IN TXT
    "v=APRFv1; rua=mailto:reports@example.org;"
*._aprf._domainkey.example.org.     IN TXT  ...   ; 全セレクタ
```

APRF のレポートの例

- UTC の1日ごとに JSON。header に送り元、DKIM の d= とセレクタ、期間 (UNIX 時間)
- classification (振り分け) と engagement (反応) は丸めた件数
- 件名と添付ファイル名: APRF: <yyyymmdd>_<DKIM ドメイン>_<セレクタ>_<source>

```
Subject: APRF: 20240229_example.com_sel1_ReceiverMBP,Inc.

[ { "header": {
      "version": 4,
      "source": "Receiver MBP, Inc.",
      "dkim_domain": "example.com",
      "dkim_selector": "sel1",
      "report_start": 1709164800,       ← 2024-02-29 00:00:00 UTC
      "report_end": 1709251199,         ← 2024-02-29 23:59:59 UTC
      "contact_info": "reports@mbp.net",
      "extra_info": "TBD",
      "sdi_used": "N/F" },              ← sdi が DNS に無い
    "body": [ {
      "classification": { "inbox": 10000, "unwanted": 100 },
      "engagement": { "positive": 200, "negative": 100, "neutral": 20 } } ] } ]
```

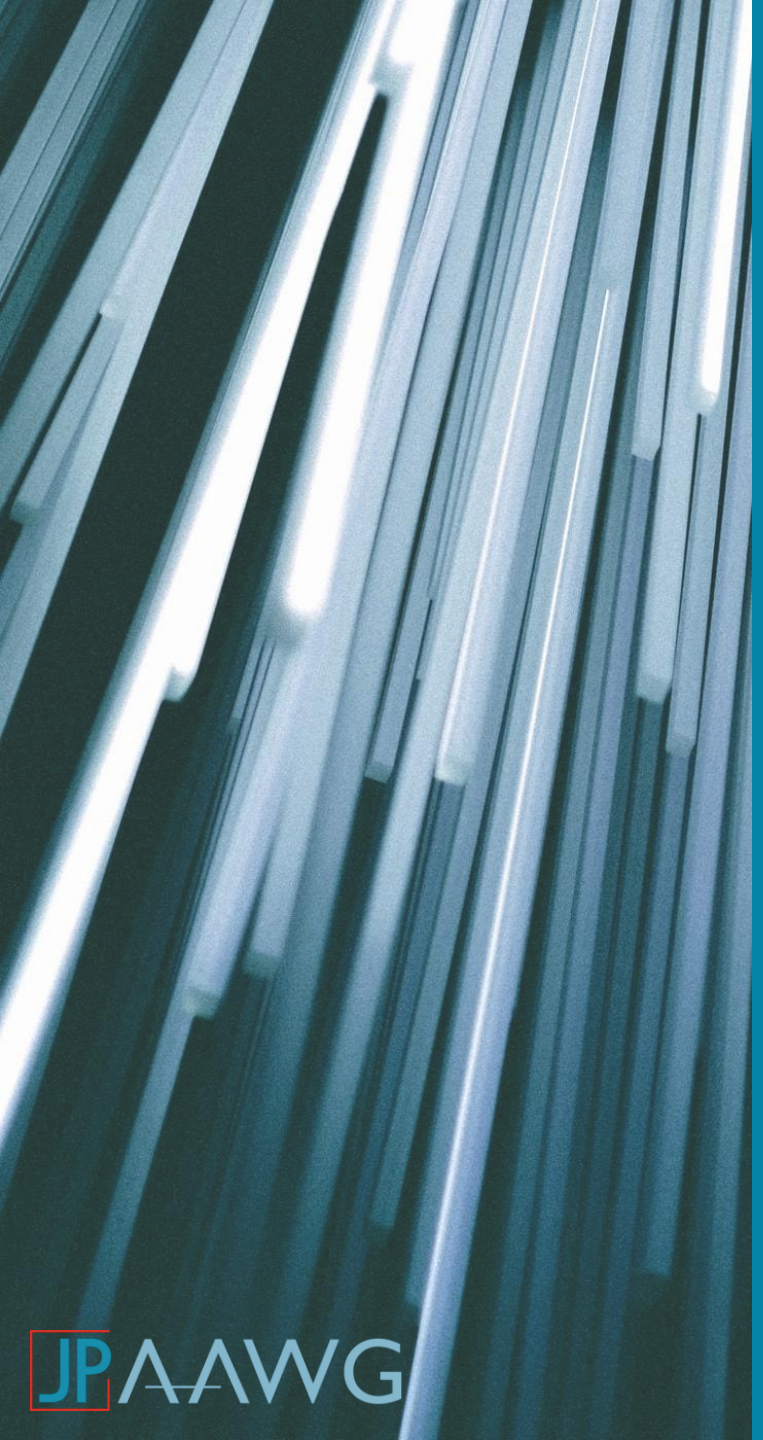
positive: 開封、正常メール報告
negative: スパム報告、開封せず削除、購読解除
neutral: 転送した、フォルダに保存

APRF のレポート例 (SDI)

- DNS の sdi= に、分類に使うヘッダ名と区切り文字を書く
  - ヘッダは DKIM で署名する。右ほど細かく、最大4階層
- 受信側は segment ごとに集計 (使うかは受信側の自由)

```
selector1._aprf._domainkey.example.com. TXT
    "v=APRFv1;rua=mailto:reports@example.com;sdi=UniqueHeaderName,^;"
UniqueHeaderName: Seg1^Seg2^Seg3          ← 送るメールに付けるヘッダ (例)

[ { "header": { "version": 42, "source": "Receiver MBP, Inc.",
      "dkim_domain": "example.com", "dkim_selector": "selector1",
      "report_start": 1709164800, "report_end": 1709251199,
      "sdi_used": "UniqueHeaderName,^" },
    "body": [
      { "segment": ["Seg1", "Seg2", "Seg3"],
        "classification": { "inbox": 10000, "unwanted": 500 },
        "engagement": { "positive": 300, "negative": 200, "neutral": 50 } },
      { "segment": ["Seg1", "Other"],
        "classification": { "inbox": 50, "unwanted": 50 },
        "engagement": { "positive": 50, "negative": 50, "neutral": 50 } } ] } ]
```



目立たない

Unobtrusive End-to-End Email Signatures

draft-ietf-mailmaint-unobtrusive-signatures-02
(2026-04-30)

JP AAWG

Unobtrusive Signatures の概要

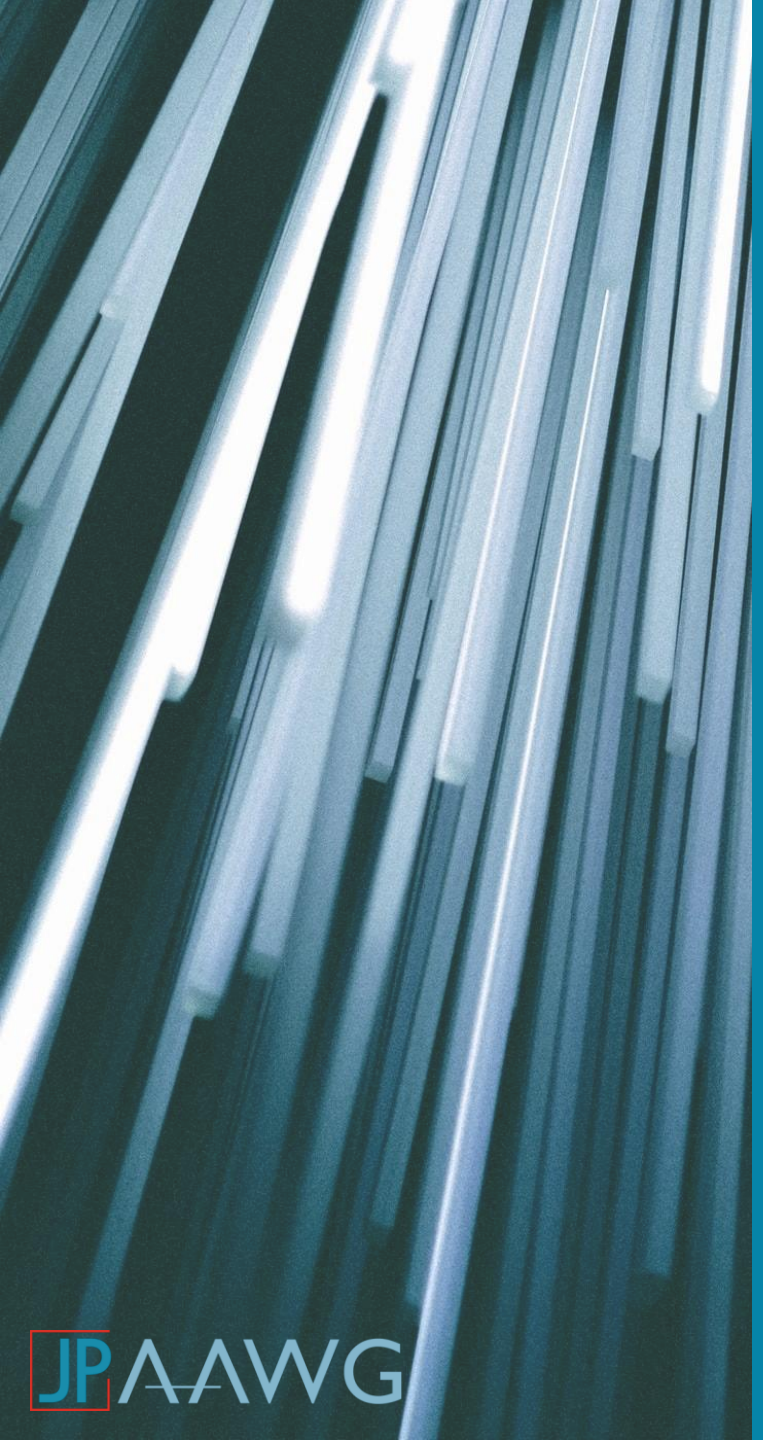
- 署名だけ(暗号化なし)のメールのための、新しいE2E署名の形
- 今の S/MIME, PGP/MIME の困りごと
  - 未対応のメーラーでは読めない (PKCS#7 signed-data)
  - 署名が謎の添付ファイルに見える (「添付が開けない」と言われる)
  - 途中で改変されると署名エラーの警告が出る
  - → 面倒なので署名しなくなる
- 未対応のメーラーでは普通のメールに見えるようにして、全メールの署名を普及させたい
- OpenPGP と CMS (S/MIME) の両方に対応。

Unobtrusive Signatures の仕組み

- 外側は子パートが1つだけの multipart/mixed。その先頭にSigヘッダ
- ヘッダ保護(RFC 9788)が必須。正規化はDKIMの simple 相当
- 壊れた署名は「署名なし」と同じ扱い。警告は出さない
- MTA は子パート1つの multipart/mixed を改変してはいけない

```
Content-Type: multipart/mixed; boundary="5d6"
From: Alice Lovelace <alice@openpgp.example>
Subject: This is a Test

--5d6
Sig: t=p; b=wnUEABYKAB0WIQTrhbtfozp14V6UTmPyMVUMT0fjjgUCaBQq...
From: Alice Lovelace <alice@openpgp.example>
Subject: This is a Test
Content-Type: multipart/alternative; boundary="913"; hp="clear"
  ... 本文 ...
--5d6--
```



Mail Action Protocol (MAP)

draft-mailschema-mail-action-protocol-00 (2026-10-08)

JP AAWG

Mail Action Protocol の概要

- エージェントやサービスがメールで「操作」をやり取りするための仕組み
  - 承認依頼、内容のレビュー、アカウントのセキュリティ対応など
- 読める本文の後ろに、操作を記述した JSON のパートを付ける
  - Structured Email の形で multipart/related に入れる
- 操作の型はレジストリで定義。契約は RFC8785 で正規化した JSON のダイジェストで固定
- Ruby / Python / Go / Rust / JavaScript のライブラリがある

処理の流れ

- メールに書かれた内容は「予告」。判断に使うのはサービスから読み直した内容

① サービス ―メール―▶ 受信者
　　読める本文 + MAP の JSON パート、From と揃った DKIM 署名
② メーラー / エージェント: メールを確認
　　DKIM (本文全体を署名、l= なし)、自分宛てか、MAP パートは1つだけか
③ 型の約束を手元のカタログから取り出し、ダイジェストを確認
　　メールに書かれた URL を取りに行ったりはしない
④ コネクタでサービスに最新の提案を問い合わせ (readback)
　　提案が存在する・自分に出されている・期限内・条件のバージョンが一致
⑤ ポリシーを当てはめ、必要なら人に確認 → 「承認」
⑥ サービスの既存 API に、条件のバージョン付きで送る
　　サービスが権限・期限・バージョンをまとめて確認してから記録

例: 記事の公開を承認してもらうメール

- 人が読む本文と、機械が読む JSON を multipart/related に並べる (Structured Email の形)
- MAP のパートはメール1通に1つだけ。添付ファイルや転送されたメールの中のものは無効

```
From: Docs Service <noreply@service.example>
To: owner@example.net
Subject: 「Getting started」の公開の承認をお願いします
DKIM-Signature: ... d=service.example; ...   ← From と揃った署名が必須
Content-Type: multipart/related; boundary="b1"

--b1
Content-Type: text/html; charset=utf-8
   (人が読む本文。同じ内容と、レビューページへのリンク)
--b1
Content-Type: application/ld+json;
    profile="https://mailschema.org/profiles/map/0.3"
Content-Purpose: Machine-readable
Content-Transfer-Encoding: base64
   (次のページの JSON)
--b1--
```

例: MAP の JSON

- 例 (Publication Approval) を短くしたもの。

```
{ "@type": "MailAction",
  "type": { "id": "https://mailschema.org/types/publication-approval",
            "version": "0.1", "contractDigest": "sha-256:de91a49d..." },  ← 型の約束
  "@id": "urn:uuid:0199a6c2-4b1e-...",                          ← このやりとりの ID
  "issuedAt": "2026-10-02T08:00:00Z", "expiresAt": "2026-10-03T08:00:00Z",
  "service": { "id": "https://service.example", "tenant": "urn:example:tenant:7" },
  "recipient": "owner@example.net",
  "subject": { "id": "urn:example:object:42", "title": "Publish the getting started guide" },
  "terms": { "id": "urn:example:proposal:p7", "version": "p7-r4" },   ← 条件の版
  "human": { "url": "https://service.example/review/p7" },    ← 人が見るページ
  "details": {
    "content": { "id": "urn:example:content:42", "revision": "r4", "title": "Getting started" },
    "destinations": [ { "id": "https://docs.service.example/guide", "visibility": "public" } ],
    "schedule": { "mode": "immediate" } },
  "operations": [ {"id": "approve"}, {"id": "decline"}, {"id": "request-changes"} ] }
```



Structured Quoted Content

draft-ietf-sml-structured-quoted-content-00 (2026-07-06)

JP AAWG

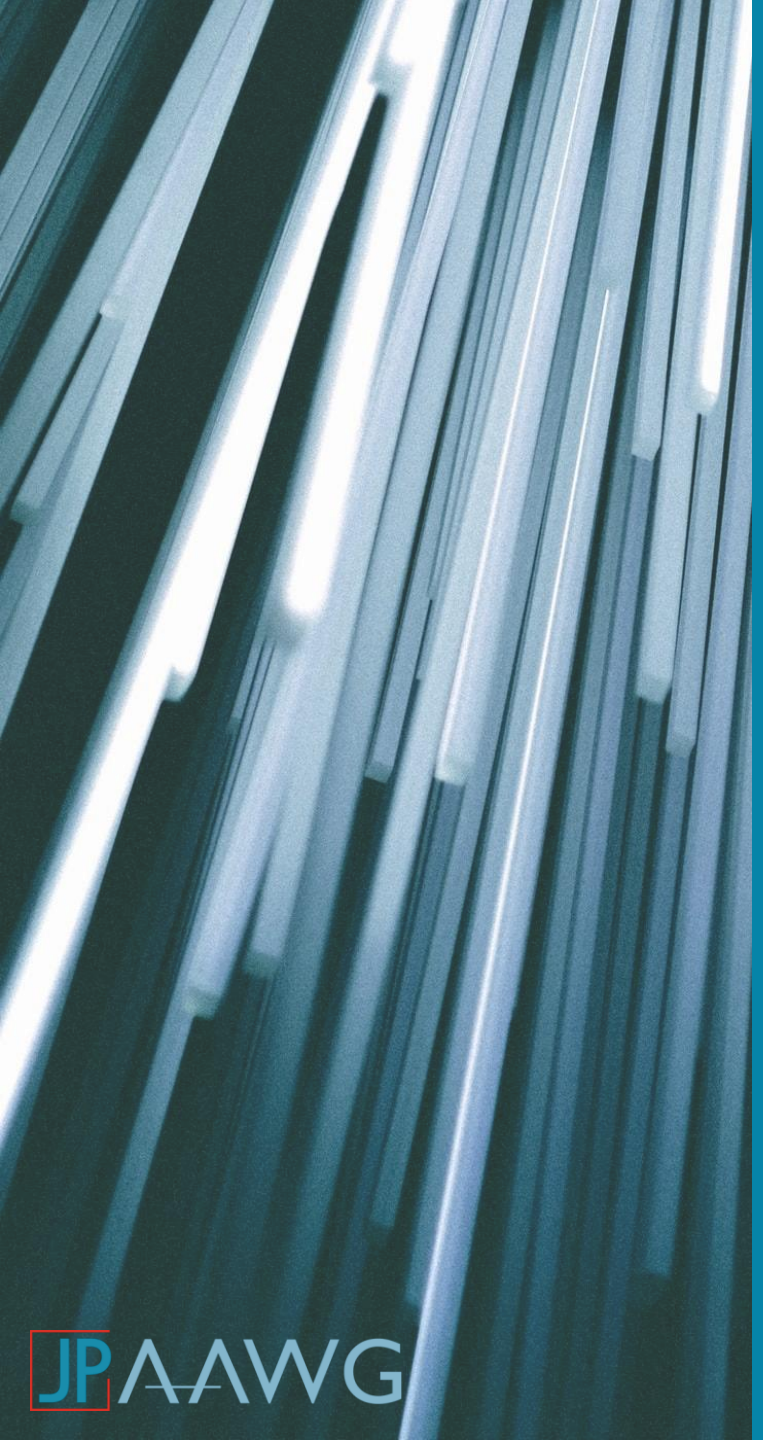
Structured Quoted Content

- 返信や転送の「引用部分」を機械可読にする
- Structured Email (JSON-LD) の Partial representation を使う
- HTML の各ブロックに data-id を付け、JSON-LD のオブジェクトと結びつける
  - EmailMessage, QuotedContent, QuotedHeader, QuoteAttribution, TextContent, ForwardPreamble など
- ネストした引用や、引用の間に書いた返信も表現できる
- HTML のみ
- kmail で実装中
- 非対応のメーラーが途中に入ったら、最新の連続した部分だけを構造化引用として扱う

例: 引用を含む転送

- MIME 構造: multipart/related の中に text/html と application/ld+json
- HTML 側は data-source-id で JSON-LD パートを、data-id でオブジェクトを指定する

```
<!-- Carol's note -->
<div data-source-id="cid:sml-ex1@example.com"
     data-id="quoted-content:A1000001-...">
  <div data-source-id="cid:sml-ex1@example.com"
       data-id="forward-preamble:A1000006-...">
    ---------- Forwarded message ---------
  </div>
  ...
  <blockquote>
    <div data-source-id="cid:sml-ex1@example.com"
         data-id="text-content:A1000004-...">
      <!-- Bob's reply text -->
    </div>
  ...
```



Email AI-Processing Preference Signal

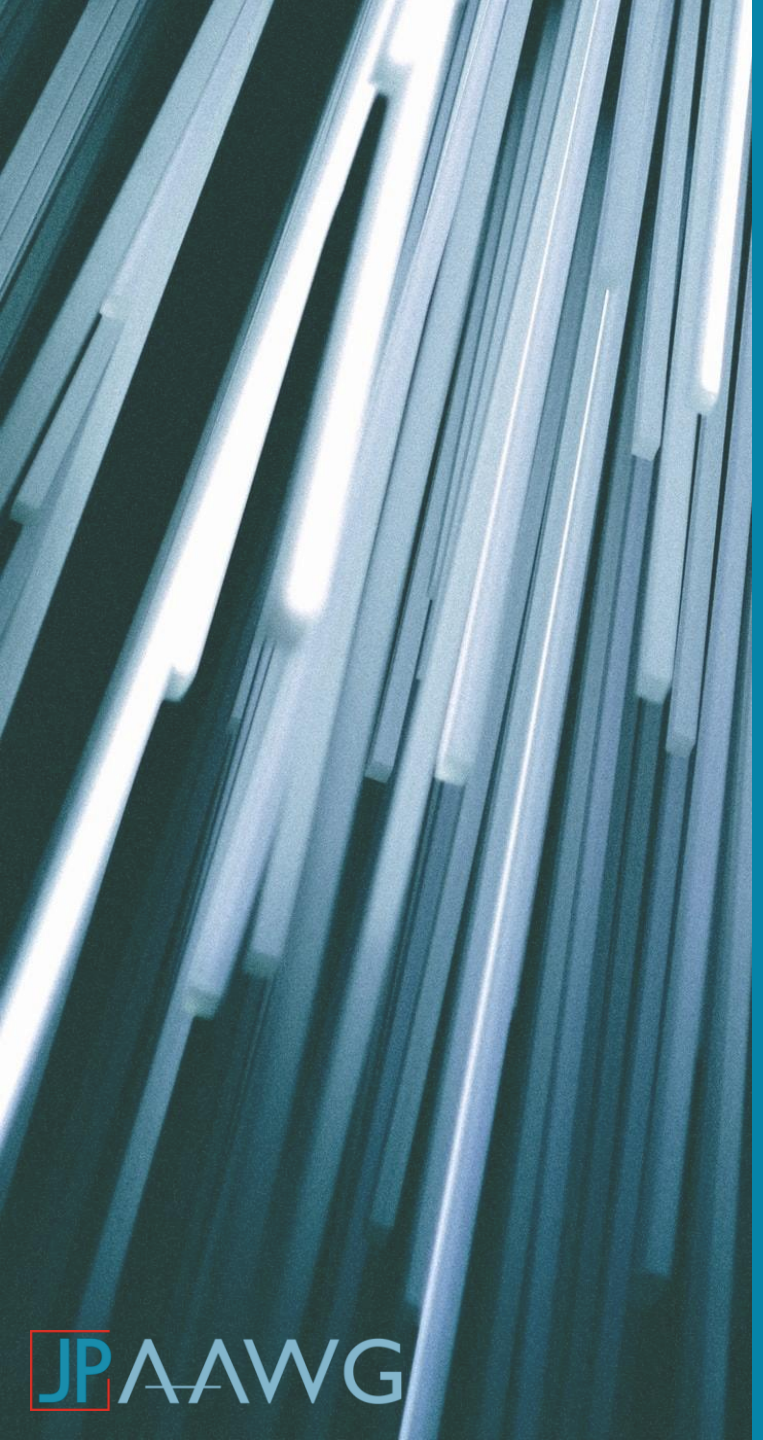
draft-nault-email-ai-signal-00 (2026-10-07)

JP AAWG

AI 処理の希望を伝えるヘッダ

- 送信者が「AI でどう処理してほしいか」をヘッダで表明
  - assist (AI アシスト), memory (AI の記憶), profile (プロファイリング), train (学習), external (外部事業者での処理)
- ヘッダが無い = 希望なし。処理してよいかは受信側のポリシー次第
- 転送や返信でも、引用した内容と一緒に引き継ぐ
- 法的な同意ではなく、受信者が従う保証もない。普通の配送・保存・人が読むことは妨げない

```
AI-Processing-Preference: v=1; assist=ask; memory=no;
    profile=no; train=no; external=no
```



Secure SMTP/TLS SRV Announcement

draft-nurpmeso-smtp-tls-srv-08 (2026-8-04)

draft-nurpmeso-smtp-tls-srv-06 (2025-7-20)から更新

JP AAWG

Secure SMTP/TLS SRV Announcement

- MXではなくSRVレコードを使う
- Port25はSTART TLS、Port842はImplicit TLS(暗黙的なTLS)

SRVレコード

- RFC2782 (2000/2月)
- ホスト名: _サービス名._プロトコル.名前
- 内容: 優先度　重み　ポート　対象サーバー

```
_abc._tcp.example.jp SRV  10  30  8080  api.example.jp.
                     SRV  10  70  8081  api.example.jp.
                     SRV  20   0  8080  api_back.example.jp.
```

Secure SMTP/TLSの仕組み

- SRVレコードのサービス名は「_smtp-tls」
- portが25の場合はSTART TLS
- portが842の場合はImplicit TLS(暗黙のTLS)
  - それ以外は25と同じ扱い
- SRVレコードがあるときはMXは調べない

START TLSの場合

```
_smtp-tls._tcp.example.jp SRV  0  0  25  mail.example.jp
```

Implicit TLSの場合

```
_smtp-tls._tcp.example.jp SRV  0  0  842  mail.example.jp
```



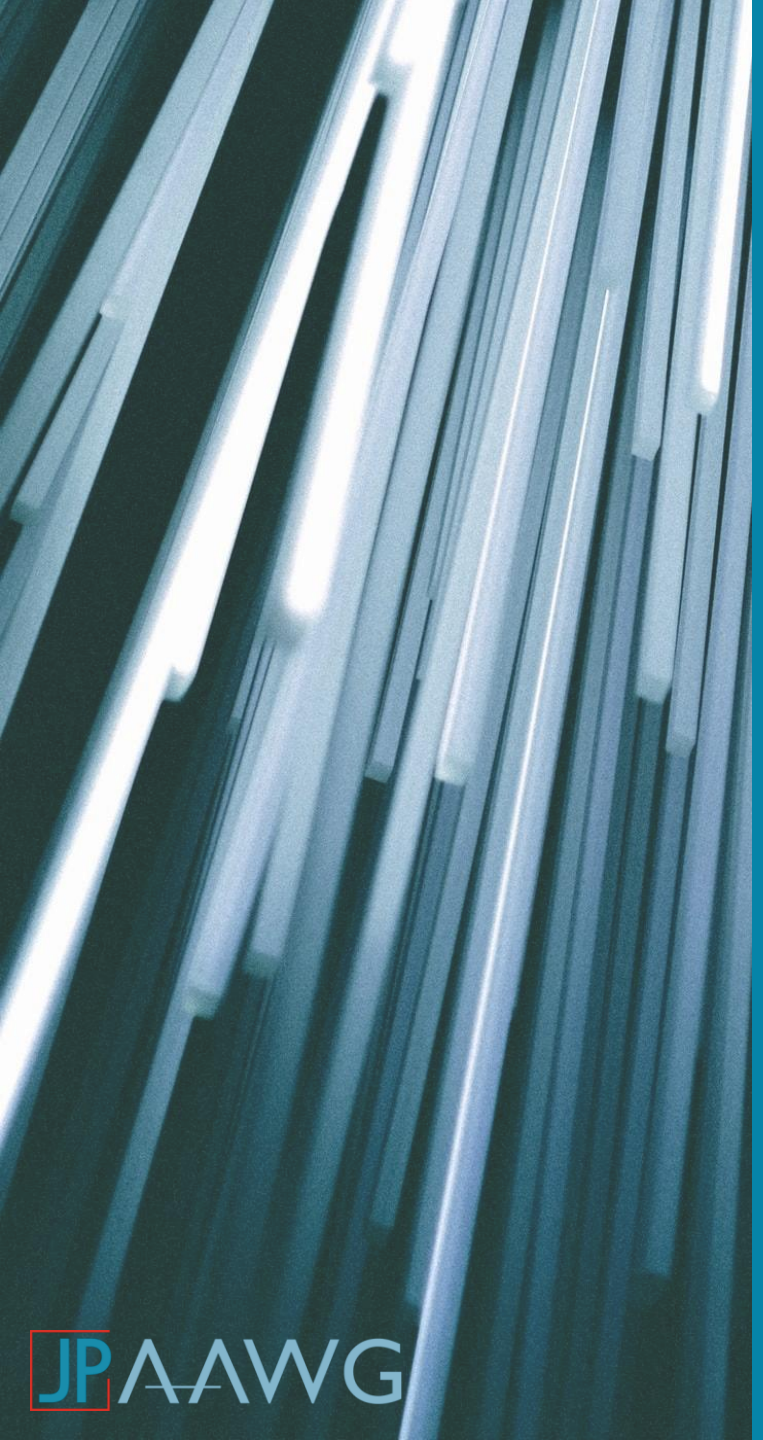
RFC5321bis

draft-ietf-emailcore-rfc5321bis-44 (2025-7-31)

JP AAWG

RFC5321からの変更点

- 基本的な動作は変わっていない
- 構文定義(ABNF)の修正・更新・明確化
- Source Routeに関する説明を削除
  - @example1.jp,@example2.jp:hirano@example3.jp
- 索引などの追加
- EHLOが複数来た場合や引数の明確化
- MTAとMSAの区別
- EHLO, HELOの引数の後ろに余分なテキストを付けることを禁止 (SHOULD NOT → MUST NOT)
  - EHLO localhost abcdefg 



RFC5322bis

draft-ietf-emailcore-rfc5322bis-12 (2025-10-3)

JPAAWG

Appendix A.5 空白やコメントの例

RFC5321

```
From: Pete(A nice \) chap) <pete(his account)@silly.test(his host)>
To:A Group(Some people)
    :Chris Jones <c@(Chris's host.)public.example>,
       joe@example.org,
 John <jdoe@one.test> (my dear friend); (the end of the group)
Cc:(Empty list)(start)Hidden recipients  :(nobody(that I know))  ;
Date: Thu,
     13
      Feb
        1969
     23:32
           -0330 (Newfoundland Time)
Message-ID:          <testabcd.1234@silly.test>

Testing.
```

RFC5321bis

```
From: Pete(A nice \) chap) <pete@silly.test(his host is silly)>
To:A Group(Some people)
    :Ed Jones <e@a.test(.host of Ed)>,
       one@y.test,
 John <jdoe@one.test> (my dear friend); (the end of the group)
Cc:(Empty list)(start)Hidden recipients  :(nobody(that I know))  ;
Date: Thu,
     13
      Feb
        1969
     23:32
           -0330 (Newfoundland Time)
Message-ID:          <testabcd.1234@silly.test>

Testing.
```

Received:で空のトークンに空白を許可

- RFC5322
  - received = "Received:" *received-token ";" date-time CRLF

Received:; Mon, 02 Jan 2006 15:04:05 -0700 ○

Received: ; Mon, 02 Jan 2006 15:04:05 -0700 ✖

- RFC5322bis
  - received = "Received:“
    [1*received-token / CFWS] ";" date-time CRLF

Received:; Mon, 02 Jan 2006 15:04:05 -0700 ○

Received: ; Mon, 02 Jan 2006 15:04:05 -0700 ○

Message-Idの<>の中に名前が付いた

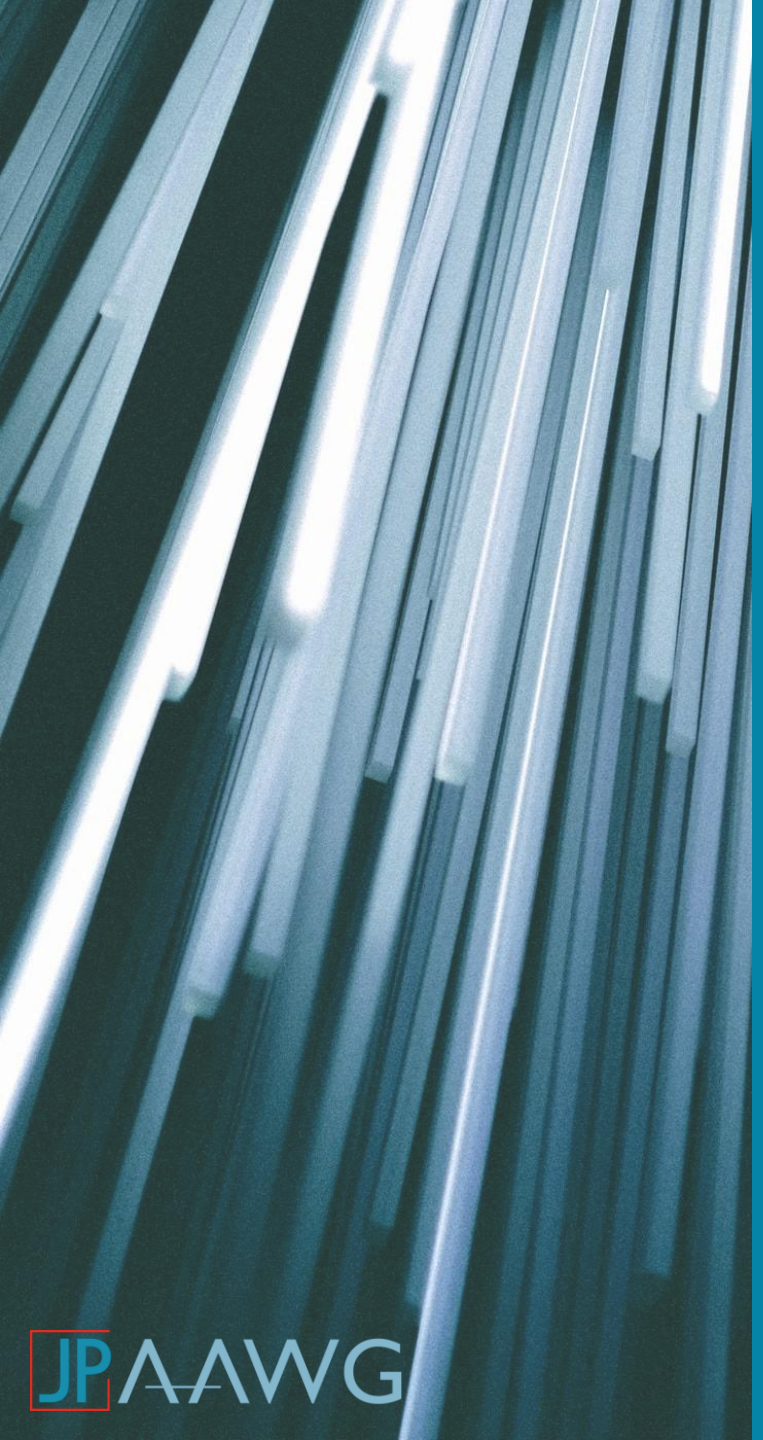
- RFC5322

  msg-id = [CFWS] "<" id-left "@" id-right ">" [CFWS]

- RFC5322bis
  - msg-id = [CFWS] "<" msg-id-internal ">" [CFWS]
    msg-id-internal = id-left "@" id-right

その他RFC5322からの変更点

- 基本的な動作は変わっていない
- タイムゾーンの表記からGMTを排除してUTCに統一
- 「printable」は空白を含む、「VCHAR」は空白を含まない
- その他細かいのたくさん



SMTP Service Extension for Client Identity

draft-storey-smtp-client-id-19 (2025-5-30)

JP AAWG

クライアント認証の追加

- SMTP認証済みクライアントを悪用したなりすましメールなどを防ぎたい
- モバイル端末内にあるSMTPクライアントなどは、ネットワークアドレスやEHLOの値など役に立たない
- SMTPを拡張して、SMTPクライアントが一意な識別子を送信できるようにする
- 識別子を元にSMTPサーバーは判断、制御できる

仕組み

- EHLOの応答: CLIENTID
  - TLS/START TLSが確立している場合のみ送信
- SMTPコマンド
  - CLIENTID　<client-id-type>　<client-id-token>
  - client-id-type: UUID、ライセンス、デバイスID、MACアドレスなど
- CLIENTIDコマンドはAUTHの前でのみ使用可能
- CLIENTIDは1度しか送信できない
- RSETでCLIENTIDは破棄される

ユースケース

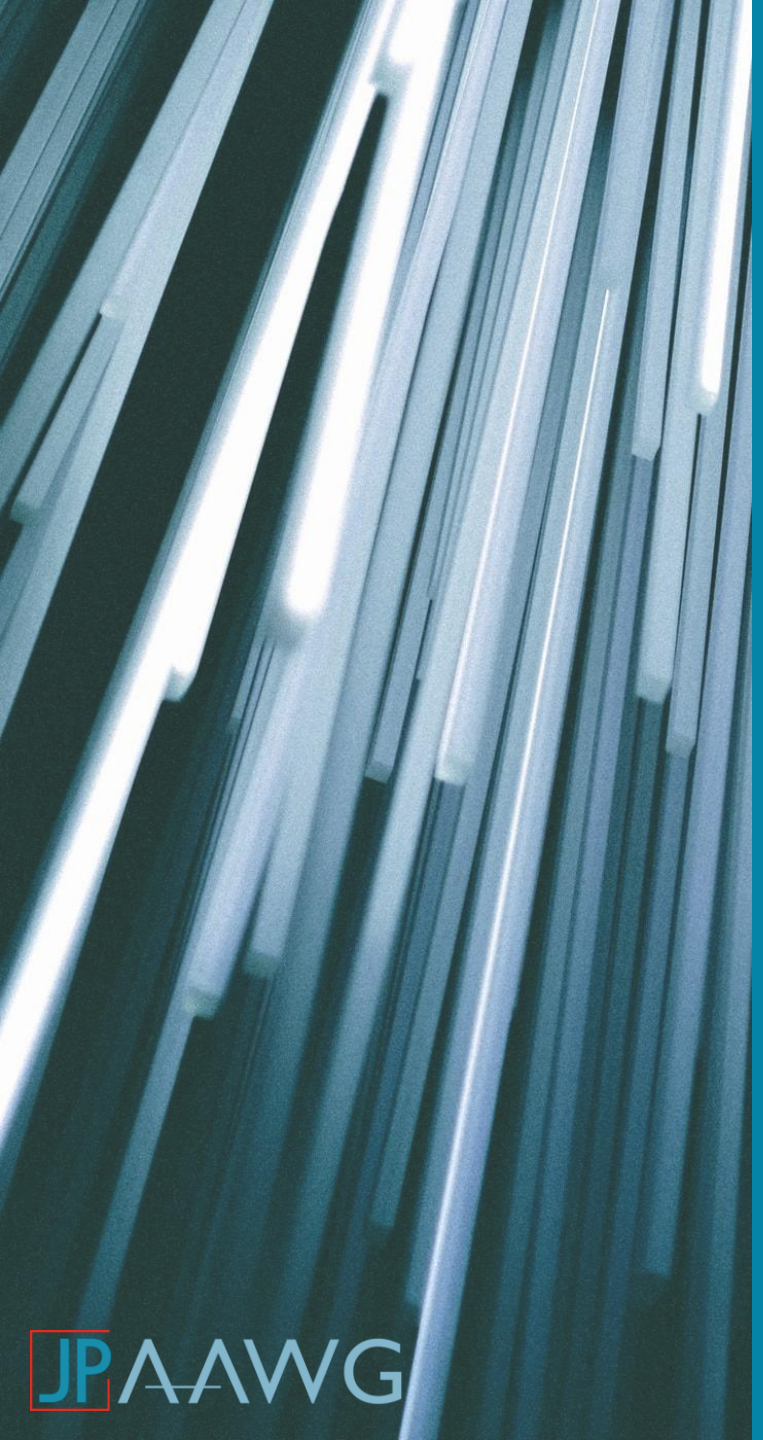
- 特定のデバイスからのみAUTHの使用を許可
- 同一のクライアントからたくさんのAUTHが成功していれば怪しいので制限する
- 過去に使われていたAUTHと違う場合制限する
- ライセンスキーなどをトークンにし、利用を許可する
- CLIENTIDのあり/なしで違うポリシーを適用する
- 複数のデバイスがIPを共有していても、CLIENTIDでレート制限すれば、他に影響を与えない
- CLIENTIDを第3の要素として認証に渡して利用する

例

C: [connection established]
S: 220 server.example.com ESMTP ready
C: EHLO client.example.net
S: 250-server.example.com
S: 250-STARTTLS
S: 250 AUTH LOGIN
C: STARTTLS
S: 220 Go ahead
C: <starts TLS negotiation>
C & S: <negotiate a TLS session>
C & S: <check result of negotiation>
C: EHLO client.example.net
S: 250-server.example.com
S: 250-AUTH LOGIN
S: 250 CLIENTID
C: CLIENTID UUID 23bf83be-aad7-46aa-9e0f-39191ccf402f
S: 250 OK
C: AUTH LOGIN dGVzdAB0ZXN0ADEyMzQ=
S: 235 Authentication successful
C: MAIL FROM:<sender@example.net>
・・・

CLIENTIDを広告

一意なIDを送信



The LIMITS SMTP Service Extension

RFC 9422 (2024-2-7)

JPAAWG

EHLOでLimitを返す拡張

- EHLOの応答: LIMITS
- 制限の種類
  - MAILMAX:
    - 1トランザクションで処理できるメール数 (MAIL FROMの回数)
  - RCPTMAX:
    - RCPT TOコマンドの回数
  - RCPTDOMAINMAX:
    - RCPT TOコマンドで渡せるユニークなドメイン数

```
C: EHLO example.org
  S: 250-mail.example.com
  S: 250-LIMITS RCPTMAX=20 MAILMAX=5
  S: 250-SIZE 100000000
  S: 250-8BITMIME
  S: 250-PIPELINING
  S: 250-CHUNKING
  S: 250 STARTTLS
```



おわり

今日の資料

JPAAWG