DKIM Working Group
Internet-Draft
Intended status: Informational
Expires: 24 November 2026

V. Moccia
ITB.it

23 May 2026

A Deployment Profile for DKIM2 via Milter Interface
draft-moccia-dkim2-deployment-profile-04

Abstract

This document defines a deployment profile for DomainKeys Identified Mail v2 (DKIM2) that is implementable via the existing milter interface without modifications to Mail Transfer Agent (MTA) core software. It identifies a mandatory core profile (DKIM2-core) covering envelope binding, chain of custody, header accountability, replay prevention and DSN authentication and an optional extended profile (DKIM2-extended) covering body recipes and Message-Instance headers. The separation is motivated by deployment realism: the core profile addresses the primary threat models identified in the DKIM2 motivation document and is deployable incrementally across heterogeneous infrastructure, including small operators, universities and research institutions, using the same milter-based deployment model that has proven effective for DKIM1 and ARC.

The intent of this document is not to obstruct DKIM2 but to make it deployable. DKIM2-core can be deployed incrementally across the heterogeneous ecosystem in a short timeframe. DKIM2-extended requires significantly longer implementation cycles and may not be deployable in jurisdictions with stricter privacy requirements. Both profiles are part of DKIM2 - the separation serves adoption, not opposition.

This Internet-Draft will expire on 24 November 2026.

Table of Contents

1. Introduction

DomainKeys Identified Mail v2 (DKIM2) addresses significant limitations of DKIM1 [RFC6376] and the experimental ARC protocol [RFC8617], including DKIM replay attacks, backscatter from unauthorized use of envelope senders and the absence of cryptographic binding between message signatures and SMTP envelope parameters.

The core technical contribution of DKIM2 - binding the MAIL FROM and RCPT TO values of each SMTP transaction to the message signature at every hop - is a genuine improvement over both DKIM1 and ARC and is sufficient to address the primary threat models identified in [I-D.ietf-dkim-dkim2-motivation].

However, the current specification also includes a body recipe mechanism that allows intermediaries to describe modifications made to the message body in sufficient detail to reconstruct previous versions. This mechanism introduces significant architectural complexity: it requires stateful milter implementations with

persistent shared storage, JSON parsing in the delivery critical path and software modifications across the entire ecosystem of intermediaries. The body recipe mechanism also raises data protection concerns under GDPR and equivalent frameworks that have not yet been addressed in the specification.

This document proposes a structured separation of DKIM2 functionality into two profiles:

* DKIM2-core: a mandatory profile implementing envelope binding, chain of custody, header accountability, replay prevention and DSN authentication. DKIM2-core is fully implementable via the milter interface without MTA core modifications, using header formats already familiar to the ecosystem through ARC deployment. Critically, DKIM2-core requires no persistent state between SMTP sessions - all information needed for signing and verification is available within the current session and the message headers themselves. State management is only required for body recipe generation, which belongs exclusively to DKIM2-extended.

* DKIM2-extended: an optional profile adding body recipe generation and verification via Message-Instance headers and JSON-encoded recipes. DKIM2-extended may require stateful milter implementations or MTA core integration and is appropriate for operators who require full body accountability and are willing to accept the associated architectural cost.

This separation is consistent with the DKIM working group charter [DKIM-CHARTER], which states that "the working group will prefer a result that is incremental to the deployed ecosystem" and that "proposed solutions are expected to be robust in terms of interoperability and scalability".

The approach taken in this document is explicitly constructive: it does not propose to replace DKIM2 but to define a deployment path that allows the ecosystem to adopt the core benefits of DKIM2 incrementally, without requiring simultaneous changes to every node in the delivery chain.

1.1. Relationship to DKIM2 Specifications

This document is a deployment profile, not a competing specification. It references and depends on [I-D.ietf-dkim-dkim2-spec] for the underlying mechanisms. Where this document proposes alternative encoding formats - specifically for envelope binding fields - these are offered as contributions to the ongoing design discussion in the working group.

1.2. Motivation

The Internet email infrastructure is not composed solely of large providers with dedicated engineering teams. A substantial portion of email is handled by operators of all sizes outside the largest commercial providers: small ISPs, universities and research institutions, regional hosting providers, non-profit organizations and small businesses. What these operators have in common is not small scale per se - a university may handle millions of messages per day - but limited engineering resources dedicated to mail infrastructure. Their administrators depend on the milter interface for incremental deployment of new authentication features precisely because it allows them to adopt new protocols without modifying MTA core software, waiting for upstream vendor releases or dedicating engineering cycles to core system changes. Every authentication protocol that has achieved broad adoption - SPF, DKIM1, DMARC and to a limited extent ARC - has been deployable via this model. DKIM2 as currently specified breaks this pattern.

Furthermore, the body recipe mechanism encounters a fundamental operational obstacle: the nodes most likely to make substantial body modifications - security gateways, DLP systems, URL rewriting proxies, antivirus engines - are precisely the nodes least likely to implement recipe generation. These nodes will systematically produce null recipes, which provide no additional accountability over incremental body hash chaining. The overhead of the recipe infrastructure is therefore paid by the entire ecosystem while the benefit accrues only to the minority of cases where all intermediaries cooperate fully.

The pragmatic approach to body integrity documented in [RFC6376] Appendix B - using relaxed canonicalization to tolerate common benign transformations rather than requiring intermediaries to declare or reconstruct them - has proven effective in practice and has contributed to the broad adoption of DKIM1. It should be noted however that relaxed canonicalization does not address all categories of involuntary body transformation: base64 line-width re-encoding, for example, breaks both simple and relaxed canonicalization equally. DKIM2-core preserves the relaxed canonicalization approach for body integrity while adding the cryptographic envelope binding and header accountability that DKIM1 lacks. The body recipe mechanism of DKIM2-extended represents a deliberate departure from this model toward deterministic reconstruction - a departure whose operational cost and deployment implications are addressed in Section 4.

1.3. Design Philosophy

This document is guided by three principles that reflect the deployment realities described in Section 1.2.

Additive, not transformative - every mechanism defined in DKIM2-core adds new header fields or new signed content to existing messages. Nothing in DKIM2-core requires existing software to change its core behavior. A legacy node that does not implement DKIM2 passes DKIM2 headers through without interpreting them, exactly as it handles any unrecognized header field today. This is the same property that allowed SPF, DKIM1, DMARC and ARC to be deployed incrementally across a heterogeneous ecosystem without flag-day transitions.

Milter-first - the milter interface is the deployment mechanism that has enabled incremental adoption of every successful email authentication protocol. DKIM2-core is designed so that every mandatory feature is implementable as a milter without MTA core modifications. This is not a constraint imposed from outside - it is a deliberate architectural choice that maximizes the probability of adoption across the full range of operators described in Section 1.2, from large providers to small ISPs and universities.

The term "milter" in the title of this document refers to the most widely deployed mechanism for extending MTA behavior without core modifications. The architectural arguments presented here apply to any filter interface that provides access to envelope callbacks during the SMTP transaction. Milter is used as the reference implementation because it is the dominant deployment model for email authentication protocols and because prototype DKIM2 implementations - including those demonstrated at the IETF hackathon - have converged on this interface independently.

Stateless by design - DKIM2-core requires no persistent state between SMTP sessions. All information needed for signing and verification is available within the current session and the message headers themselves. This eliminates the need for shared storage between milter instances, reduces operational complexity and removes a category of failure modes that stateful implementations introduce. State management is only required for body recipe generation, which belongs exclusively to DKIM2-extended.

These three principles together define the boundary between DKIM2-core and DKIM2-extended. Any mechanism that requires MTA core modifications, persistent inter-session state, or content parsing beyond what the milter interface provides belongs in DKIM2-extended. Any mechanism that can be implemented additively, via milter and statelessly belongs in DKIM2-core.

2. Terminology and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document uses terminology from [RFC5598] (Internet Mail Architecture) and inherits definitions from [I-D.ietf-dkim-dkim2-spec]. The following additional terms are defined for use in this document.

DKIM2-core: The mandatory deployment profile defined in this document. DKIM2-core implements envelope binding, chain of custody, header accountability, replay prevention and DSN authentication. It is fully implementable via the milter interface without MTA core modifications and requires no persistent state between SMTP sessions.

DKIM2-extended: The optional deployment profile defined in this document. DKIM2-extended adds body recipe generation and verification via Message-Instance headers and JSON-encoded recipes. It may require stateful milter implementations with persistent shared storage or MTA core integration.

Milter-capable node: An MTA deployment that implements DKIM2-core functionality via the milter interface. A milter-capable node requires no modifications to the MTA core software.

Legacy node: A node in the delivery chain that does not implement DKIM2 in any form. Legacy nodes pass DKIM2 headers through the delivery chain without interpreting them, exactly as they handle unrecognized header fields today.

Transparent relay: A node that adds only trace headers (Received:, Return-Path:) and makes no other modifications to the message. Transparent relays do not need to participate in DKIM2 signing or verification. Note however that in practice many nominally transparent relays introduce involuntary body transformations - base64 re-encoding at different line widths, MIME reassembly by antivirus engines, whitespace normalization - that affect body hash values.

Reviser: A node that makes intentional modifications to message headers or body. A Reviser MUST declare its modifications using DKIM2-Mod headers (DKIM2-core) or Message-Instance recipes (DKIM2-extended) and MUST add a new DKIM2 signature covering the modified message.

Inbound milter: The milter instance responsible for verifying incoming DKIM2 signatures during the SMTP session. The inbound milter operates at EndOfBody and has access to the complete envelope state accumulated during the session via MailFromRequest and RcptToRequest callbacks.

Outbound milter: The milter instance responsible for generating DKIM2-Mod headers and adding DKIM2 signatures to outgoing messages. The outbound milter is positioned last in the milter chain, after all other milters that may modify the message and signs the final state of the message.

DKIM2-Mod header: A signed header field added by a Reviser to declare a modification made to an existing header field. DKIM2-Mod headers carry an instance index (i=) aligned with the DKIM2-Signature hop number, a sequence index (seq=) for multiple modifications to the same field at the same hop and an optional frame index (fr=) to split values longer than 998 characters per [RFC5322].

Envelope binding: The cryptographic binding of SMTP MAIL FROM and RCPT TO values to the DKIM2 signature at each hop, implemented via DKIM2-Sig-mf and DKIM2-Sig-rt header fields. Envelope binding prevents DKIM replay attacks by making it impossible to reuse a valid signature for a different recipient without breaking the chain.

Chain of custody: The verifiable record of all entities that have handled a message, established by the sequence of DKIM2 signatures and envelope bindings from originator to final recipient. Each hop in the chain signs the current state of the message and references the previous hop’s signature.

Null recipe: A Message-Instance declaration that a modification was made to the message body but that the previous state cannot be reconstructed. Null recipes are only relevant in DKIM2-extended. In DKIM2-core, body modifications are declared implicitly through a changed bh= value with attribution to the signing hop.

Relaxed domain match: The relaxed domain matching algorithm for

matching the signing domain (d=) against the MAIL FROM domain, allowing subaddress schemes used for bounce handling. Labels are removed from the left side of the MAIL FROM domain until a match is found or no labels remain. Implementations MUST NOT remove more than two labels - that is, the MAIL FROM domain MUST be at most two levels below the d= domain. For example, bounce.mail.example.com matches d=example.com (two labels removed) but a.b.c.example.com does not (three labels would need to be removed). This limit is consistent with the subdomain matching defined in [RFC6376] Section 3.5.

This document inherits definitions from [I-D.ietf-dkim-dkim2-spec]. Where terms are used without definition in this document, the definitions in that document apply.

3. DKIM2-core: Mandatory Profile

DKIM2-core is the mandatory deployment profile. All nodes that wish to participate in DKIM2 MUST implement DKIM2-core. Nodes that implement only DKIM2-core are fully interoperable with nodes that implement DKIM2-extended.

The milter interface is an integration point, not a replacement for MTA functionality. DKIM2-core uses the milter interface to observe, verify and sign messages as they flow through the MTA’s normal processing pipeline. Functions that belong to the MTA - transaction management, BCC splitting, queue handling, routing - remain with the MTA. The milter does not substitute for these functions and is not designed to do so. This separation of concerns is what makes milter-based deployment non-invasive and compatible with existing MTA infrastructure.

3.1. Envelope Binding

Envelope binding is the primary technical contribution of DKIM2 over DKIM1 and ARC. By cryptographically binding the SMTP MAIL FROM and RCPT TO values to the message signature at every hop, DKIM2-core makes it impossible to replay a valid signature for a different recipient or sender without breaking the chain.

Envelope binding is implemented via two new signed header fields: DKIM2-Sig-mf (carrying the MAIL FROM value) and DKIM2-Sig-rt (carrying the RCPT TO values).

3.1.1. Envelope Binding Format

The DKIM2-Sig-mf field carries exactly one address - the SMTP MAIL FROM value. The DKIM2-Sig-rt field carries one or more RCPT TO values, using one header per address.

Both fields use the i= instance indexing convention already established by ARC [RFC8617], where ARC-Seal, ARC-Message-Signature and ARC-Authentication-Results use the same i= numbering to build a verifiable chain across multiple hops. Implementers familiar with ARC will recognize this pattern immediately. Major email providers including Google and Microsoft implement ARC natively in their infrastructure and ARC milter implementations are deployed across the ecosystem. DKIM2-Sig-mf and DKIM2-Sig-rt extend this existing pattern to carry envelope values, requiring no new parsing concepts beyond what ARC implementations already handle.

```
DKIM2-Sig-mf: i=1; addr=bounce@mailchimp.com
DKIM2-Sig-rt: i=1; v=1; addr=dest1@esempio.com
DKIM2-Sig-rt: i=1; v=2; addr=dest2@esempio.com
DKIM2-Sig-rt: i=1; v=3; addr="john;doe"@rare.com
```

One header per address, including recipients with quoted local parts. The v= tag provides a sequence number that distinguishes multiple DKIM2-Sig-rt headers at the same hop. DKIM2-Sig-mf always carries exactly one address and does not require v=.

This format handles all RFC5321 local part cases including quoted local parts containing special characters - within a single header value there is no separator ambiguity regardless of the characters present in the local part. The format copies exactly the ARC instance indexing pattern, requires no new parsing logic beyond what ARC implementations already provide and produces output directly inspectable in mail logs and headers without decoding tools.

The design avoids the parsing complexity that arises when envelope addresses with display names, quoted local parts, or multiple addresses are embedded in more complex header formats. DKIM2-Sig-rt carries only the bare RFC5321 addr-spec - the envelope address without display name - one per header. This is the natural format for SMTP envelope values, which by definition do not carry display names. It avoids the parsing complexity of RFC5322 address-list syntax, which includes display names, group syntax and other constructs that are irrelevant for envelope binding purposes.

This format could produce more bytes than the JSON+base64 encoding currently specified in [I-D.ietf-dkim-dkim2-spec] for messages with very many recipients. For the common case of messages with a small

number of recipients - which represents the substantial majority of real-world email traffic - the size difference is negligible or absent. It remains directly human-readable without decoding tools, has a parsing complexity of O(1) per header rather than O(n) on the full recipient list and allows individual recipient membership verification without deserializing the complete list.

DKIM2-core lists each RCPT TO value as an individual DKIM2-Sig-rt header, all covered by a single DKIM2-Signature. This provides per-address cryptographic binding: a verifier can confirm that the message was signed for exactly these recipients and no others. Replaying the message to a recipient not listed in a DKIM2-Sig-rt header produces a verifiable mismatch, even if that recipient is at the same domain as the original recipients. Alternative approaches such as DARA bind at the domain level - all recipients at the same destination domain share a single signature via darn=. DARA prevents replay to a different domain but does not prevent replay to a different recipient at the same domain. DKIM2-core closes this gap without requiring any additional MTA splitting beyond the standard per-domain delivery that MTAs already perform. The base specification [I-D.ietf-dkim-dkim2-spec] encodes rt= values as base64 without JSON in its current revision, which reduces parsing overhead but retains opacity - the values are not directly human-readable without decoding. DKIM2-core's plaintext tag-value format for envelope binding fields remains directly inspectable without tooling. DKIM2-core achieves equivalent cryptographic binding of each recipient to the signed message via the standard milter envrcpt callback, listing all transaction recipients in DKIM2-Sig-rt fields. No MTA modification is required, making the mechanism deployable by any operator regardless of infrastructure.

It is worth noting that quoted local parts containing special characters such as semicolons are permitted by [RFC5321] but are not observed in practice in real-world email infrastructure. The format handles them correctly without any special-casing.

3.1.2. Relationship to Existing From:/To: Headers

The DKIM2-Sig-mf and DKIM2-Sig-rt fields carry the SMTP envelope values, which may differ significantly from the RFC5322 From:, To: and Cc: header fields. This distinction is particularly important in two scenarios:

Programmatic and bulk email - the envelope sender is typically a bounce handling address such as bounce-12345@mail.mailchimp.com and the envelope recipients are individual subscriber addresses that may not appear in the message headers at all. Using RFC5322 headers to infer envelope values would fail for this common case.

Group syntax and empty groups - RFC5322 permits constructs such as To: Empty Group:; where the To: header carries no actual recipient addresses. The envelope RCPT TO values are the only reliable source of recipient information in these cases.

Using existing RFC5322 headers to infer envelope values, as has been proposed in some alternatives, would require parsers to handle these edge cases correctly and would fail for programmatic email where no correspondence between envelope and headers exists. DKIM2-Sig-mf and DKIM2-Sig-rt carry the envelope values explicitly and independently of the IMF headers, eliminating this ambiguity.

It is also worth noting that the support for multiple rt= values - rather than a single recipient per message - was motivated by a specific operational requirement from a large mailbox provider whose anti-fraud system relies on verifying consistency between envelope recipients and To:/Cc: header fields to detect messages fraudulently presented as sent to multiple recipients but actually sent to only one. This use case requires that all RCPT TO values for a given SMTP transaction be visible in the signed envelope binding.

3.1.3. Relaxed Domain Match

The relaxed domain match algorithm is retained unchanged. The signing domain (d=) must match the domain of the DKIM2-Sig-mf address via relaxed domain match, allowing subaddress schemes used for bounce handling such as bounce-12345@mail.mailchimp.com to match d=mailchimp.com. The MAIL FROM domain MUST be at most two levels below the d= domain, consistent with [RFC6376] Section 3.5.

3.1.4. BCC Recipients

BCC recipients require separate SMTP transactions to prevent disclosure of BCC addresses to other recipients, consistent with [RFC5322] Section 3.6.3. This is already best practice for correct BCC semantics independent of DKIM2 - most MTAs already split BCC recipients into separate transactions for privacy reasons. DKIM2-core makes this a cryptographic requirement in addition to a privacy requirement.

The milter processes two distinct transaction types:

For the To:/Cc: transaction, the milter receives all RCPT TO values in a single transaction, compares them against the To: and Cc: header fields and includes only the visible recipients in DKIM2-Sig-rt. The comparison requires address extraction from RFC5322 headers - display names and comments must be stripped to obtain bare addr-spec values - followed by case-insensitive comparison on the domain part. Note

that a recipient appearing in both To: and BCC is possible but rare; in that case the milter correctly includes the address in rt= since it is a visible recipient. The signed DKIM2-Sig-rt reflects exactly the recipients who will see the message.

For each BCC transaction, the MTA generates a separate SMTP transaction per BCC recipient. The milter signs each transaction independently with a single DKIM2-Sig-rt carrying that recipient's address. No special handling is required - the milter processes each BCC transaction as a normal single-recipient message.

Including BCC recipients in the To:/Cc: transaction with all addresses listed in DKIM2-Sig-rt would reveal BCC addresses in the signed headers, visible to all recipients and any archiving system. This directly violates [RFC5322] Section 3.6.3 and MUST NOT be done.

This design requires no MTA core modifications. BCC splitting is already standard MTA behavior for RFC 5322 compliance - DKIM2-core formalizes an existing practice rather than introducing a new operational burden. A future standardized mechanism for BCC signaling at the SMTP level would simplify this comparison but is not required for correct operation.

The milter buffers RCPT TO values during the envrcpt callbacks and performs the comparison against To: and Cc: headers in the eom callback, when all header fields are available. This is the standard milter processing pattern. In the common case the milter can identify BCC recipients through this comparison and generate the appropriate transactions. The truly pessimistic default - treating every RCPT TO as a separate transaction regardless - is only necessary when To:/Cc: headers are absent or unparseable, which is an edge case in practice. When BCC signaling is available through MTA configuration or a local convention, the milter can group visible To:/Cc: recipients into a single transaction and split only the BCC recipients, recovering full efficiency. The splitting cost is operational, not architectural - it requires no changes to the MTA core.

3.1.5. Formal Syntax

The following ABNF defines the header fields introduced by this document. Rules are imported from [RFC5234] and [RFC5321] as noted.

```
   ; DKIM2-Sig-mf header field
   ; Carries exactly one SMTP MAIL FROM address
   dkim2-sig-mf     = "DKIM2-Sig-mf:" SP dkim2-mf-params CRLF
   dkim2-mf-params  = dkim2-hop-index ";" SP dkim2-addr

   ; DKIM2-Sig-rt header field
   ; Carries exactly one SMTP RCPT TO address per header
   ; Multiple recipients use multiple headers with incrementing v=
   dkim2-sig-rt     = "DKIM2-Sig-rt:" SP dkim2-rt-params CRLF
   dkim2-rt-params  = dkim2-hop-index ";" SP
                      dkim2-rcpt-index ";" SP
                      dkim2-addr

   ; DKIM2-Mod header field
   ; Declares a modification made to an existing header field
   dkim2-mod        = "DKIM2-Mod:" SP dkim2-mod-params CRLF
   dkim2-mod-params = dkim2-hop-index ";" SP
                      dkim2-mod-seq ";" SP
                      dkim2-field-tag
                      [ ";" SP dkim2-fr-tag ]
                      ";" SP dkim2-mod-value

   dkim2-field-tag  = "field=" header-field-name
   dkim2-fr-tag     = "fr=" 1*2DIGIT
                    ; DKIM2-core implementations MUST NOT use fr= values greater than 2
                    ; fragment index, 1-20
                    ; OPTIONAL - absent when value fits in single header

   dkim2-mod-value  = 1*(VCHAR / WSP)
   dkim2-del-tag    = "del=" dkim2-mod-value
                    ; MUST appear last in header
   dkim2-new-tag    = "new=" dkim2-mod-value
                    ; MUST appear last in header

   ; del= and new= MUST be on separate header lines
   ; Presence rules per complete operation:
   ;   del= only  -> removal
   ;   new= only  -> addition
   ;   del= + new= -> modification


   ; Shared index rules
   dkim2-hop-index  = "i=" 1*2DIGIT
                    ; hop sequence number, 1-20, starts at 1
   dkim2-rcpt-index = "v=" 1*3DIGIT
                    ; recipient sequence number for DKIM2-Sig-rt
                    ; starts at 1, increments per recipient at same hop, 1-500
   dkim2-mod-seq    = "seq=" 1*2DIGIT
```

```
                    ; modification sequence number for DKIM2-Mod, 1-20

   ; Address rule
   dkim2-addr       = "addr=" addr-spec
                    ; addr-spec as defined in [RFC5321] Section 4.1.2
                    ; including quoted local parts

   ; Header field name
   header-field-name = 1*( ALPHA / DIGIT / "-" )
                    ; as defined in [RFC5322] Section 2.2

   ; Supporting rules
   tag-list         = tag-spec *( ";" tag-spec ) [ ";" ]
   tag-spec         = tag-name "=" tag-value
   tag-name         = ALPHA *( ALPHA / DIGIT / "_" )
   tag-value        = *( %x21-3A / %x3C-7E )
                    ; printable ASCII except semicolon

   domain-name      = sub-domain *("." sub-domain)
   sub-domain       = 1*( ALPHA / DIGIT / "-" )
   selector-name    = 1*( ALPHA / DIGIT / "-" / "_" )
   base64string     = 1*( ALPHA / DIGIT / "+" / "/" / "=" )
                    ; standard base64 alphabet per [RFC4648]

   ; Core rules imported from [RFC5234]
   ALPHA            = %x41-5A / %x61-7A
   DIGIT            = %x30-39
   SP               = %x20
   CRLF             = %x0D %x0A
   VCHAR            = %x21-7E
   WSP              = SP / %x09
```

The base64string production uses the standard base64 alphabet defined in [RFC4648].

3.1.6. DKIM2-Signature Envelope Tags

```
   ; DKIM2-Signature envelope tags
   ; These tags appear within the DKIM2-Signature header field
   ; as defined in [I-D.ietf-dkim-dkim2-spec]

   dkim2-sig-tag-i  = %x69 "=" 1*2DIGIT
                    ; i= hop sequence number
                    ; MUST start at 1 and increment by 1 at each signing hop
                    ; gaps in sequence MUST be treated as making the message unsigned

   dkim2-sig-tag-v  = %x76 "=" 1*3DIGIT
                    ; v= value sequence number
                    ; used in DKIM2-Sig-rt to distinguish
                    ; multiple recipients at the same hop
                    ; MUST start at 1 and increment by 1

   dkim2-sig-tag-d  = %x64 "=" domain-name
                    ; d= signing domain

   dkim2-sig-tag-s  = %x73 "=" selector-name
                    ; s= selector

   dkim2-sig-tag-bh = %x62 %x68 "=" base64string
                    ; bh= body hash

   dkim2-sig-tag-b  = %x62 "=" base64string
                    ; b= signature value

   dkim2-sig-tag-hh = %x68 %x68 "=" base64string
                    ; hh= header hash
                    ; hash of all non-excluded message header fields
                    ; computed before signing, using alphabetical ordering by lowercase f
ield name
                    ; OPTIONAL in DKIM2-core signatures

   ; Tag-value list structure - as defined above
```

3.2. Chain of Custody

Each hop in the delivery chain MUST add a DKIM2-Signature header field signing the current state of the message. The chain of custody is established by the sequence of signatures and envelope bindings from originator to final recipient.

DKIM2-core verification is hop-by-hop, not end-to-end. Each hop verifies the previous signature against the body it actually receives at the time of receipt. Attribution of body modifications does not require post-facto reconstruction: if bh= changes between hop N and hop N+1, the signing domain at hop N+1 is cryptographically identified as the modifier at the moment N+1 signs. No subsequent

reconstruction is needed to establish who modified the body - the chain of signatures provides this attribution directly and verifiably.

3.2.1. Signature Content

The DKIM2-Signature at each hop covers:

* All DKIM2-Sig-mf and DKIM2-Sig-rt headers with the current i= value
* All DKIM2-Mod headers with i= values less than or equal to the current hop
* All previous DKIM2-Signature headers
* The incomplete current DKIM2-Signature header with the signature value absent

3.2.2. Body Hash

Each hop MUST calculate and include a body hash (bh=) of the current message body using the canonicalization algorithm specified in the signature. The bh= value changes whenever the body is modified, providing implicit notification that a modification occurred and attribution of that modification to the signing hop.

No body recipe or reconstruction is required or expected in DKIM2-core. The change in bh= value between hops, combined with the identity of the signing domain at the modifying hop, provides sufficient accountability for delivery decisions.

3.2.3. Sequence Numbering

DKIM2-Signature headers are numbered sequentially starting at i=1. Gaps in the sequence MUST be treated as making the message unsigned. Implementations MUST enforce the following per-type limits as a denial-of-service mitigation:

* i= (hop index): MUST NOT exceed 20. A realistic worst-case delivery path including nested mailing lists and security gateways at both ends does not exceed 15-16 hops; 20 provides margin without permitting pathological chains.
* v= (recipient sequence): MUST NOT exceed 500 per hop. This is consistent with the recipient limits enforced by major MTA implementations.

* seq= (modification sequence): MUST NOT exceed 20 per hop per field. A single hop modifying more than 20 instances of the same header field is not a legitimate scenario.

* fr= (fragment index): In DKIM2-core, implementations MUST NOT exceed 2 fragments per declaration. This limit reflects the practical constraint that a standard [RFC5322] header field value cannot exceed 998 characters, requiring at most one continuation frame when combined with the DKIM2-Mod tag overhead of approximately 30 characters. A limit of 20 fragments would only be relevant if recipe content were transported in DKIM2-Mod headers - a use case that depends on architectural decisions about recipe storage not yet resolved in the base specification [I-D.ietf-dkim-dkim2-spec]. DKIM2-extended implementations MAY increase this limit when recipe transport in headers is formally specified.

In addition, implementations MUST reject messages whose total size of DKIM2-specific headers (DKIM2-Signature, DKIM2-Sig-mf, DKIM2-Sig-rt and DKIM2-Mod) exceeds 128KB. This global byte cap is a secondary defense-in-depth measure; the per-type limits above are the primary mitigation.

3.2.4. Signed Header Set

The DKIM2 signature at each hop covers a specific set of header fields. The signed header set MUST include:

* All DKIM2-Sig-mf headers with i= value equal to the current hop

* All DKIM2-Sig-rt headers with i= value equal to the current hop

* All DKIM2-Mod headers with i= value less than or equal to the current hop. DKIM2-Mod headers are subject to the same canonicalization rules as other signed headers per [RFC6376] Section 3.4. The del= and new= values are canonicalized as header field values - line endings are normalized and folding whitespace is handled per the canonicalization algorithm specified in the signature.

* All previous DKIM2-Signature headers with i= value less than the current hop

* The incomplete current DKIM2-Signature header with the b= value set to empty or a placeholder as defined in [I-D.ietf-dkim-dkim2-spec]

Received:, Return-Path: and other transport trace headers are excluded from the signed header set. They are also excluded from the hh= computation as specified in Section 3.3.5.

Header fields not listed above MAY be included in the signed header set at the discretion of the signer. Signers SHOULD include headers that have security relevance for the message - From:, To:, Subject:, Date: - and SHOULD declare their inclusion or exclusion consistently across all messages.

The b= value is calculated over the signed header set using the canonicalization algorithm specified in the signature, following the same procedure defined in [RFC6376] Section 3.7 with the modifications for DKIM2 header ordering defined in [I-D.ietf-dkim-dkim2-spec].

Implementations MUST NOT assume that header fields arrive in a specific order. The signed header set is identified by field name and instance index (i=, seq=), not by physical position in the message. Intermediate MTA software may reorder header fields during transit without affecting chain of custody verification, since canonical ordering is established through the i= and seq= index values embedded in DKIM2-Sig-mf, DKIM2-Sig-rt and DKIM2-Mod headers. Implementations MUST use these index values, not header field position, to reconstruct the chain of custody.

3.2.5. Header Hash (hh=)

The hh= tag contains a hash of all message header fields not excluded from the signed header set. This tag is OPTIONAL in DKIM2-core signatures. When present, it enables additional hop-by-hop integrity verification of declared modifications: while b= and DKIM2-Mod already provide cryptographic proof of each hop’s own declarations, hh= with the rollback mechanism described in Section 3.2.5 allows each hop to verify that its predecessor has not omitted undeclared header modifications.

The following header fields are excluded from the hh= computation:

* Headers managed by DKIM2 itself: DKIM2-Signature, DKIM2-Sig-mf, DKIM2-Sig-rt, DKIM2-Mod

* Headers with independent authentication mechanisms: DKIM-Signature, ARC-*

* Headers that change legitimately at every hop: Received, Return-Path, Authentication-Results

* Vendor-specific diagnostic headers: X-MS-_, X-GM-_

All remaining header fields, including other X-* headers, are included in the hh= computation. The rationale for including non-vendor X-* headers and the security implications of excluding them are discussed in Section 7.

The hash algorithm used for hh= MUST match the algorithm used for bh= in the same DKIM2-Signature, as specified in the a= tag.

Canonicalization follows the DKIM1 relaxed algorithm per [RFC6376] Section 3.4: header field names are lowercased, whitespace is normalized and continuation lines are unfolded.

3.3. Header Accountability

When a Reviser modifies, removes, or adds a header field, it MUST declare the change by adding one or more DKIM2-Mod headers before signing. These headers MUST be included in the signed header set of the modifying hop and all subsequent hops, making the declaration cryptographically bound to the chain.

3.3.1. Modification Cases

Modification - header existed and its value was changed. del= and new= on separate headers with same i= and seq=:

```
DKIM2-Mod: i=2; seq=1; field=Subject; del="Original subject text"
DKIM2-Mod: i=2; seq=1; field=Subject; new="Modified subject text"
```

Removal - header existed and was removed. Only del=:

```
DKIM2-Mod: i=2; seq=1; field=Subject; del="Removed subject text"
```

Addition - header did not exist and was added. Only new=:

```
DKIM2-Mod: i=2; seq=1; field=List-Id; new=<lista@esempio.com>
```

Note that new= is technically redundant for additions since the added value is already visible in the message headers. It is declared explicitly to provide an unambiguous cryptographic binding between the modification declaration and the hop signature, particularly in cases where multiple headers of the same type exist in the message.

When a DKIM2-Mod header with del= and a DKIM2-Mod header with new= share the same (i=, seq=, field=) tuple, they MUST be interpreted as a single modification operation representing a value change. Implementations MUST NOT interpret them as independent operations -

that is, as a removal followed by an unrelated addition. The tuple (i=, seq=, field=) is the sole correlating key. When present as a pair, the del= header MUST precede the new= header in the message.

The dkim2-mod-value production requires at least one character - empty string values are not permitted. A header field whose value is the empty string is treated as absent for the purposes of DKIM2-Mod declarations.

3.3.2. Multiple Modifications

Multiple instances of the same field, each with its own seq=:

```
DKIM2-Mod: i=2; seq=1; field=X-EVALUATION; del="pippo"
DKIM2-Mod: i=2; seq=1; field=X-EVALUATION; new="paperino"
DKIM2-Mod: i=2; seq=2; field=X-EVALUATION; del="pluto"
DKIM2-Mod: i=2; seq=2; field=X-EVALUATION; new="paperone"
```

Successive hops use their own i=:

```
DKIM2-Mod: i=2; seq=1; field=Subject; del="Value before hop 2"
DKIM2-Mod: i=2; seq=1; field=Subject; new="Value after hop 2"
DKIM2-Mod: i=3; seq=1; field=Subject; del="Value after hop 2"
DKIM2-Mod: i=3; seq=1; field=Subject; new="Value after hop 3"
```

The del= and new= tags MUST appear last in the DKIM2-Mod header field value. All other tags (i=, seq=, field=, fr=) MUST precede them.

3.3.3. Long Values

For values that exceed practical inline length, implementations MAY use the optional fr= tag to split across multiple headers with incrementing fr= values. fr= is independent for del= and new=:

```
DKIM2-Mod: i=2; seq=1; field=X-Microsoft-Antispam; fr=1; del="first part..."
DKIM2-Mod: i=2; seq=1; field=X-Microsoft-Antispam; fr=2; del="...second part..."
DKIM2-Mod: i=2; seq=1; field=X-Microsoft-Antispam; fr=3; del="...third part"
DKIM2-Mod: i=2; seq=1; field=X-Microsoft-Antispam; new="new value"
```

fr= absent means value is complete in that single header. When fr= is present, headers with same i=, seq=, field= and same tag type (del= or new=) are concatenated in fr= order.

Implementations that do not support fr= MUST treat a fragmented declaration as a null declaration for that field.

Fragments MUST be concatenated without any intervening whitespace or separators. The reconstructed value is the exact concatenation of the fragment values in fr= order.

The fr= tag value MUST be a positive integer starting at 1. A value of 0 or any negative value is malformed and MUST cause verification of that DKIM2-Mod declaration to fail. Fragments MAY appear in any physical order within the header set - implementations MUST reassemble them by fr= value rather than by physical position. If the fr= sequence contains a gap - that is, if fragment N is present but fragment N-1 is absent - verification of that DKIM2-Mod declaration MUST fail. Partial reconstruction from available fragments MUST NOT be attempted. A gap in the fr= sequence or a malformed fr= value SHOULD be logged as a potential integrity violation.

3.3.4. Relationship to Existing Conventions

DKIM2-Mod headers formalize and cryptographically bind the informal X-Original- convention already widely deployed in the ecosystem. Systems using X-Original-From, X-Original-Subject and similar headers preserve previous header values across intermediary modifications - the same semantic goal as the del= tag in DKIM2-Mod. The key difference is that DKIM2-Mod declarations are included in the signed header set and cryptographically bound to the chain of custody, making them verifiable rather than merely informational.

This pattern is already practiced informally in the ecosystem - for example, Google Groups preserves the original sender address in an X-Original-Sender header alongside its own Sender: field. DKIM2-Mod formalizes this existing convention by making the declaration cryptographically signed and part of the verifiable chain of custody, rather than an informational annotation with no authentication binding.

This approach has precedent in [I-D.chuang-mailing-list-modifications], which proposed X-Prior- headers and Content-Footer headers as ARC extensions for the same purpose, declaring mailing list modifications in human-readable form without JSON encoding. DKIM2-Mod formalizes and generalizes this approach within the DKIM2 chain of custody framework.

3.3.5. Header Order and Multiple Instances

DKIM2-core signatures cover a defined set of header fields. The mandatory signed header set is specified in Section 3.2.4. Signers MAY additionally include security-relevant header fields such as From:, Subject: and Reply-To: by declaring them in the signature.

DKIM2-core signatures protect header fields at two levels. The h= tag in DKIM2-Signature covers the mandatory set of security-critical header fields, providing value-level protection consistent with DKIM1. The optional hh= tag provides set-level integrity for all non-excluded header fields, detecting undeclared additions and removals as described in Section 3.2.5.

When multiple header fields with the same name exist, their relative physical order in the message may be altered by intermediate systems - milters, antivirus scanners and content filters routinely reconstruct messages in ways that can reorder header fields with identical names. If the signed header set includes such fields and their order changes, any existing signature covering the original order will no longer validate.

DKIM2-core addresses this through a two-level ordering strategy that minimises the surface area of ambiguous sorting.

The first level covers all header fields included in the hh= computation. These are sorted alphabetically by lowercase field name. For most header fields, [RFC5322] prohibits multiple instances at origination, so this sort operates on unique keys and is fully deterministic. This ordering is required at every hop that computes hh=, making it a wire-level constraint - but one that operates on field names only, not on field values, and that produces a fixed-size hash rather than an ordered sequence that downstream verifiers must reconstruct.

The second level handles duplicate instances of the same header field. Duplicates introduced by intermediate hops are declared via DKIM2-Mod with explicit i= and seq= index values. A verifier encountering multiple instances of the same field uses these indices to determine the temporal order of additions - the instance without a corresponding DKIM2-Mod entry is the original and instances declared via DKIM2-Mod are ordered by their i= value. This is an integer lookup, not a string sort and is unambiguous by construction.

A residual case exists: header fields that [RFC5322] permits in multiple instances at origination (such as Comments: and Keywords:) and that have no corresponding DKIM2-Mod entries. These instances are not attributed to any DKIM2-Mod declaration and must be disambiguated by a secondary sort on canonicalised field value. This sort is limited in scope - it applies only to the narrow set of [RFC5322] multi-instance fields, which are rare in production traffic. If a field that [RFC5322] defines as singular appears in multiple instances without a corresponding DKIM2-Mod declaration, this is a protocol violation and the verifier MAY treat it as a signature error.

Both DKIM2-core and the DKIM2 base specification [I-D.ietf-dkim-dkim2-spec] require ordering computations. The difference is in the scope and determinism of the sort. The base specification mandates alphabetical ordering of all header field names with bottom-up ordering for duplicate names - a sort that operates on the full header set including body recipes, whose size grows with message complexity and delivery path length. The sort result depends on the stability of the sort implementation, on tie-breaking rules for headers with the same name and on how headers containing non-ASCII characters are ordered - all points where independent implementations can and do diverge. Interoperability testing at the IETF 124 hackathon confirmed failures between independent implementations attributable to disagreements on tag ordering within DKIM2-Signature fields - a simpler problem than duplicate header ordering, yet sufficient to cause failures.

DKIM2-core reduces the surface of ordering ambiguity to the minimum achievable without eliminating ordering entirely. Field-name sorting is deterministic on unique ASCII-lowercase keys. Duplicate ordering uses explicit integer indices for all hop-introduced instances. Value-based sorting is confined to the residual case of originals not attributed to any DKIM2-Mod declaration for [RFC5322] multi-instance fields - a corner case that is rare in practice and bounded in scope. The base specification has no equivalent mechanism for disambiguating duplicates without relying on physical header position, which intermediate systems may alter.

3.3.6. Hop-by-Hop Header Integrity Verification

When a verifier at hop i=N receives a message carrying an hh= tag, it performs verification in two sequential steps. This mechanism requires that all hops in the chain include hh= in their DKIM2-Signature.

The verifier first establishes the canonical header set: all header fields present in the received message are ordered alphabetically by lowercase field name. Duplicate fields are ordered using the DKIM2-Mod declarations of hop i=N-1: fields without a corresponding DKIM2-Mod entry are originals; fields declared via DKIM2-Mod are ordered by their i= and seq= values. A duplicate field instance that cannot be explained by any DKIM2-Mod declaration in the chain is treated as an original field present at origination. If a header field defined as single-instance by [RFC5322] appears more than once, each additional instance beyond the first MUST be accounted for by a corresponding DKIM2-Mod with a new= declaration. Unaccounted duplicates of single-instance fields constitute a protocol violation and the verifier MUST treat it as a verification failure.

The first step verifies the integrity of what hop i=N-1 signed. The verifier recomputes hh= over the canonical header set and compares the result against the hh= value declared in the DKIM2-Signature of hop i=N-1. If the values differ, the header set has been altered since hop i=N-1 signed it and the verifier MUST treat this as a verification failure.

The second step is OPTIONAL and verifies that hop i=N-1 has not lied about its own declared modifications. The verifier applies the DKIM2-Mod declarations of hop i=N-1 in reverse: additions declared via new= are removed from the header set and removals declared via del= are restored to the header set with their original values. Modifications declared via both del= and new= are reversed by removing the new= value and restoring the del= value. The result is the header state as it should have been at hop i=N-2 according to what hop i=N-1 has declared. The verifier then recomputes hh= over this reconstructed header set and compares it against the hh= value declared in the DKIM2-Signature of hop i=N-2. If the values differ, hop i=N-1 has declared modifications that do not account for all changes it made, and the verifier MUST treat this as a verification failure. In the transitional deployment phase described in Section 3.6.4, the verifier MAY record a dkim2=fail result in the DKIM2-Authentication-Results header rather than rejecting the message.

This check is a single step backward, not a full chain traversal. Each honest hop performs the same check on its predecessor, so a lie introduced at any point in the chain is detected at the immediately following hop rather than at the final receiver. This property means that the chain self-validates during transit without requiring the receiver to reconstruct the entire message history.

At i=1 there are no DKIM2-Mod declarations. The hh= value at i=1 covers the original header set and serves as the baseline against which hop i=2 will perform its rollback check.

The mechanism does not require trust in intermediaries. Each hop verifies the previous signature cryptographically before signing its own. A hop that modifies a header without declaring it in DKIM2-Mod will produce a reconstructed header set that does not match the hh= of its predecessor and the discrepancy will be detected by the next honest verifier in the chain.

The h= tag in DKIM2-Signature already provides cryptographic protection for the mandatory security-critical header fields. The hh= rollback mechanism extends this protection to the full non-excluded header set. The two mechanisms are complementary: h= provides hard guarantees on critical fields regardless of whether hh= is present and hh= provides additional assurance that no undeclared modification has occurred across the complete header set.

3.4. Replay Prevention

Replay prevention is a direct consequence of envelope binding. Since DKIM2-Sig-mf and DKIM2-Sig-rt are cryptographically signed at every hop and contain the actual SMTP envelope values, it is impossible to reuse a valid signature for a different recipient without breaking the chain. An attacker who attempts to replay a message to a recipient not listed in DKIM2-Sig-rt will produce a mismatch between the signed rt= value and the actual RCPT TO of the new transaction, which MUST be rejected by a conformant verifier.

Mailing list redistribution is not a replay attack. When a mailing list receives a message and redistributes it to subscribers, it adds its own DKIM2 signature with its own mf= and rt= values, explicitly declaring the redistribution. The chain of custody shows the original sender, the mailing list and the final recipient as distinct entities in the chain.

3.5. DSN and Bounce Authentication

DKIM2-core provides the infrastructure for authenticated Delivery Status Notifications (DSNs) that prevent backscatter to innocent sender domains.

3.5.1. Verification During SMTP Session

DKIM2-core verification is performed during the SMTP session at two distinct points:

During RCPT TO - the inbound milter MAY perform local policy checks based on the envelope sender and recipient, for example checking against local allowlists or denylists. No DKIM2-specific verification is possible at this stage because the message headers, including DKIM2 signatures, have not yet been received.

At EndOfBody - full signature verification is performed when the complete message and accumulated envelope state are available. The inbound milter verifies the DKIM2 signature chain, the envelope binding in DKIM2-Sig-mf and DKIM2-Sig-rt and the consistency of DKIM2-Mod declarations with the signed header set. The milter then returns one of the following to instruct the MTA:

* SMFIS_CONTINUE - the message is correctly signed and the chain is valid; the MTA proceeds with delivery

* SMFIS_REJECT - the signature is invalid, the chain is broken, or the envelope binding does not match; the MTA issues a 5xx rejection to the connected peer

* SMFIS_TEMPFAIL - a transient error occurred, for example a DNS timeout during key lookup; the MTA issues a 4xx temporary failure to the connected peer

When the milter returns SMFIS_REJECT, the MTA issues a 5xx rejection directed at the connected peer - the system currently delivering the message over the active SMTP connection. This rejection is never directed at the original envelope sender. This is the fundamental mechanism that prevents backscatter: no DSN is ever generated toward an address that was not the source of the current SMTP session.

3.5.2. Reject Propagation and Backscatter Prevention

When a DKIM2-core node rejects a message with 5xx during the SMTP session, the connected peer - the previous hop in the delivery chain - receives the rejection and is responsible for propagating it back toward the original sender. Each hop manages its own rejection toward its direct peer. The rejection propagates back along the authenticated chain hop by hop without generating backscatter at any point.

A node that accepts a message with an invalid or missing DKIM2 signature and subsequently generates a DSN to the original envelope sender violates a MUST NOT in the protocol and produces backscatter. During the transition period, nodes MAY accept messages with invalid or missing DKIM2 signatures as a matter of local policy while adoption is incomplete. Nodes that do so MUST NOT generate DSNs to the original envelope sender - they MUST either discard silently or generate DSNs only to the connected peer.

3.5.3. Authenticated DSN Generation

A node that has accepted a DKIM2-signed message and needs to generate a DSN does not need to possess a signing key aligned with the rt= value of the incoming signature. It needs only to sign the DSN with a key authorized for its own domain and direct it to the connected peer that delivered the original message. The DSN propagates back along the chain via the same hop-by-hop rejection mechanism described in Section 3.5.2.

3.5.4. Transition Period Behavior

During the transition period when DKIM2-capable and legacy nodes coexist, a receiver that gets a message without a valid DKIM2 signature cannot perform DKIM2-specific verification - envelope binding, chain of custody validation and authenticated DSN generation toward the connected peer are not available. The receiver falls back to pre-DKIM2 behavior: verify DKIM1 signatures if present, apply DMARC policy if configured and generate DSNs as today - including the backscatter risk that DKIM2 was designed to eliminate. Backscatter prevention is only fully effective when all intermediate nodes participate in DKIM2. A single legacy node in the chain breaks authenticated DSN propagation for downstream receivers. This is a known limitation of the transition period addressed in Section 5.

3.6. Cryptographic Algorithms

DKIM2-core mandates support for RSA-SHA256 and Ed25519-SHA256 signing algorithms as defined in [I-D.ietf-dkim-dkim2-spec]. Verifiers MUST implement both algorithms. Signers SHOULD implement both algorithms and MAY sign a message with multiple algorithms simultaneously - for example RSA-SHA256 for compatibility with legacy verifiers and Ed25519-SHA256 for efficiency. Additional signing algorithms, including post-quantum algorithms, MAY be defined in future documents and added to the set of supported algorithms without requiring changes to this profile. The DKIM2-core architecture does not publish the hash value separately from the signature, which permits use of signing algorithms that incorporate their own hash function.

When a message carries signatures for multiple algorithms at the same hop, a verifier that supports all those algorithms MUST treat the failure of any one signature as invalidating the entire hop. A partial pass - where one algorithm verifies and another fails - MUST be treated as a verification failure. This prevents downgrade attacks where an attacker invalidates the stronger algorithm signature to force acceptance based solely on the weaker one.

For body hash calculation, DKIM2-core supports both relaxed and strict canonicalization as defined in [RFC6376] Section 3.4. The choice of canonicalization algorithm is indicated in the signature and is a per-hop decision.

The choice between relaxed and strict canonicalization for body hashing reflects a fundamental tradeoff documented in [RFC6376] Appendix B. Relaxed canonicalization tolerates common benign transformations made by intermediate systems - whitespace normalization, line ending conversion, quoted-printable to 8bit transcoding - at the cost of reduced sensitivity to intentional modifications. Strict canonicalization detects all byte-level changes but is more sensitive to involuntary transformations. DKIM2-core implementations that include legacy infrastructure in their deployment path SHOULD use relaxed canonicalization for body hashing to maximize chain continuity.

3.7. Milter Implementation

DKIM2-core is designed to be implementable via the milter interface without modifications to MTA core software. This section describes the recommended deployment patterns.

3.7.1. Two-Milter Pattern

The recommended implementation uses two milter instances. The protocol flow described in Section 3.5 maps to the milter callbacks as follows:

Inbound milter - runs on the receiving MTA. Operates at two points in the SMTP session:

* During RCPT TO: MAY perform local policy checks based on the envelope sender and recipient - for example checking against local allowlists or denylists. No DKIM2-specific verification is possible at this stage because the message headers, including DKIM2 signatures, have not yet been received.

* At EndOfBody: performs full DKIM2 verification with access to the complete message and accumulated envelope state from MailFromRequest and RcptToRequest callbacks. The milter verifies the DKIM2 signature chain, the envelope binding in DKIM2-Sig-mf and DKIM2-Sig-rt and the consistency of DKIM2-Mod declarations with the signed header set. The milter returns SMFIS_REJECT to instruct the MTA to issue a 5xx rejection, SMFIS_TEMPFAIL for transient errors, or SMFIS_CONTINUE to accept.

The inbound milter requires no persistent state between sessions - all information needed for verification is available within the current SMTP session.

Implementations using RSA-SHA256 MAY initiate DNS key lookup at EndOfHeaders as an optimization when d= and s= are available from the DKIM2-Signature header, reducing the time spent waiting for DNS resolution at EndOfBody. Implementations using Ed25519-SHA256 or supporting multiple algorithms MUST defer all DNS lookups and signature operations to EndOfBody, where the complete signed header set is available. Full signature verification including body hash comparison MUST be performed at EndOfBody regardless of algorithm.

Outbound milter - runs on the transmitting MTA, positioned last in the milter chain after all other milters that may modify the message. Operates at EndOfBody with access to:

* The complete message in its final state after all other milters

* Envelope values accumulated via MailFromRequest and RcptToRequest callbacks

* All DKIM2-Mod headers added by modifying entities earlier in the chain

Constructs DKIM2-Sig-mf and DKIM2-Sig-rt from envelope values, validates the formal correctness of any DKIM2-Mod headers present and adds the DKIM2 signature covering the final state of the message. Requires no persistent state between sessions.

The inbound milter is inactive on outbound traffic and the outbound milter is inactive on inbound traffic - this is standard milter behavior already implemented and deployed.

The two milter instances do not need to run on the same server. Each hop in the delivery chain signs independently with its own key. The inbound milter of hop N and the outbound milter of hop N+1 have no need to communicate - the chain of custody is established through the signed headers in the message itself.

3.7.2. Responsibility for Declaring Modifications

A fundamental principle of DKIM2-core is that every entity that modifies a message MUST declare its modifications via DKIM2-Mod headers at the time of modification. This responsibility belongs to the entity that makes the modification, not to the signing milter.

This principle has an important architectural consequence: the outbound milter does not need to reconstruct what was changed by comparing the current message with a previously cached version. It trusts that modifications have already been declared by whoever made them and signs the message as presented. This eliminates the need for stateful milter implementations with persistent shared storage in the DKIM2-core profile.

Implementations that delegate modification declaration to the signing milter rather than to the modifying entity - requiring the milter to infer changes by comparing with a cached copy - are technically possible but architecturally unsound. They couple the signing infrastructure to the modification logic in ways that create operational fragility and are incompatible with the stateless deployment model described here.

3.7.3. Mailing List Managers Delegating to an MTA

When a mailing list manager such as Mailman, mlmmj or Sympa passes messages to a local MTA for transmission, the recommended pattern is:

* The list manager modifies the message - adding List-* headers, modifying Subject, appending footer
* The list manager adds DKIM2-Mod headers declaring each modification it made
* The list manager submits the message to the local MTA, forcing the MAIL FROM to the list bounce address
* The outbound milter on the MTA constructs DKIM2-Sig-mf and DKIM2-Sig-rt from the envelope values and signs the message

List managers that cannot be modified to add DKIM2-Mod headers MAY rely on the outbound milter to detect undeclared modifications by comparing the signed headers against the incoming DKIM2 signature. However this approach requires stateful milter operation and is therefore classified as DKIM2-extended behavior. It is NOT part of the DKIM2-core profile.

3.7.4. Mailing List Managers with Integrated SMTP

Some mailing list managers - including mlmmj and similar lightweight implementations - can open SMTP connections directly without passing through a local MTA. These implementations act as their own MTA for the purpose of message transmission and MUST implement DKIM2-core signing directly, without relying on an external milter.

Such implementations MUST:

* Declare all modifications made to the message via DKIM2-Mod headers before signing

* Construct DKIM2-Sig-mf from the MAIL FROM value used in the SMTP transaction

* Construct DKIM2-Sig-rt from the RCPT TO values used in the SMTP transaction

* Sign the message with a key authorized for the signing domain

Alternatively, these implementations MAY be configured to relay through a local MTA that carries an outbound milter, delegating signing responsibility to that MTA. This is the RECOMMENDED approach for operators who wish to minimize the scope of DKIM2-core implementation.

3.7.5. Hop Counting and Multiple Hops Within the Same Administrative Domain

When a message passes through multiple MTAs within the same administrative domain - for example, a receiving MTA that passes to a list manager that passes to a transmitting MTA - each SMTP transaction that adds a new DKIM2 signature constitutes a new hop with an incremented i= value.

Operators MAY choose not to add a DKIM2 signature at intermediate hops within their own administrative domain if the intermediate hop does not modify the message and does not need to be independently attributed in the chain of custody. Transparent internal relays that add only Received: headers do not need to participate in DKIM2 signing.

However, any hop that modifies the message - including the list manager hop - MUST be represented in the chain of custody with its own DKIM2 signature, regardless of whether it is within the same administrative domain as adjacent hops.

3.7.6. Confirmation of Milter-Based Implementability

The feasibility of implementing DKIM2-core via the milter interface without MTA core modifications has been confirmed by multiple participants in the working group discussion.

Murray Kucherawy, co-chair of the DKIM working group, confirmed publicly on the working group mailing list that core MTA modifications are not necessary to add DKIM2 support via milter - consistent with the deployment model used for DKIM1, SPF, DMARC and ARC.

G.W. Haywood, maintainer of Sendmail::PMilter - a Perl milter implementation supporting both Sendmail and Postfix - noted that milter protocol version 6 already provides the necessary primitives: adding, deleting and modifying headers and replacing the message body. He assessed that implementing DKIM2 via milter would not be a significant implementation effort once the specification stabilizes and expressed intent to implement DKIM2 support. John Levine subsequently confirmed on the working group list that the primary difference from existing DKIM in terms of milter implementation is access to envelope addresses - and that SMFIC_MAIL and SMFIC_RCPT callbacks already provide these in the milter protocol. He characterized the milter-based implementation of DKIM2 as a Small Matter Of Programming. G.W. Haywood concurred with this assessment.

A working milter implementation of DKIM2 in Perl using Sendmail::PMilter has been published by Bron Gondwana, co-author of the DKIM2 specification, in the working group interoperability repository at https://github.com/dkim2wg/interop/. This implementation demonstrates that the milter interface provides the primitives necessary for DKIM2 implementation - including Message-Instance generation and outbound signing - without MTA core modifications. The existence of this implementation confirms the milter-based deployment model described in this document, independently of whether the full DKIM2-extended profile or only DKIM2-core is implemented.

These confirmations from active MTA implementers are consistent with the DKIM2-core design principle described in this document: all mandatory functionality is expressible within the existing milter interface without requiring changes to MTA core software.

4. DKIM2-extended: Optional Profile

4.1. Overview and Scope

DKIM2-extended is a superset of DKIM2-core. A node that implements DKIM2-extended implements all of DKIM2-core plus the body recipe mechanism described in this section. DKIM2-extended is not a separate protocol - it is an additional layer of functionality built on top of the mandatory core profile.

The body recipe mechanism allows intermediaries to describe modifications made to the message body in sufficient detail to allow reconstruction of previous versions. This capability has forensic value for operators who need to investigate message handling after the fact, audit modification chains, or satisfy compliance requirements that mandate retention of original message content. Body recipes cannot provide guidance as to whether content is safe - safety determination remains in the domain of content scanning and reputation systems. The operational value of reconstruction data depends on the receiver’s capacity to process it at scale, which varies enormously across the ecosystem.

However, the body recipe mechanism is not required for the primary operational objectives identified in the DKIM working group charter: replay prevention, backscatter mitigation and modification attribution. These objectives are fully satisfied by DKIM2-core. The working group charter states that "the working group will prefer a result that is incremental to the deployed ecosystem" and that "proposed solutions are expected to be robust in terms of interoperability and scalability." DKIM2-extended should be evaluated against these criteria by operators considering deployment.

Operators who do not require forensic body reconstruction SHOULD implement DKIM2-core only. Operators who require body accountability for compliance or forensic purposes MAY implement DKIM2-extended, subject to the operational considerations described in this section.

A note on enforcement models: the deployment of DKIM2-extended cannot rely on the policy decisions of individual large providers as an enforcement mechanism. [RFC3935] states that the IETF works for the benefit of the Internet as a whole, not for the interests of particular entities. A protocol whose correct operation depends on major receivers choosing to penalise non-conformant implementations introduces a dependency that is outside the scope of the protocol specification and that structurally disadvantages operators who lack leverage with those receivers. DKIM2-core provides verifiable security properties through cryptographic mechanisms alone, independent of any provider’s enforcement decisions. This ensures that small operators, universities, regional ISPs and non-commercial organisations can participate on equal terms.

4.2. Body Recipes and Message-Instance Headers

The body recipe mechanism is described in detail in [I-D.ietf-dkim-dkim2-spec]. This section summarizes the mechanism and identifies operational considerations relevant to deployment decisions.

A Message-Instance header is added by each hop that modifies the message body. It contains a JSON-encoded recipe - a structured description of the modifications made - encoded in base64. The recipe provides sufficient information for a verifier to reconstruct the previous version of the message body from the current version by applying the recipe in reverse.

A null recipe declares that a modification was made but that the previous state cannot or should not be reconstructed. Null recipes are discussed further in Section 4.5.

An alternative approach of storing the original message content in the MIME preamble or epilogue area has been discussed in the working group. This approach has two significant limitations identified during discussion: first, a substantial fraction of modern email - particularly bulk and transactional mail - is sent as single-part HTML without MIME boundaries, making preamble and epilogue areas unavailable; second, the use of these areas for structured signed content has not been tested and their behavior across the heterogeneous ecosystem of MTA and filtering software is unknown. The approach requires extensive testing before it could be considered for standardization.

Furthermore, proposals to store original message content in the MIME epilogue and reference it via non-monotone copy instructions would require recipe processors to access arbitrary positions in the message rather than reading it sequentially. Working group discussion has confirmed that non-monotone recipes make streaming recipe processing with bounded memory impossible, as the processor must buffer the complete message before applying any recipe step. This eliminates the streaming processing model that DKIM1 and DKIM2-core support natively.

The three architectural options for recipe transport each present fundamental limitations that cannot be resolved simultaneously. Storing recipes in message headers is subject to MTA header size limits that make the mechanism unusable for any recipe containing removed content of significant size, as documented in Section 4.3.1. Storing recipes as MIME attachments makes them visible to end users as message attachments, which is operationally unacceptable and raises additional privacy concerns. Storing recipes in the MIME preamble or epilogue requires non-monotone access patterns that make streaming processing impossible. No transport option exists that is simultaneously compatible with production MTA infrastructure, invisible to end users and streamable with bounded memory. Furthermore, any header-based recipe transport implicitly requires coordinated reconfiguration of header size limits across every MTA in the transit path - including nodes that do not implement

DKIM2-extended - to prevent silent truncation. This represents a hidden dependency on ecosystem-wide MTA updates that contradicts the incremental deployment model that DKIM2 is intended to support.

4.3. Computational and Traffic Overhead

Operators considering DKIM2-extended deployment should be aware of the following overhead costs:

JSON parsing in the delivery critical path - Message-Instance headers contain base64-encoded JSON that must be decoded and parsed at every verifying hop. JSON parsing introduces a dependency on a JSON library in the MTA or milter critical path. While JSON libraries are available in all languages, their presence in the delivery path introduces attack surface that does not exist in DKIM1 or DKIM2-core. Section 4.4 addresses the security implications.

Traffic overhead - every message that transits DKIM2-extended-aware nodes accumulates Message-Instance headers with base64-encoded JSON recipes, additional DKIM2-Signature headers and potentially substantial body recipe content. These headers travel in the message to all recipients, including those on servers that do not implement DKIM2 at all. The overhead is not optional - it is imposed on the entire ecosystem by nodes that implement DKIM2-extended, regardless of whether downstream infrastructure can use it.

Stateful milter requirement - unlike DKIM2-core, which is stateless by design, DKIM2-extended requires the signing milter to have access to the message state before and after modifications in order to calculate body recipes. This requires either a stateful milter implementation with persistent shared storage accessible to both the inbound and outbound milter instances, or MTA core modifications that provide equivalent capability. This is a significant architectural difference from DKIM2-core and from all previous email authentication protocols.

4.3.1. Recipe Size Limits and Computational Overhead

The JSON recipe format introduces complexity dimensions that must be explicitly bounded to prevent denial of service attacks. Unlike DKIM1 and ARC whose computational cost is bounded and predictable, DKIM2-extended recipe processing has a cost that depends on recipe complexity - a parameter controlled by the sender or any intermediary in the delivery chain. Verification of a complete recipe chain has complexity O(n*m) where n is the number of hops with recipes and m is the maximum size of any version of the message. Both n and m grow with message complexity and delivery path length. DKIM2-core verification is O(1) per hop regardless of chain length or message

size.

The following limits MUST be enforced by all DKIM2-extended implementations:

Maximum recipe object count

Implementations MUST enforce a maximum limit on the number of top-level objects in a recipe. During the development of this specification, a limit of 50 top-level objects was proposed as a DoS mitigation measure. This value has not been formally validated and implementations MAY choose a different limit based on their operational experience. The need for any such limit is itself evidence of the attack surface introduced by the recipe mechanism.

Maximum nesting depth

The current specification does not define a maximum JSON nesting depth. Implementations MUST enforce a maximum nesting depth of 8 levels. This is sufficient for any legitimate MIME structure - real-world messages rarely exceed 4-5 levels of MIME nesting - while preventing attacks that use artificially deep nesting to exhaust parser stack space or trigger pathological behavior in JSON libraries.

Maximum recipe size in bytes

Implementations MUST enforce a maximum size for individual recipes and for the total accumulated Message-Instance header content in a message. Until normative limits are defined in [I-D.ietf-dkim-dkim2-spec], implementations SHOULD enforce a maximum individual recipe size of 16KB and a maximum total Message-Instance header size of 32KB. These values are conservative estimates consistent with the header size limits enforced by production MTA software on default configurations.

The recipe format uses copy-range instructions for unmodified content and literal data instructions for content that has been removed or replaced. For the common case of footer addition, copy-range instructions are compact. However, when an intermediary removes content - including attachments stripped by antivirus gateways, cloud security gateways performing DLP inspection or mailing list managers applying size or content-type policies - the removed content must be represented as literal data in the recipe. A removed attachment of n bytes produces a recipe of approximately n bytes, which then travels as a base64-encoded JSON structure in the Message-Instance header to all downstream recipients. This is the inverse of the intended purpose of attachment removal.

When a removed attachment is represented as literal data in a recipe, the resulting Message-Instance header may reach sizes comparable to the removed content itself.

Production MTA header size limits

MTA implementations and milters that enforce header size limits - as most production systems do for operational reasons - may truncate or reject Message-Instance headers that exceed those limits, silently corrupting the recipe chain for all downstream verifiers. Postfix enforces a default header_size_limit of 102400 bytes per individual header; Sendmail enforces a default MaxHeadersLength of 32768 bytes for the total header block. A recipe containing a removed attachment of even modest size may exceed these limits on default-configured systems. The recipe size limits discussed in this section are therefore not merely a denial-of-service mitigation but a practical necessity imposed by the constraints of existing MTA infrastructure. However, any size limit that is low enough to be operationally safe is also low enough to exclude legitimate use cases involving common attachment removals and any limit high enough to accommodate those cases creates unacceptable memory pressure on constrained systems.

Maximum header line count

Operators with MTA configurations that enforce limits on header count or total header size MUST be aware that accumulated Message-Instance headers across multiple hops can exceed these limits, causing silent truncation that will break recipe chain verification downstream.

The one concrete data point currently available is from an implementation demonstrated during the development of this specification: a message of six lines of plain text with a footer added produces a compact recipe. However, messages containing base64-encoded attachments require recipe content that describes base64 line-width re-alignment, which can produce Message-Instance headers substantially larger than the modified content itself. At the time of writing, no quantitative analysis of overhead across representative message types has been produced. Operators should evaluate this overhead against their specific traffic profiles before committing to DKIM2-extended deployment.

The absence of a viable solution for attachment removal was confirmed during working group discussion at the April 2026 interim meeting [DKIM-INTERIM-2026-04]. The base specification authors proposed per-MIME-part hashing as a potential mechanism to allow cryptographic elision of removed attachments without null recipes. This approach was subsequently described by the lead author of the base specification as "super expensive" and "not worth the candle" - an

acknowledgment that the body recipe mechanism as currently defined does not cover a fundamental use case and that the proposed alternative is operationally impractical. At the same meeting, Allen Robinson of Google confirmed that Google Workspace performs attachment removal actively in production - directly contradicting the characterization of this scenario as rare or legacy.

It is worth noting that all four limits defined above are operationally motivated values derived from implementation experience rather than from principled bounds inherent in the protocol design. DKIM1 and ARC require no equivalent limits because their tag-value structure has bounded complexity by design: a tag-value list of N items has exactly N items, no recursive structures and no parser state beyond the current position in the list. The fact that DKIM2-extended requires explicit limits to prevent denial of service attacks is evidence that the recipe format introduces complexity that cannot be bounded by the protocol design itself. This is a qualitative difference from all previous email authentication protocols and should be evaluated as such by operators and implementers.

Beyond CPU cycles, the requirement to reassemble fragmented Base64-encoded JSON buffers forces MTAs to move from a zero-copy or stream-oriented header processing model to a buffered model, significantly increasing the per-connection memory footprint and the pressure on memory allocation subsystems at scale. DKIM1, ARC and DKIM2-core tag-value headers can be processed in a streaming fashion with constant memory per header - each tag is read, processed and discarded. DKIM2-extended recipe processing requires accumulating the complete recipe content before any verification can begin.

At tens of thousands of transactions per second, even a modest increase in per-message processing time - one millisecond of additional JSON parsing - translates to hundreds of core-seconds per second of additional load. On a server processing 50,000 messages per second, one additional millisecond per message requires 50 additional CPU cores dedicated exclusively to recipe processing.

Containerized architectures support horizontal scaling to absorb this load, but scaling has latency. An attacker who sends a burst of messages with complex recipes can saturate the processing pool before the autoscaler responds. Operators running on-premise infrastructure without horizontal scaling - including universities, regional ISPs and small businesses, precisely the operators for whom milter-based deployment is most important - have no autoscaling fallback.

The fundamental issue is that DKIM2-extended introduces a protocol component whose computational cost is controlled by potentially adversarial input - the recipe content - rather than being bounded by the protocol design itself. DKIM1 and ARC do not have this property.

4.3.2. Base64 Re-encoding and Recipe Complexity

Working group discussion has identified the need for additional recipe types beyond the initial design, including base64-decode-and-copy operations to handle transfer encoding changes. Each additional recipe type increases parser complexity and attack surface. This pattern of complexity growth under contact with real-world deployment scenarios is consistent with the general concern raised in this section about unbounded complexity.

Base64 re-encoding with different line widths changes the body hash under both strict and relaxed canonicalization equally. Relaxed canonicalization per [RFC6376] Section 3.4.2 addresses trailing whitespace and internal whitespace compression but does not affect the CRLF line separators that determine base64 line boundaries. Re-wrapping base64 content at a different line width inserts CRLF sequences at different positions, producing a different hash regardless of canonicalization algorithm.

The complexity of base64 re-wrapping in recipe generation has been acknowledged by the authors of the base specification. Maintaining synchronization between original and modified base64 line boundaries requires tracking both representations simultaneously and fails when the original line boundary information is unavailable due to prior intermediary processing. In practice, such cases are expected to produce null recipes.

4.4. Security Considerations for Body Recipes

The body recipe mechanism introduces security considerations beyond those of DKIM2-core. Three categories of concern are relevant to deployment decisions:

JSON parsing attack surface - recipe processing introduces a JSON parser in the delivery critical path that processes untrusted external input. This creates attack surface that does not exist in DKIM2-core or DKIM1. The need for explicit resource limits, discussed in Section 4.3.1, is evidence that this attack surface is real.

Recipe chain integrity - a malicious intermediary can construct a recipe that presents a clean original message to verifiers while delivering modified content to recipients. This attack is feasible for any compromised node in the chain.

Recipe stripping - operators may strip recipe content for operational or compliance reasons, removing information that downstream verifiers depend on.

These concerns are addressed in detail with normative requirements in Section 7.3.

4.5. The Null Recipe and Its Implications

A null recipe declares that a modification was made to the message body but that the previous state cannot be reconstructed. Null recipes are the correct response for several categories of intermediary that are common in real-world deployment:

* Security gateways that rewrite URLs - the original URLs may be sensitive and should not be reconstructed

* Antivirus gateways that remove malicious attachments - the removed content should not be preserved or transmitted

* DLP systems that redact sensitive content - reconstruction would defeat the purpose of the redaction

* Legacy MTAs that perform 7bit/8bit conversion - the conversion may not be perfectly reversible

* Mailing list managers that strip attachments or filter content - Mailman supports configurable MIME type removal natively via its filter_content and filter_mime_types parameters, accessible to any list administrator without server-level configuration. Sympa supports attachment removal through custom delivery scenarios. Both represent active deployment options, not legacy configurations. A common operational variant replaces removed attachments with a URL pointing to a shared repository, preserving message flow while eliminating large or problematic content. The suggestion that lists simply reject messages containing attachments rather than stripping them does not reflect common operational practice - rejection is one option among several and stripping is frequently preferred to preserve the communication value of the message body. This is a currently deployed scenario, not a legacy or transitional case.

* Any intermediary that makes modifications it cannot or should not describe in a recipe

These categories collectively represent a substantial fraction of real-world email infrastructure. When any of these nodes is present in the delivery chain, the result is a null recipe at that hop - which provides no additional body accountability beyond the bh= change already available in DKIM2-core. If null recipes are acceptable at the nodes most likely to make substantial body modifications, the incremental benefit of DKIM2-extended over DKIM2-core for body accountability is limited to the cases where all intermediaries cooperate fully - which is the minority of real-world delivery paths.

This is not a criticism of the body recipe mechanism. It is an accurate characterization of the deployment landscape that operators need to understand before committing to DKIM2-extended infrastructure.

It is worth noting that [RFC6376] Appendix B already addressed the fragility of body signatures in DKIM1 through a pragmatic approach: relaxed canonicalization tolerates common benign transformations - whitespace normalization, line ending differences, quoted-printable to 8bit conversion - without requiring intermediaries to declare or reconstruct those changes. DKIM2's strict canonicalization and body recipe mechanism represent a deliberate departure from this pragmatism in favor of a deterministic reconstruction model. The null recipe outcome is the price of that departure: cases that relaxed canonicalization would have handled silently become explicit failures in the DKIM2-extended model. Operators should evaluate whether the forensic value of body reconstruction justifies this tradeoff for their specific deployment scenario.

The systematic use of null recipes by security gateways is not a theoretical concern - it has been confirmed empirically by a major gateway vendor participating in this working group. Philip Guenther of Proofpoint, whose products perform substantial alteration of message headers and bodies under customer security policies, has stated explicitly on the working group list that reversibility of those changes is "the opposite of a goal" for their customers and that their products will use the null recipe mechanism "when necessary" - and will not follow the specification at all if null recipes are not available as an option.

This confirmation from a major in-path gateway vendor illustrates the structural limitation of the body recipe mechanism: the nodes most likely to make substantial body modifications - security gateways, DLP systems, antivirus engines - are by design and by customer

requirement the nodes that will systematically produce null recipes. The forensic value of body reconstruction is therefore unavailable precisely at the hops where attribution would matter most.

4.6. Privacy Considerations for Body Recipes

Body recipes raise data protection concerns that operators in GDPR and equivalent jurisdictions must evaluate before deployment. These concerns are addressed in detail in Section 6. A summary relevant to the deployment decision is provided here.

Body recipes create structured, recoverable representations of previous message content that travel in the message to all downstream recipients and archiving systems. For most recipe types - range references, line counts - the privacy implications are limited. However for recipes that contain literal text from the original message, or for the specific cases of DLP redaction and URL rewriting, the recipe mechanism may cause personal data or sensitive content to circulate in a form that was not intended by the operator that originally processed it.

Operators subject to GDPR should evaluate whether body recipe generation and transmission is consistent with their data minimization obligations under Article 5 and whether their use of null recipes for sensitive content modifications is sufficient to meet their compliance requirements.

5. Transition and Interoperability

5.1. Overview

The deployment of DKIM2 will occur incrementally across a heterogeneous ecosystem that includes DKIM2-core nodes, DKIM2-extended nodes, DKIM1-only nodes and legacy nodes that implement no cryptographic authentication. This section describes the expected behavior of each node type in the presence of the others and identifies the properties that are and are not achievable during the transition period.

5.2. Node Types and Behaviors

DKIM2-core node - implements envelope binding, chain of custody, header accountability, replay prevention and DSN authentication as defined in Section 3. Adds DKIM2-Sig-mf, DKIM2-Sig-rt, DKIM2-Mod and DKIM2-Signature headers - verifies incoming DKIM2 signatures. Passes DKIM2-extended headers through without interpreting them.

DKIM2-extended node - implements all of DKIM2-core plus body recipe generation and verification as defined in Section 4. Adds Message-Instance headers with JSON-encoded recipes in addition to all DKIM2-core headers.

DKIM1-only node - implements DKIM1 [RFC6376] but not DKIM2. Passes DKIM2 headers through as unrecognized header fields. Does not add DKIM2 signatures. Does not break DKIM2 chains - it simply does not extend them.

ARC node - implements ARC [RFC8617]. ARC and DKIM2-core address overlapping problems with different mechanisms. The relationship between ARC and DKIM2-core is described in Section 5.4.

Legacy node - implements no cryptographic authentication. Passes all authentication headers through without interpreting them. Does not add signatures. Does not break chains but does not extend them.

5.3. Chain Continuity and Legacy Nodes

A legacy node in the delivery chain does not break the DKIM2 signature chain, it simply passes existing signatures through without adding new ones. A downstream receiver that encounters a message with a valid DKIM2 chain ending at a hop before the legacy node can verify the chain up to that point and apply local policy for the unsigned segment.

A legacy node that makes modifications to the message - adding or changing headers, modifying the body, rewriting URLs - represents a more significant gap in the chain of custody than a transparent relay. Such modifications are not declared via DKIM2-Mod headers and cannot be attributed to any signing domain. A downstream receiver that encounters a changed bh= value or unexpected header differences between two consecutive DKIM2 signatures can identify that a modification occurred in the gap but cannot determine what was changed or by whom. Receivers that apply strict chain of custody policies SHOULD treat gaps containing undeclared modifications with additional suspicion, even if the signatures on either side of the gap are individually valid.

This is a known limitation of the transition period. The properties achievable end-to-end depend on the composition of the delivery path:

Replay prevention - fully effective only when every hop adds a DKIM2 signature with envelope binding. A legacy node between the originator and the final recipient means that the rt= value at the last signed hop does not necessarily reflect the actual final recipient.

Backscatter prevention - fully effective only when every hop participates in DKIM2. A single legacy node breaks authenticated DSN propagation for all downstream receivers. During the transition period, receivers that encounter messages with broken DKIM2 chains MUST fall back to current DKIM1 behavior: apply local policy, generate DSNs as today.

Chain of custody - provides attribution for all hops that participate in DKIM2. Legacy hops are visible as gaps in the i= sequence. A gap in the sequence does not invalidate the chain - it identifies the segment where accountability is absent.

Header accountability - fully effective for all hops that implement DKIM2-core. Modifications made by legacy nodes are not declared but are detectable as changes in the signed header set between consecutive DKIM2 signatures.

5.4. Coexistence with DKIM1

DKIM2 reuses DKIM1 DNS key infrastructure. A domain that has DKIM1 keys deployed does not need to make DNS changes to support DKIM2 signing by an ESP or MTA acting on its behalf. This is a deliberate design decision in [I-D.ietf-dkim-dkim2-spec] that significantly reduces the barrier to adoption for domain owners.

During the transition period, messages MAY carry both DKIM1 and DKIM2 signatures. Receivers that implement only DKIM1 will verify the DKIM1 signature and ignore the DKIM2 headers. Receivers that implement DKIM2 will verify the DKIM2 chain and MAY also verify the DKIM1 signature for compatibility with existing policy frameworks such as DMARC.

5.5. Coexistence with ARC

ARC [RFC8617] and DKIM2-core address overlapping problems with different mechanisms. ARC provides a trust chain for mailing list redistribution by recording the authentication state of a message as it passes through intermediaries. DKIM2-core is a functional superset of ARC - it provides the same modification attribution and trust chain capabilities plus cryptographic envelope binding that ARC lacks.

During the transition period, nodes that implement ARC but not DKIM2-core continue to provide ARC chains independently of the DKIM2 chain. The two chains coexist without conflict but do not bridge each other - a gap in the DKIM2 chain caused by a non-participating node remains a gap regardless of whether that node implements ARC. ARC does not compensate for the absence of DKIM2 participation.

Operators that have deployed ARC should not remove it during the DKIM2 transition period. ARC continues to provide value for receivers that evaluate ARC chains as part of their local policy, independently of DKIM2 adoption status.

The limited adoption of ARC after six years of availability - approximately 10,000 signing domains compared to 9.5 million DKIM1 records as reported in [I-D.adams-arc-experiment-conclusion] - is informative for DKIM2 deployment expectations. ARC is milter-deployable and architecturally simpler than DKIM2-extended. Its adoption trajectory suggests that even milter-deployable protocols face significant ecosystem inertia. This reinforces the importance of the DKIM2-core profile: reducing deployment complexity to the minimum necessary to achieve the primary objectives of the protocol maximizes the probability of meaningful adoption.

DKIM2-core enables a model of trust based on cryptographic chain of custody rather than direct domain alignment - a model that major providers already implement empirically through ARC evaluation. The approach taken in DKIM2-core is consistent with and builds upon experimental work already deployed in production. [I-D.chuang-replay-resistant-arc] proposes extending ARC with explicit envelope binding via dara= and darn= tags in the ARC-Seal, signing SMTP recipients at each hop and declaring the forwarding path. This protocol has been implemented in production by Google on Google Groups infrastructure, as evidenced by headers observed in real message flows. DKIM2-core formalizes and extends this approach with stronger cryptographic binding and a complete chain of custody model.

5.5.1. ARC Header Selection in Practice: Google and Microsoft

The following ARC-Message-Signature was captured from a live message relayed by Google’s MTA during beta testing of a DKIM2-core milter implementation. It illustrates how production infrastructure interacts with DKIM2-core headers without any modification to Google’s software:

```
ARC-Message-Signature: i=2; a=rsa-sha256; c=relaxed/relaxed;
  d=google.com; s=arc-20240605;
  h=dkim2-sig-rt:dkim2-sig-mf:dkim2-signature
    :dkim2-authentication-results:content-transfer-encoding:message-id
    :mail-reply-to:reply-to:subject:to:from:date:mime-version
    :dkim-signature;
  bh=ZH14BpgDWH4ekZ2qz7x5de78YDPKMtMIePHA1LzPCFw=;
  fh=cFHJjx+I/cl9v81+GadEx4bWcN+lzkRpBVPfbJvMiUQ=;
  b=[...]; dara=google.com
```

Google’s ARC implementation includes dkim2-sig-rt, dkim2-sig-mf, dkim2-signature and dkim2-authentication-results in its signed header set without any explicit knowledge of DKIM2. This is the natural consequence of a defensive signing strategy: when an MTA encounters header fields that carry authentication or chain-of-custody semantics, it includes them in its ARC signature to prevent downstream manipulation. The DKIM2-core headers are recognized as structural metadata and protected accordingly.

The contrast with Microsoft Exchange Online Protection is instructive. Microsoft’s ARC implementation includes vendor-specific diagnostic headers in its signed set:

```
ARC-Message-Signature: i=2; a=rsa-sha256; c=relaxed/relaxed;
  d=microsoft.com; s=arcselector10001;
  h=From:Date:Subject:Message-ID:Content-Type:MIME-Version
    :X-MS-Exchange-AntiSpam-MessageData-ChunkCount
    :X-MS-Exchange-AntiSpam-MessageData-0
    :X-MS-Exchange-AntiSpam-MessageData-1;
  bh=LfoXw5zEEFL0p4CikYely3pA5A1KXLrtYuUMyQ04I/c=;
  b=[...]
```

By signing internal diagnostic headers, Microsoft’s ARC seal cryptographically binds downstream MTA infrastructure to Microsoft’s internal telemetry format. Any system that strips or modifies these headers - as many security gateways do - will invalidate the Microsoft ARC seal, effectively requiring the entire delivery chain to preserve proprietary diagnostic data. This is architecturally unsound: the chain of custody mechanism is being used to enforce transport of vendor-specific payload rather than to protect message identity.

The two examples illustrate a principle that applies equally to DKIM2 deployment: a protocol designed around message identity and envelope binding will be adopted and protected by existing infrastructure automatically, because the headers it produces are recognizable as structural metadata. A protocol that embeds implementation-specific complexity in its headers will produce fragile seals that break at the first hop that does not share the same implementation assumptions. This distinction argues for the minimal, identity-focused approach of DKIM2-core over approaches that bind body content or vendor-specific data into the signed set.

5.6. Incremental Deployment Path

The recommended incremental deployment path for operators adopting DKIM2-core is:

Phase 1 - outbound signing only: deploy the outbound milter to add DKIM2-Sig-mf, DKIM2-Sig-rt and DKIM2-Signature headers to outgoing messages. No inbound verification. This establishes the operator's presence in the DKIM2 ecosystem and allows downstream receivers to begin building chain of custody records.

Phase 2 - inbound verification: deploy the inbound milter to verify incoming DKIM2 signatures and record results in Authentication-Results headers. Apply local policy based on verification results. This phase can be implemented in monitoring-only mode initially - logging verification results without affecting delivery - to assess the impact before enforcing policy.

During Phase 2, the verifier SHOULD add a DKIM2-Authentication-Results header to record the outcome of chain verification. The format is:

DKIM2-Authentication-Results: i=N; authserv-id; dkim2=result

where N is the current hop index, authserv-id is the verifying server identity (e.g. dns.itb.it) and result is one of:

* pass - chain verified, envelope binding confirmed

* fail - chain invalid or envelope mismatch

* none - no DKIM2-Signature present in the received message

In Phase 2 the verifier MUST NOT reject messages on dkim2=fail but SHOULD record the result for downstream policy decisions and local monitoring. In Phase 3 (policy enforcement), a dkim2=fail result corresponds to SMFIS_REJECT during the SMTP session and no DKIM2-Authentication-Results header is written, as the message does not reach any downstream processor.

DKIM2-Authentication-Results is an implementation-defined header for transitional monitoring. It is not currently registered in the IANA header field registry [RFC3864] and its use is scoped to the transitional deployment period. The format of the definitive authentication result mechanism for DKIM2 will be specified when the IANA registry for DKIM2 authentication methods is established. This header is excluded from the hh= computation as specified in Section 3.3.5, consistently with the treatment of Authentication-Results in the DKIM2 base specification [I-D.ietf-dkim-dkim2-spec].

Phase 3 - policy enforcement: begin rejecting messages that fail DKIM2 verification according to local policy. This phase requires confidence that the majority of expected senders have deployed DKIM2 outbound signing.

Phase 4 - mailing list and intermediary participation: update mailing list managers and other intermediaries to add DKIM2-Mod headers for their modifications. This phase completes the chain of custody for messages that transit these systems.

DKIM2-extended MAY be added at any phase by operators who require body accountability, subject to the operational considerations described in Section 4.

5.7. The DMARC p=reject Mailing List Problem

The DMARC p=reject mailing list problem is a known limitation of the current email authentication ecosystem that predates DKIM2. It arises from the interaction between DMARC's domain alignment requirement and the routing changes introduced by mailing list redistribution. The problem has been extensively documented, including in [I-D.levine-dmarc-listugh], which describes forwarding failures, mailing list failures and various workarounds - none of which provide a satisfactory solution. That document concludes that the workarounds available today all impose unacceptable costs: those that preserve sender identity break DMARC alignment and those that achieve DMARC alignment do so by destroying sender identity or degrading the user experience in ways that make the message unreadable in standard mail clients. DKIM2-core addresses the structural gap that all those workarounds fail to close.

The mechanism of failure: DMARC requires that at least one of SPF or DKIM provide a pass with domain alignment to the RFC5322 From: header. When a message passes through a mailing list two failure modes are possible depending on how the list handles the From: header.

Case A - list without From: rewriting

When a mailing list redistributes a message without modifying the From: header, SPF alignment fails because the mailing list infrastructure is not authorized by the original sender's SPF record. If the list rewrites the envelope-from for bounce handling - as most do - SPF may technically pass for the list's own domain, but that pass is not aligned with the From: domain and DMARC therefore fails. Similarly, DKIM alignment fails because the list's signing domain differs from the From: domain.

It should be noted that Case A is only trivially resolved if the mailing list makes no modifications whatsoever to the message. In practice, mailing lists invariably add List-* headers (List-Id, List-Unsubscribe, List-Archive), subject tags and footers. These additions invalidate the original DKIM1 signature through header modification alone, regardless of body integrity - adding List-Unsubscribe: to a message whose DKIM1 signature covers that header field is sufficient to break alignment. Major email platforms including Gmail and Microsoft 365 now require List-Unsubscribe headers with one-click unsubscribe support to avoid messages being classified as spam and to protect the sending domain’s reputation - making this header addition a de facto mandate for any mailing list that wishes to reach subscribers at these providers without deliverability penalties. The invariant addition of this header means that the "body unchanged, signature preserved" scenario is operationally irrelevant for any compliant mailing list.

The following Authentication-Results header was observed on a message from itb.it that transited Google Groups and was delivered to a Microsoft Exchange recipient:

```
dmarc=fail header.from=itb.it
dkim=pass header.d=googlegroups.com
spf=pass smtp.mailfrom=googlegroups.com
arc=pass
```

The message body was plain text with no modifications. DMARC failed purely on alignment grounds - not because the body was modified. Microsoft delivered the message via ARC override. Body recipes cannot fix this failure - even with full body reconstruction, the signing domain googlegroups.com is not aligned with the From: domain itb.it.

Case A also applies to nested mailing lists - a list that redistributes to another list. Each list adds its own List-* headers and subject tags, producing multiple instances of the same header field. This creates the duplicate header problem documented in Section 3.3.5: signing software that relies on physical header position rather than explicit index values will compute different hashes depending on which instance it selects. This has been observed in production: OpenDKIM signing both instances of a duplicate header while Microsoft Exchange selecting only one, producing a hash mismatch and a DKIM verification failure despite the message being legitimate. DKIM2-Mod’s explicit i= and seq= indexing eliminates this failure mode entirely.

Case B - list with From: rewriting

When a mailing list replaces the From: header with its own domain, DMARC passes because the list's signing domain is now aligned with the new From: domain. The following Authentication-Results header was observed on a message from vittorio.moccia@gmail.com that transited a mailing list on itb.it and was delivered to a Microsoft Exchange recipient:

```
dkim=pass header.d=itb.it
dmarc=pass header.from=itb.it
spf=pass smtp.mailfrom=itb.it
arc=pass
```

DMARC passed completely. But the original sender's identity was destroyed - the recipient sees "Lista Utenti utenti@itb.it" instead of the original author. This is the architectural hack that makes DMARC work today for mailing lists - and it is unsound because it destroys sender identity to achieve alignment, breaking the accountability chain that DKIM2 is designed to establish.

In Case B, DMARC alignment is achieved at the cost of sender attribution. If the goal of DKIM2 is to enhance authentication, a solution that requires destroying the primary identity marker - the From: header - to function is self-defeating.

Body recipes do not resolve this problem. DMARC fails due to an alignment failure - the signing domain does not match the From: domain. Body recipes address integrity - whether the body has been modified. These are orthogonal problems. In Case A, DMARC fails on domain alignment - a problem in the header and envelope layer that body reconstruction cannot address. In Case B, DMARC passes only because the From: header was replaced - a header modification that has nothing to do with body integrity. In neither case does body reconstruction affect the DMARC outcome.

DKIM2-core offers a third path that neither Case A nor Case B provides today. A mailing list implementing DKIM2-core can:

* Retain the original From: header - preserving the original sender's identity

* Declare the addition of List-* headers and any Subject: modifications via DKIM2-Mod headers

* Sign with its own DKIM2 key, cryptographically binding the envelope and the chain of custody

* Provide receivers with a verifiable record that the list handled the message and what modifications were made

This allows receivers that evaluate DKIM2 chain of custody - as Microsoft and Google already do empirically today - to make an informed trust decision without requiring From: rewriting or body reconstruction. The trust model is not theoretical - it is already deployed at scale. Critically, it achieves this without requiring new DNS records or SMTP capabilities.

The concrete mechanism is not an override of DMARC but a transformation of the trust decision from probabilistic to deterministic. Today, receivers that accept mailing list messages despite DMARC failure do so based on reputation heuristics - an implicit judgment that the list infrastructure is trustworthy. This judgment cannot be verified cryptographically. DKIM2-core provides the substrate for a verifiable equivalent: the mailing list signs the message with its own DKIM2 key, binding its identity to the specific SMTP transaction via DKIM2-Sig-mf and DKIM2-Sig-rt and declaring any header modifications via DKIM2-Mod. The body hash bh= at each hop is not an isolated integrity check but a cryptographically attributed statement - any intermediary that modifies the body must sign the new hash with its own domain, making body modification traceable to a specific accountable identity in the chain. The receiver can then verify not just that a trusted party claims to have handled the message, but that a specific identifiable domain handled this specific message to this specific recipient with these specific declared modifications - a chain of custody that is mathematically verifiable rather than reputationally assumed. If the intermediary is trusted, the chain confirms the message transited through known hands. If the intermediary is not trusted, the chain identifies exactly who touched the message. In both cases the decision is based on cryptographic proof of the transit path, not on body content or reputation alone.

By utilizing the i= and rt= fields, DKIM2-core establishes a verifiable cryptographic link between the original message and the modified version distributed by the list. Trust is derived from the verifiable path of the message rather than from an obsolete and hidden body state. This approach maintains the "What You See Is What Was Authenticated" principle, which is abandoned by the body reconstruction mechanism.

DKIM2-core provides the cryptographic substrate necessary for an evolved DMARC policy evaluation that can recognize transitive trust through a verified chain of custody. This allows receivers to distinguish between messages that fail DMARC due to spoofing and those that fail due to legitimate mailing list redistribution where the original sender’s authentication chain remains intact - a distinction that current DMARC cannot make. The trust model is not theoretical: major providers already apply equivalent heuristics

empirically today when evaluating forwarded mail, accepting messages that fail strict DMARC alignment when the handling chain is verifiable. DKIM2-core formalizes and strengthens this existing practice by adding cryptographic envelope binding that makes the trust decision verifiable through cryptographic proof rather than dependent on external reputation systems or allow lists.

The ongoing tightening of DMARC enforcement by major providers - including the adoption of strict alignment requirements that mandate exact domain matches rather than subdomain tolerance - makes the mailing list problem more acute over time. Solutions based on From: rewriting become less viable as strict alignment prevents subdomain matches that relaxed alignment would have tolerated. Body reconstruction addresses neither strict nor relaxed alignment failures. DKIM2-core chain of custody provides the only path that preserves original sender identity while remaining compatible with evolving DMARC enforcement requirements.

Ultimately, DKIM2-core restores DMARC interoperability with mailing lists by authenticating the modification path, whereas DKIM2-extended fails to address the root cause of misalignment while introducing significant privacy and security overhead.

6. Privacy Considerations

This section addresses privacy implications of DKIM2-core and DKIM2-extended in accordance with [RFC6973], which provides guidance on privacy considerations in Internet protocol design.

6.1. DKIM2-core Privacy Properties

DKIM2-core adds the following information to messages in transit:

DKIM2-Sig-mf and DKIM2-Sig-rt headers - these fields carry the SMTP envelope values, which may include email addresses not present in the RFC5322 headers. In the common case of direct delivery, these values are already implicit in the message routing. For mailing list redistribution, the rt= value at each hop reveals the address of the individual subscriber receiving that copy of the message. This is not new information - the SMTP envelope already contains this information - but it is now carried in a signed header field that persists in the message and may be archived by downstream systems.

DKIM2-Mod headers - these fields carry previous values of modified header fields. For header modifications that involve personal data - for example a From: header that is rewritten by a mailing list - the original value is preserved in a signed header that travels with the message. Operators that modify headers containing personal data should be aware that the original values will be visible to all downstream recipients and archiving systems.

This pattern is already practiced informally in the ecosystem, as described in Section 3.3.4.

Authentication-Results headers - these fields record the outcome of DKIM2 verification at each hop. They do not contain message content or personal data beyond what is already present in the message headers.

The privacy impact of DKIM2-core is limited and proportionate to its functional objectives. The additional data carried in DKIM2-core headers is necessary for the envelope binding and chain of custody mechanisms and does not represent a material increase in the personal data surface of the message beyond what is already present in the SMTP transaction.

6.2. DKIM2-extended Privacy Considerations

DKIM2-extended raises privacy concerns that are qualitatively different from those of standard email processing. Email messages already transit through systems that read, analyze and archive them - existing data protection frameworks address this reality. What DKIM2-extended adds is the systematic creation of structured, cryptographically signed records of previous versions of message content that did not previously exist as discrete data objects. A standard email message represents the current state of a communication. A message with body recipes represents both the current state and a signed history of previous states distributed to every downstream recipient and archiving system in cryptographically immutable form.

This distinction is relevant to data protection law in the ways described below.

6.2.1. The Null Recipe Mechanism

The null recipe is the primary technical instrument available to intermediaries for managing privacy risk in DKIM2-extended deployments. An intermediary that modifies message content may declare null rather than generating a content-bearing recipe, signalling that a modification occurred and who made it without creating any structured record of the previous content.

The null recipe preserves the core accountability property of DKIM2 - the modification is declared and attributed - while avoiding the creation of personal data records that would otherwise travel with the message to all downstream recipients and archiving systems. It is the correct response in any situation where generating a content-bearing recipe would conflict with data protection obligations or organizational security policy.

The current specification does not provide normative guidance on when intermediaries are obligated to use null recipes. This document addresses that gap for specific deployment scenarios in the sections that follow.

6.2.2. Data Minimization

GDPR Article 5(1)(c) requires that personal data be adequate, relevant and limited to what is necessary for the purposes for which it is processed. The primary operational objectives of DKIM2 - replay prevention, backscatter mitigation, modification attribution - are achievable without body recipes as argued in Section 4.5. Body recipe data collection may therefore not meet the necessity test under Article 5(1)(c).

The specific minimization problem of body recipes is not that personal data is processed - it is that body recipes create a new category of structured data that did not previously exist: recoverable representations of previous message content, distributed in signed form to all downstream systems. This is qualitatively different from the processing that occurs in standard email transit and raises minimization questions that do not arise for DKIM2-core.

This is not a definitive legal assessment. Operators subject to GDPR should seek legal advice on whether their specific use of body recipes is consistent with their data minimization obligations.

6.2.3. Data Retention

GDPR Article 5(1)(e) requires that personal data be kept in a form that permits identification of data subjects for no longer than necessary. Body recipes travel in the message and may be archived by any system that receives or intercepts it - including the final recipient's mail store, compliance archiving systems and, in some jurisdictions, systems operated under lawful interception or judicial oversight obligations. Unlike the message body itself, body recipes embedded in signed headers cannot be selectively removed without invalidating the signature chain. This creates a retention problem that has no clean technical resolution: the data persists in signed form in every copy of the message held by any system that archived it, regardless of the originating organization's retention policy.

Furthermore, intermediaries that implement DKIM2-extended may find that the body recipe itself constitutes an audit trail of modifications - and in many jurisdictions, audit records are subject to mandatory retention obligations that may exceed the retention period applicable to the communication content itself. An intermediary may therefore find itself obligated to retain recipe content as an audit record for extended periods, regardless of its normal data retention policy. This obligation did not exist before DKIM2-extended because no structured modification record was created during transit.

Operators should evaluate whether their archiving systems can handle recipe content consistently with applicable obligations under [GDPR] Article 5(1)(e) and any sector-specific retention requirements in their jurisdiction.

6.2.4. DLP Systems and Body Recipes

A Data Loss Prevention system that redacts sensitive content does so precisely because that data must not circulate. A body recipe that allows reconstruction of the content before redaction creates a structured record of the very data the DLP system was designed to protect - a structural contradiction between the purpose of the system and what DKIM2-extended asks it to generate.

Operators deploying DLP systems in conjunction with DKIM2-extended MUST use null recipes as described in Section 6.2.1 for all modifications that involve redaction of sensitive content.

6.2.5. Antivirus Gateways and Body Recipes

When an antivirus gateway removes a malicious attachment, the removed content should be eliminated, not preserved. A content-bearing recipe for such a removal creates a structured record of content that should not exist downstream.

Operators deploying antivirus gateways in conjunction with DKIM2-extended MUST use null recipes as described in Section 6.2.1 for all modifications that involve removal of malicious or suspicious content.

6.2.6. URL Rewriting and Body Recipes

Security gateways that rewrite URLs generate recipes containing the original URLs, which may reveal sensitive communication content or internal infrastructure details not intended for downstream exposure.

Operators who cannot expose original URLs in recipe content MUST use null recipes as described in Section 6.2.1.

6.2.7. Compliance with Data Subject Rights

GDPR Articles 16 and 17 grant data subjects the right to rectification of inaccurate personal data and the right to erasure. Body recipes create structural conflicts with both rights that manifest differently depending on the stage of the message lifecycle. In addition to GDPR, operators must consider Directive 2002/58/EC (ePrivacy Directive) [ePrivacy], which mandates the confidentiality of communications. By creating structured, cryptographically signed records of previous body states that are durably embedded in the message itself, DKIM2-extended converts transient communication content into a verifiable content history that travels with the message to every downstream system. This conversion raises questions about the applicability of mere conduit exemptions under Article 12 of the e-Commerce Directive and its successor provisions, which condition that exemption on the intermediary not modifying the information transmitted. Generating a body recipe may be considered a form of content processing that goes beyond mere transport, potentially requiring an explicit legal basis for the creation and retention of such records at intermediate hops.

The erasure problem - the null recipe described in Section 6.2.1 is a preventive instrument: if applied consistently before personal data enters the recipe, no erasure conflict arises. However it is not a remedial instrument. Once a content-bearing recipe has been generated and distributed, the Right to Erasure under Article 17 GDPR becomes technically unenforceable: the organization that generated

the recipe can delete its own copy, but the recipe exists in cryptographically signed form in every downstream copy of the message. GDPR Article 17 imposes an obligation on the controller to erase data - but it provides no mechanism to compel deletion from systems in other administrative domains, other jurisdictions, or operated by parties who are not data controllers under the same legal framework.

Furthermore, an intermediary that generated a recipe may itself face conflicting obligations: the erasure request requires deletion, but audit trail or documentary evidence obligations - contractual, regulatory, or legal - may require retention of records of modifications made. These two obligations cannot be simultaneously fulfilled. This conflict is not a gap in the legal framework - it is a structural consequence of DKIM2-extended creating cryptographically immutable records at transit nodes.

The rectification problem - the rectification conflict is structurally irresolvable and arises specifically when the message is in archival state. During transit the message exists transiently and the problem does not arise. The problem emerges when the message is archived - by any system at any point in the delivery chain - with a recipe containing inaccurate personal data.

At that point three obligations come into direct conflict. First, GDPR Article 16 requires that the inaccurate personal data be corrected. Second, correcting the value in the recipe invalidates the cryptographic signature of the hop that generated it, which cascades through all subsequent signatures in the chain. Third, if the archive is used as certified documentary evidence - for compliance, audit, or legal purposes - its integrity must be preserved. Modifying it to fulfill the rectification request destroys its evidentiary value. Not modifying it violates the data subject’s right.

This three-way conflict between data subject rights, cryptographic integrity and documentary evidence obligations has no technical resolution within the current DKIM2-extended architecture. It does not arise in standard email archiving, where an organization can modify or delete its own archives without affecting cryptographic chains, because standard email archives do not carry immutable signed records of previous content versions.

Joint controllership - an intermediary that generates body recipes is no longer merely transporting the message - it is creating a new structured record of personal data by determining what previous content to include in the recipe and ensuring its persistence through cryptographic binding. This may constitute joint controllership

under GDPR Article 26, with associated obligations including the potential requirement for a Data Protection Impact Assessment under Article 35. Operators should evaluate whether their recipe generation activities trigger these obligations.

6.2.8. Privacy Review Recommendation

At the time of writing, [I-D.ietf-dkim-dkim2-spec] does not include a Privacy Considerations section and the Security Considerations section is marked TBA. Subsequent versions may address some of the concerns raised here following discussion on the ietf-dkim mailing list initiated in March 2026.

For a specification intended to become an IETF Standards Track document, privacy and security considerations are required per [RFC3552] (BCP 72). This document provides suggested privacy considerations text based on [RFC6973] for consideration by the working group as a contribution to the base specification.

Note on [RFC6973]: this is an IAB document rather than an IETF document. However IAB documents on protocol design are explicitly relevant to IETF Standards Track work - the IAB and IETF coordinate on architectural and privacy matters as part of the overall Internet standards process. The privacy considerations in this document are consistent with both [RFC6973] and the security considerations requirements of [RFC3552].

6.2.9. Architectural Conclusion

The privacy considerations described in this section lead to a clear architectural conclusion: DKIM2-extended MUST remain optional and loosely coupled to DKIM2-core. This is not merely a deployment preference - it is a requirement derived from data protection principles.

An operator subject to [GDPR] and [ePrivacy] who activates DKIM2-extended takes on explicit data processing obligations for the personal data contained in body recipes, including obligations that may not arise under standard email processing - among them the creation of audit-trail records at transit nodes, potential joint controllership under GDPR Article 26 and questions about mere conduit exemptions under the e-Commerce Directive. An operator who implements only DKIM2-core has no such obligations beyond those that exist today for standard email processing. The optional nature of DKIM2-extended is therefore not a technical convenience but a legal necessity for a significant portion of the global email infrastructure.

The interoperability dimension of this concern extends beyond individual operators. The Digital Markets Act (DMA) requires gatekeepers to ensure interoperability with third-party services and prohibits technical measures that create barriers to entry for smaller operators. An email authentication standard that is only practically implementable by large providers with dedicated engineering teams and legal resources to manage GDPR obligations for body recipe processing raises questions about compliance with DMA interoperability requirements. DKIM2-core, by contrast, is implementable by any operator regardless of size and imposes no data processing obligations beyond those that exist today.

DKIM2-core and DKIM2-extended are designed to coexist cleanly. A DKIM2-core-only node passes Message-Instance headers through as opaque signed content without interpreting them, preserving the integrity of the chain without requiring participation in body recipe processing. A DKIM2-extended node adds full body recipe functionality on top of the core.

This is a deliberate application of graceful degradation: a core-only deployment does not break DKIM2-extended deployments - it simply does not extend them. The chain of custody remains intact, the envelope binding remains verifiable and the signed header accountability remains functional. Operators who cannot or choose not to implement body recipe processing - whether for technical, operational or legal reasons - remain full participants in the DKIM2 ecosystem. Activating DKIM2-extended adds capabilities; not activating it does not degrade the core functionality in any way.

This clean separation is the architectural property that makes DKIM2 deployable across the heterogeneous ecosystem and across jurisdictions with different data protection requirements.

7. Security Considerations

7.1. Security Properties of DKIM2-core

Replay prevention - a valid DKIM2 signature cannot be reused for a different recipient without breaking the chain. The cryptographic binding of RCPT TO values in DKIM2-Sig-rt to the signature at every hop makes replay attacks detectable by any conformant verifier.

DKIM2-core envelope binding uses one DKIM2-Sig-rt header per recipient address. A verifier checks that the RCPT TO of the current SMTP session matches exactly the addr= value of the corresponding DKIM2-Sig-rt header. This provides complete replay prevention - a message signed for recipients A, B and C cannot be replayed to only recipient A because the DKIM2-Sig-rt header carrying addr=A was

signed as part of a chain that also includes headers for B and C. This is a structural improvement over designs that aggregate all recipients in a single signed blob, where a subset of recipients could be used without breaking the signature.

The base specification [I-D.ietf-dkim-dkim2-spec] addresses this risk through an optional "exploded" flag (Section 8.9) that signals a message was sent to multiple recipients. However this mitigation has structural weaknesses: the flag is optional and depends on the signer including it and if the signer omits it or an attacker removes it, the protection fails. The flag also does not prevent replay to a subset of the original recipients - it only signals that multiple recipients existed. DKIM2-core's one-header-per-address design eliminates this category of attack without requiring any optional flag: the signed set of DKIM2-Sig-rt headers is cryptographically bound to the chain and any attempt to replay the message to a recipient subset or a different recipient produces a verifiable mismatch.

Sender accountability - the signing domain at each hop is cryptographically bound to the message and the envelope. A receiver can establish a verifiable chain of custody from originator to final recipient, identifying every entity that handled the message and declared its modifications.

Backscatter prevention - DSN generation is authenticated through the chain of custody. A receiver that rejects a message during the SMTP session directs the rejection to the connected peer, never to the envelope sender. This prevents backscatter to innocent sender domains.

Header integrity - modifications to header fields are declared via DKIM2-Mod headers and are covered by the signature of the modifying hop. Undeclared header modifications are detectable as inconsistencies between the signed header set and the declared modifications.

Body integrity - the bh= value at each hop provides a hash of the current message body. Changes in bh= between consecutive signatures identify hops where body modifications occurred and attribute them to the signing domain. Full body reconstruction is not provided in DKIM2-core and is not required for the primary security objectives.

7.1.1. The Binary Trust Model

The security model of DKIM2 relies on the chain of custody established by envelope binding. In operational environments, trust in an intermediary is a binary decision based on reputation and verifiable identity - there is no partial trust in email authentication. This has a direct consequence for the role of body recipes in security decisions.

If a receiver does not trust intermediary B, body recipes provide no additional assurance. B could have inserted malicious content, removed a legal disclaimer, or rewritten URLs to phishing sites - a JSON recipe documenting those changes does not make them safe. As the authors of the base specification have acknowledged, recipes "don’t stop B from adding bad things."

If a receiver does trust intermediary B, the chain of custody established by DKIM2-core is sufficient. The cryptographic binding of MAIL FROM and RCPT TO at every hop, combined with DKIM2-Mod declarations for header modifications, provides the verifiable path that enables the trust decision. This is precisely the model that Microsoft and Google already apply empirically today when evaluating forwarded mail.

Body recipes are a descriptive capability for forensic and archival purposes. Envelope binding is the mandatory security foundation. Furthermore, body recipes provide zero protection against replay attacks where the content remains identical - only the envelope binding of DKIM2-core addresses this fundamental vulnerability. In summary: if trust is binary, recipes are a descriptive luxury, not a security necessity. The separation between DKIM2-core and DKIM2-extended reflects this distinction: core provides the security properties that matter for delivery decisions, extended provides additional accountability for operators who need it and can afford the operational cost.

DKIM2 verification produces three qualitatively different outcomes that operators must distinguish in their delivery policy. A message that transits the chain without body modifications and with a complete chain of valid signatures provides the strongest assurance - full cryptographic accountability from originator to recipient. A message with body modifications declared via recipes provides weaker assurance - the modifications are attributed and reversible, but the displayed content differs from what was originally signed. A message with null recipes at one or more hops provides assurance equivalent to DKIM2-core: the chain of custody and envelope binding are intact, but no body reconstruction is possible for those hops. These three outcomes have different risk profiles and MUST NOT be treated

equivalently in delivery policy. In particular, the second and third outcomes do not provide the same level of assurance as the first for purposes of domain reputation, BIMI display or any policy that depends on content integrity rather than transit accountability. DKIM2-core provides the properties common to all three outcomes - envelope binding, chain of custody, header accountability - without requiring operators to distinguish between them or to deploy the additional infrastructure that the second outcome requires. DKIM2-extended adds the distinction between the second and third outcomes at the deployment cost documented in Section 4 - a cost that is only justified for operators with the infrastructure to benefit from body reconstruction data.

7.1.2. Implementation Robustness and Reduced Attack Surface

DKIM2-core uses the tag-value syntax of DKIM1 [RFC6376] throughout, without nested or opaque data structures such as JSON-encoded values within header fields. This design choice has direct security implications.

The elimination of multi-layered parsing - JSON-in-base64 embedded in header fields - removes a category of attack surface that does not exist in DKIM1 or ARC. Parser vulnerabilities in JSON libraries, including memory exhaustion and type confusion attacks, cannot be triggered by DKIM2-core header processing. This is consistent with the attack surface analysis in Section 7.3.1.

All DKIM2-core declarations - envelope binding, modification attribution, chain of custody - are in cleartext tag-value format, directly inspectable in mail logs without specialized tooling. This transparency supports real-time threat detection and incident response in ways that base64-encoded opaque blobs do not. The need for dedicated tooling to render DKIM2 header fields in human-readable form confirms that the current encoding is not suitable for operational diagnostics without specialised software, a property that tag-value encoding does not share.

Interoperability testing of [I-D.ietf-dkim-dkim2-spec] conducted during IETF hackathon activities in March 2026 identified multiple confirmed interop failures between independent implementations, including disagreements on header ordering for signing input and header hash computation. DKIM2-core’s use of explicit i= and seq= index values rather than positional ordering eliminates this category of implementation ambiguity.

The practical consequence of these design choices is immediate implementability. DKIM2-core header processing requires only the tag-value parser already present in every DKIM1 and ARC

implementation - no new libraries, no new data structures, no new parsing infrastructure. A conformant DKIM2-core milter can be built by extending an existing ARC milter with envelope binding and modification declaration. The incremental implementation cost above a working ARC milter is measured in a few weeks, not months. This stands in contrast to DKIM2-extended, which requires JSON parsing infrastructure, stateful message buffering and persistent shared storage, none of which are present in existing DKIM1 or ARC implementations.

7.2. Security Limitations of DKIM2-core

Legacy node gap - a legacy node in the delivery chain does not extend the DKIM2 chain. Modifications made by legacy nodes are not declared and cannot be attributed. The gap is visible as a discontinuity in the i= sequence but the content of the gap is unknown. Receivers must apply local policy for messages with gaps in the chain.

Key compromise - compromise of a signing key allows an attacker to generate valid signatures for that domain. Key rotation procedures as defined in [RFC6376] apply to DKIM2 keys. The i= sequence provides some mitigation - an attacker who generates a forged signature must insert it into a plausible position in the chain.

DNS security - DKIM2 inherits the DNS security properties of DKIM1. Key publication relies on DNS, which is subject to cache poisoning and other attacks unless DNSSEC is deployed. Operators SHOULD deploy DNSSEC for their signing domains.

Relaxed domain match - the relaxed domain match algorithm for mf= allows subaddress schemes used for bounce handling. Operators should be aware that this algorithm permits signatures from subdomains of the MAIL FROM domain, which may be exploitable if subdomain delegation is not carefully controlled.

7.3. Security Considerations for DKIM2-extended

DKIM2-extended introduces additional attack surface beyond DKIM2-core. Operators considering DKIM2-extended deployment should evaluate the following.

7.3.1. JSON Parsing Attack Surface

Message-Instance headers contain base64-encoded JSON that must be decoded and parsed at every verifying hop. JSON parsers process untrusted external input in the delivery critical path and are subject to algorithmic complexity attacks that cause excessive CPU consumption, memory exhaustion through deeply nested structures, parser inconsistency across implementations and buffer overflows in poorly implemented parsers.

A specific concern is Type Confusion: differences in JSON parser implementations regarding duplicate keys and numerical precision can cause a recipe to be interpreted differently by intermediaries and final recipients. An attacker could craft a recipe that validates correctly at intermediate hops - where the signature is verified - but is interpreted differently at the final recipient, causing signature validation to succeed on a message body that differs from what the signer intended. This attack exploits parser inconsistency rather than cryptographic weakness and cannot be mitigated by stronger signing algorithms.

The JSON recipe syntax also exhibits semantic ambiguity: the same construct - an empty array [] - is used in different contexts with different meanings. In backward-looking recipes it declares that a header field was added by the current hop and had no previous value. In forward-looking recipe proposals discussed in the working group, the same construct declares that a header field should be ignored during verification of a future message. This dual meaning requires implementations to determine the correct interpretation from context, introducing a category of parser confusion that does not exist in the tag-value formats used by DKIM1 and DKIM2-core.

The need to define explicit limits on object count, nesting depth and total recipe size - as discussed in Section 4.3.1 - demonstrates that this attack surface has been recognized. Operators MUST ensure that their JSON parsing implementation enforces strict resource limits on input size, nesting depth and object count appropriate to their operational environment. Implementations SHOULD use a JSON parser that strictly conforms to [RFC8259] and rejects input that is ambiguous under that specification - in particular, implementations MUST reject JSON objects with duplicate keys rather than silently selecting one value. Sandboxing the JSON parser from the MTA delivery process is RECOMMENDED where operationally feasible.

7.3.2. Recipe Chain Integrity

A malicious intermediary that controls a node in the delivery chain can construct a recipe that presents a clean original message to the verifier while the delivered content is malicious. This attack requires control of a node in the chain and the ability to generate a valid signature for that node's domain, but is feasible for a compromised or malicious intermediary. Verifiers MUST validate the complete recipe chain from originator to final recipient and MUST NOT rely on individual recipes in isolation.

7.3.3. Semantic Gap Between Verification and Visualization

DKIM2-extended introduces a structural discrepancy between the reconstructed body - the previous state that is cryptographically verified - and the transferred body - the modified state that is rendered to the end user. This creates a semantic gap that undermines the fundamental premise of email authentication: that the content being displayed is the content that was authenticated.

When a receiver applies a body recipe to validate a DKIM signature on a version of the message that is no longer present, a verification pass result is semantically ambiguous. The user is presented with a verified status - a positive Authentication-Results header or a trust indicator in a mail client - but the content displayed does not correspond to the cryptographically covered data.

This gap is a vector for social engineering. A compromised intermediary can craft modifications that are functionally malicious while providing a valid reconstruction recipe that produces a clean original. The receiver's verification passes on the ghost version; the user sees and acts on the malicious version.

There is no standardized mechanism to communicate a "reconstructed authentication" state to human recipients without creating UI confusion or warning fatigue. The DKIM2-extended body recipe mechanism therefore constitutes a departure from the "What You See Is What Was Signed" principle. It should be treated as a specialized tool for automated forensic processing rather than a general-purpose body integrity mechanism for end-user trust decisions.

This concern is distinct from but related to the Recipe Injection attack described in Section 7.3.4. Recipe Injection exploits the gap to authenticate a stolen message. The semantic gap concern exists even without malicious intent - any legitimate body modification creates a divergence between what was authenticated and what is displayed.

7.3.4. Attribution of Change vs. Verification of State

It has been argued that body recipes provide attribution for modifications. However, attribution of a state change is not equivalent to verification of the previous state. In mixed environments (DKIM1/DKIM2), an intermediary can provide a cryptographically consistent recipe for a state that never existed, effectively signing a fabrication. As long as the fabrication is consistent with a previously obtained DKIM1 signature, the mechanisms described in the DKIM2-extended profile validate the lie as truth.

The attack proceeds as follows: a malicious intermediary in possession of a stolen DKIM1-signed message receives a legitimate but unsigned message. It provides a signed recipe that, when applied to the legitimate message, reconstructs the stolen DKIM1 message. The intermediary's DKIM2 signature validates the recipe's integrity. The recipe validates the stolen message's DKIM1 signature. The receiver sees a valid chain and believes the trusted domain sent the stolen message in the current delivery.

DKIM2-core avoids this entirely. An intermediary declares what it received and what it changed - attestation of flow, not reconstruction of state. There is no recipe mechanism that can be used as a payload injector because there is no mechanism for claiming what the previous state was.

DKIM2-core provides a stateless chain of custody over message headers and body content as transmitted. This property is well-suited to general Internet mail flow across administrative boundaries. DKIM2-extended introduces stateful body reconstruction across those same boundaries, with the verification limitations described above. Implementers SHOULD carefully evaluate the operational, security and legal implications of deploying DKIM2-extended before adoption and MUST NOT treat a passing DKIM2-extended verification result as equivalent to verification of the reconstructed prior message state.

7.3.5. Null Recipe Ambiguity

A null recipe declares that a modification was made but provides no information about the nature or extent of the modification. A malicious intermediary can use a null recipe to conceal arbitrary body modifications while remaining nominally compliant with the protocol. Receivers that rely on body recipe verification for security decisions MUST treat null recipes as untrusted modifications equivalent to a complete body replacement.

7.3.6. Recipe Stripping

An intermediary that strips body recipe content from a message removes information that downstream verifiers depend on. The specification does not currently define how receivers should handle messages with stripped or truncated recipes. Implementations MUST handle this case gracefully without crashing or producing incorrect verification results. Stripped recipes SHOULD be treated as null recipes for the purpose of verification policy.

7.3.7. Stateful Milter Attack Surface

The persistent shared storage required by stateful DKIM2-extended milter implementations is an additional attack surface. Compromise of the shared storage allows an attacker to manipulate cached message state and cause the milter to generate incorrect recipes or signatures. Operators MUST apply appropriate access controls to stateful milter storage and MUST monitor for unexpected modifications.

7.4. Cryptographic Agility

DKIM2-core mandates support for RSA-SHA256 and Ed25519-SHA256. The architecture does not publish the hash value separately from the signature, which permits future adoption of post-quantum signing algorithms that incorporate their own hash function. Operators should monitor the development of post-quantum algorithm standards and be prepared to add support for new algorithms as they are standardized.

The use of both RSA-SHA256 and Ed25519-SHA256 in parallel is RECOMMENDED during the transition period to provide cryptographic agility. Ed25519 signatures are significantly shorter than RSA signatures and impose lower computational overhead at scale.

7.5. Timestamp Handling

DKIM2 signatures include a timestamp. Receivers MUST reject signatures with timestamps more than 5 minutes in the future. This prevents pre-generated signature replay while accommodating normal clock skew between NTP-synchronized systems. A tolerance of 5 minutes is sufficient for any NTP-synchronized infrastructure and eliminates the replay window that a looser future timestamp check would create.

Signatures with timestamps more than 15 days in the past SHOULD be treated as potential replays and rejected subject to local policy. The 15-day threshold accommodates legitimate redistribution delays

including mailing list queuing, temporary delivery failures and held messages without creating an excessive replay window. Note that mailing list redistribution introduces a new signature at the time of redistribution - the relevant timestamp is when the list signed the message, not when the original sender signed it. Operators SHOULD NOT configure thresholds beyond 15 days without explicit operational justification.

7.5.1. X-* Header Inclusion in hh=

The DKIM2 base specification [I-D.ietf-dkim-dkim2-spec] excludes all X-* header fields from the signed header set. The hh= mechanism defined in this profile takes a more granular approach: vendor-specific diagnostic headers are excluded, but all other X-* headers are included.

The exclusion covers header fields used as internal diagnostic telemetry by major mail infrastructure operators, which are legitimately stripped by security gateways and content filters at domain boundaries. Examples of such prefixes are X-MS-* (Microsoft Exchange Online Protection), X-GM-* (Google) and equivalent vendor-internal diagnostic namespaces used by other operators. Including them in hh= would cause verification failures at every hop that performs normal header hygiene, effectively requiring the entire delivery chain to preserve proprietary diagnostic data - the same architectural problem illustrated by Microsoft’s ARC implementation in Section 5.4. The list of excluded prefixes is implementation-defined and SHOULD be configurable to accommodate the diagnostic namespaces of operators in the specific deployment environment.

All remaining X-* headers are included in the hh= computation. This is a deliberate security decision. X-* headers that are not vendor diagnostic telemetry are frequently used to carry security-relevant metadata between trusted components: antispam scores, phishing detection results, authenticated user identities, original envelope information and compliance annotations. An attacker who can inject an X-* header without detection can attempt to influence downstream security decisions - suppressing a spam score, asserting a false authenticated identity, or altering retention classification. Including non-vendor X-* headers in hh= closes this attack surface: any undeclared addition or modification will cause the rollback check to fail at the next honest verifying hop.

Operators who use X-* headers for internal trust signals between their own components SHOULD ensure these headers are stripped before external delivery, independently of DKIM2.

8. IANA Considerations

Header field names defined in this profile use the DKIM2- prefix in preference to X- in accordance with [RFC6648].

This document requests the registration of the following header fields in the Permanent Message Header Field Registry maintained by IANA in accordance with [RFC3864]: DKIM2-Sig-mf, DKIM2-Sig-rt and DKIM2-Mod. These headers are part of the DKIM2-core protocol and appear in messages in transit.

8.1. DKIM2-Sig-mf

* Header field name: DKIM2-Sig-mf
* Applicable protocol: mail
* Status: standard
* Author/Change controller: IETF
* Specification document: this document, Section 3.1
* Related information: carries the SMTP MAIL FROM value bound to a DKIM2 signature at a specific hop, indexed by i=

8.2. DKIM2-Sig-rt

* Header field name: DKIM2-Sig-rt
* Applicable protocol: mail
* Status: standard
* Author/Change controller: IETF
* Specification document: this document, Section 3.1
* Related information: carries exactly one SMTP RCPT TO address per header, bound to a DKIM2 signature at a specific hop. Multiple recipients use multiple headers with incrementing v= sequence index. Indexed by i= for hop and v= for recipient sequence.

8.3. DKIM2-Mod

* Header field name: DKIM2-Mod
* Applicable protocol: mail

* Status: standard

* Author/Change controller: IETF

* Specification document: this document, Section 3.3

* Related information: declares a modification made to a header field by a Reviser node, using:

  - field= - identifies the modified header field name

  - del= - previous value, present for removals and modifications

  - new= - new value, present for additions and modifications

  - fr= - optional frame index for splitting long values across multiple headers

  - i= - hop index

  - seq= - field instance index

  - del= and new= MUST appear on separate header lines and MUST be last in the header field value

* Applicable protocol: mail

* Status: provisional

* Author/Change controller: IETF

* Specification document: this document, Section 3.7.3

* Related information: internal interoperability header used by mailing list managers to communicate the intended MAIL FROM value to the outbound signing milter. MUST be removed before external transmission. MUST NOT appear in messages delivered to external recipients. Registered provisionally to prevent namespace collision between independent implementations of the same convention.

9. References

9.1. Normative References

[I-D.ietf-dkim-dkim2-spec]
Clayton, R., Chuang, W., and B. Gondwana, "DomainKeys Identified Mail Signatures v2 (DKIM2)", Work in Progress,

Internet-Draft, draft-ietf-dkim-dkim2-spec-02, May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-dkim-dkim2-spec-02>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/rfc/rfc3552>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/rfc/rfc3864>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <https://www.rfc-editor.org/rfc/rfc5321>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/rfc/rfc5322>.

[RFC6376] Crocker, D., Ed., Hansen, T., Ed., and M. Kucherawy, Ed., "DomainKeys Identified Mail (DKIM) Signatures", STD 76, RFC 6376, DOI 10.17487/RFC6376, September 2011, <https://www.rfc-editor.org/rfc/rfc6376>.

[RFC6648] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, DOI 10.17487/RFC6648, June 2012, <https://www.rfc-editor.org/rfc/rfc6648>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[RFC8617] Andersen, K., Long, B., Ed., Blank, S., Ed., and M. Kucherawy, Ed., "The Authenticated Received Chain (ARC) Protocol", RFC 8617, DOI 10.17487/RFC8617, July 2019, <https://www.rfc-editor.org/rfc/rfc8617>.

9.2. Informative References

[DKIM-CHARTER] "IETF DKIM Working Group Charter", March 2026, <https://datatracker.ietf.org/wg/dkim/about/>.

[DKIM-INTERIM-2026-04] "DKIM Working Group Virtual Interim Meeting", 29 April 2026, <https://meetings.conf.meetecho.com/interim/?session=35406>.

[ePrivacy] European Union, "Directive 2002/58/EC (Directive on privacy and electronic communications)", July 2002, <http://data.europa.eu/eli/dir/2002/58/oj>.

[GDPR] European Union, "Regulation (EU) 2016/679 (General Data Protection Regulation)", April 2016, <http://data.europa.eu/eli/reg/2016/679/oj>.

[I-D.adams-arc-experiment-conclusion] Adams, T. and J. Levine, "Wrap-up of the ARC Experiment", Work in Progress, Internet-Draft, draft-adams-arc-experiment-conclusion-01, January 2026, <https://datatracker.ietf.org/doc/html/draft-adams-arc-experiment-conclusion-01>.

[I-D.chuang-mailing-list-modifications] Chuang, W., "Tolerating Mailing-List Modifications", Work in Progress, Internet-Draft, draft-chuang-mailing-list-modifications-04, February 2024, <https://datatracker.ietf.org/doc/html/draft-chuang-mailing-list-modifications-04>.

[I-D.chuang-replay-resistant-arc]
Chuang, W., "Replay Resistant Authenticated Receiver Chain", Work in Progress, Internet-Draft, draft-chuang-replay-resistant-arc-11, 2024, <https://datatracker.ietf.org/doc/html/draft-chuang-replay-resistant-arc-11>.

[I-D.ietf-dkim-dkim2-motivation]
Herr, T., "DKIM Version 2 Motivation", Work in Progress, Internet-Draft, draft-ietf-dkim-dkim2-motivation, 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-dkim-dkim2-motivation>.

[I-D.levine-dmarc-listugh]
Levine, J., "Mailing lists and mail forwarders vs. DMARC", Work in Progress, Internet-Draft, draft-levine-dmarc-listugh-01, August 2023, <https://datatracker.ietf.org/doc/html/draft-levine-dmarc-listugh-01>.

[RFC3935] Alvestrand, H., "A Mission Statement for the IETF", BCP 95, RFC 3935, DOI 10.17487/RFC3935, October 2004, <https://www.rfc-editor.org/rfc/rfc3935>.

[RFC5598] Crocker, D., "Internet Mail Architecture", RFC 5598, DOI 10.17487/RFC5598, July 2009, <https://www.rfc-editor.org/rfc/rfc5598>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

Appendix A. Implementation Notes (Informative)

A.1. DKIM2-Mod Header Representation

A DKIM2-Mod header declaration maps directly to a simple flat data structure. The following Go example illustrates a complete representation:

```
// HeaderMod represents a single DKIM2-Mod header declaration
type HeaderMod struct {
    HopIndex   int    // i= tag: hop sequence number
    SeqIndex   int    // seq= tag: field instance index
    FrameIndex int    // fr= tag: fragment index, 0 if not fragmented
    FieldName  string // field= tag: name of the modified header field
    DelValue   string // del= tag: previous value, empty if addition
    NewValue   string // new= tag: new value, empty if removal
}

// ModChain accumulates all DKIM2-Mod declarations for a message
// at a single hop
type ModChain []HeaderMod

// ModificationType returns the type of modification
func (m HeaderMod) ModificationType() string {
    switch {
    case m.DelValue != "" && m.NewValue != "":
        return "modification"
    case m.DelValue != "":
        return "removal"
    case m.NewValue != "":
        return "addition"
    default:
        return "invalid"
    }
}
```

Note: the ModificationType function treats empty string as absence of the tag, which is consistent with the ABNF definition dkim2-mod-value = 1*(VCHAR / WSP) that requires at least one character. A del= or new= tag with an empty value is not valid per the grammar and MUST be rejected by parsers. The "invalid" return value covers this case and any other combination where both DelValue and NewValue are empty.

All fields are primitive types. No JSON parser, no base64 decoder, no recursive structures. The complete state for a hop can be built by iterating the message headers once in O(n) time with O(1) memory per header.

A.2. Comparison: DKIM2-Mod vs JSON Header Recipe Encoding

The current DKIM2 specification encodes header modifications as JSON inside the Message-Instance header. The following is a real example from a working implementation demonstrated during the development of this specification. The r= field decoded from base64 contains:

```
   {
     "h": {
       "content-transfer-encoding": [],
       "content-type": [],
       "list-help": [],
       "list-id": [],
       "list-owner": [],
       "list-post": [],
       "list-subscribe": [],
       "list-unsubscribe": [],
       "precedence": [],
       "subject": ["s= format 23:26:28"]
     },
     "b": [[1,1]]
   }
```

This structure is not directly readable in mail logs - it requires base64 decoding followed by JSON parsing before any field can be inspected. The equivalent declaration using DKIM2-Mod headers:

```
DKIM2-Mod: i=2; seq=1; field=Subject; del="s= format 23:26:28"
DKIM2-Mod: i=2; seq=1; field=List-Id; new="dkim2.mailman.dkim2.com"
DKIM2-Mod: i=2; seq=1; field=List-Help; new="<mailto:dkim2-request@mailman.dkim2.com>"
DKIM2-Mod: i=2; seq=1; field=Precedence; new="list"
```

The Go data structures required for each approach illustrate the implementation complexity difference:

```
// DKIM2-Mod - flat structure, no JSON dependency
type HeaderMod struct {
    HopIndex   int    // i=
    SeqIndex   int    // seq=
    FrameIndex int    // fr= optional
    FieldName  string // field=
    DelValue   string // del= empty if addition
    NewValue   string // new= empty if removal
}

// JSON recipe - requires recursive structure and runtime type assertion
type HeaderRecipe struct {
    Headers map[string][]interface{} `json:"h"`
    Body    [][]int                  `json:"b"`
}

// Processing requires:
// 1. Accumulate complete base64 value across folded lines
// 2. Decode base64 into buffer
// 3. Unmarshal JSON into map with interface{} values
// 4. Type-assert each value to determine if string or empty
// 5. Apply resource limits before processing
```

The DKIM2-Mod approach requires only the tag-value parser already present in every DKIM1 and ARC implementation. The JSON recipe approach requires base64 decoding, JSON unmarshaling with dynamic type handling and resource limit enforcement as discussed in Sections 4.3.1 and 7.3.

Appendix B. Milter Implementation Notes (Informative)

The following describes the milter callback structure for a DKIM2-core implementation. The structure demonstrates that all mandatory DKIM2-core functionality is expressible within the standard milter interface without MTA core modifications. DKIM2-core requires two separate milter instances: an inbound milter for verification and an outbound milter for signing, positioned respectively first and last in the milter chain.

B.1 Inbound milter - verification path

dk2_connect, dk2_helo - connection-level callbacks. No DKIM2-specific processing.

dk2_envfrom - initializes the per-session private structure and resets all hash contexts and header reservoirs. Stores the MAIL FROM value for later verification against DKIM2-Sig-mf. On connection reuse (RSET), resets all per-message state without freeing the allocated structures.

dk2_envrcpt - called once per recipient. Appends each RCPT TO value to a per-session list for later verification against DKIM2-Sig-rt headers.

dk2_header - called once per header field. Accumulates each field in a dynamically allocated in-memory reservoir with DoS protection via a configurable maximum header count and a limit on total header memory. Includes removal of other spoofed internal headers arriving from external sources. When fr= fragmentation tags are present in DKIM2-Mod headers, accumulates fragments for reassembly at dk2_eom.

dk2_eoh - end of headers. For RSA-SHA256 implementations MAY initiate DNS key lookup as an optimization, reducing the time spent waiting at dk2_eom. Signing algorithms that require the complete input before verification, including Ed25519-SHA256 and future elliptic curve algorithms, defer all operations to dk2_eom.

dk2_body - called in chunks as the body arrives. Updates the body hash incrementally using relaxed or strict canonicalization as specified in the signature. This streaming approach means the milter never holds the message body in memory - only the hash context state is updated at each chunk. A message of arbitrary size imposes no additional memory cost beyond the fixed hash context.

dk2_eom - finalizes the body hash, fetches the public key via DNS, reconstructs the signed header input from the reservoir, adds the incomplete DKIM2-Signature header to the signed input, verifies the signature, checks DKIM2-Sig-mf against the stored MAIL FROM and checks each stored RCPT TO against a corresponding DKIM2-Sig-rt header. Verifies that each DKIM2-Mod declaration accurately reflects the modification declared at that hop. Returns SMFIS_REJECT on envelope mismatch, invalid signature or inconsistent modification declaration, SMFIS_TEMPFAIL on transient DNS error and SMFIS_CONTINUE on success.

dk2_abort, dk2_close - release all per-session resources including the header reservoir and hash contexts.

B.2 Outbound milter - signing path

dk2_connect, dk2_helo - connection-level callbacks. No DKIM2-specific processing.

dk2_envfrom - initializes the per-session private structure. Stores the MAIL FROM value for construction of DKIM2-Sig-mf.

dk2_envrcpt - called once per recipient. Appends each RCPT TO value to a per-session list for construction of DKIM2-Sig-rt headers.

dk2_header - called once per header field. Accumulates the field in the reservoir. For RSA-SHA256 implementations, simultaneously updates the signed header digest incrementally, allowing a single-pass streaming implementation. Signing algorithms that require the complete input before signing, including Ed25519-SHA256 and future elliptic curve algorithms, accumulate only in the reservoir and reconstruct the signed header input in dk2_eom.

dk2_eoh - no DKIM2-specific processing for the outbound path.

dk2_body - called in chunks. Updates the body hash incrementally using relaxed or strict canonicalization as specified in the signature. This streaming approach means the milter never holds the message body in memory - only the hash context state is updated at each chunk. A message of arbitrary size imposes no additional memory cost beyond the fixed hash context.

dk2_eom - Finalizes the body hash, constructs DKIM2-Sig-mf with i= and addr= from the MAIL FROM value, constructs one DKIM2-Sig-rt per stored RCPT TO with incrementing v= values, validates any DKIM2-Mod headers present, adds the incomplete DKIM2-Signature header to the digest, finalizes and signs, then adds the complete DKIM2-Signature to the message. For signing algorithms that require the complete input before signing, including Ed25519-SHA256 and future elliptic curve algorithms, the complete signed header input is constructed from the reservoir at this stage and signed in a single operation.

dk2_abort, dk2_close - release all per-session resources.

B.3 Stateless design confirmation

Both milter instances are fully stateless between sessions. The private structure allocated at dk2_connect carries the complete session state - envelope values, header reservoir and hash contexts - and is released at dk2_close. No shared storage between milter instances or between sessions is required at any point. No MTA core modifications are required.

DKIM2-core header and body processing is fully streaming - no message content is buffered at any point. The header reservoir holds only header field names and values, not body content. This is a fundamental architectural difference from DKIM2-extended, which requires buffering body content for recipe generation.

Acknowledgements

The author would like to thank the participants of the IETF-DKIM mailing list for the rigorous technical exchange that motivated this document.

Special thanks to Pete Resnick for his guidance on the IETF process, for encouraging the formalization of these concerns into an Internet-Draft and for clarifying the process by which any contributor may address the architectural and security implications of proposed standards.

The author acknowledges Wei Chuang for his independent convergence on the importance of human-readable envelope binding fields and Bron Gondwana for the extensive debate regarding stateful delivery models.

Valuable perspectives were provided by Philip Guenther on security gateway deployment requirements and by Steffen Nurpmeso on architectural simplification and attack surface reduction. Hannah Stern contributed detailed technical observations on base64 encoding complexity and recipe streaming limitations. John Levine and Richard Clayton are thanked for their participation in the working group discussion. R. Latimer is thanked for raising the perspective of MTA implementers and for drawing attention to the Alternate Submission Scenarios. Murray Kucherawy is thanked for his clarification on the scope of interface-level implementability in IETF protocol design.

Author’s Address

Vittorio Moccia
ITB.it
Email: v.moccia@itb.it