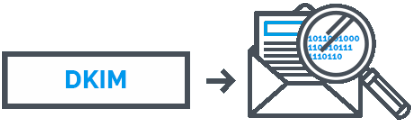
DomainKeys Identified Mails (DKIM)

- The sharpest sword in our fight against Spam (?) -
106th FRAOSUG meeting

Dr. Erwin Hoffmann
www.fehcom.de

19. September 2023



DomainKeys Identified Mails – DKIM

TELNET, FTP, SMTP, and HTTP are ASCII-based communication and transport protocols. They follow a simple command/reply scheme. While TELNET and FTP are considered obsolete and are in practice substituted by the SSH/SCP protocol family, HTTP is an end-to-end communication protocol now mostly used in the context of HTTP/2 requiring TLS encryption.

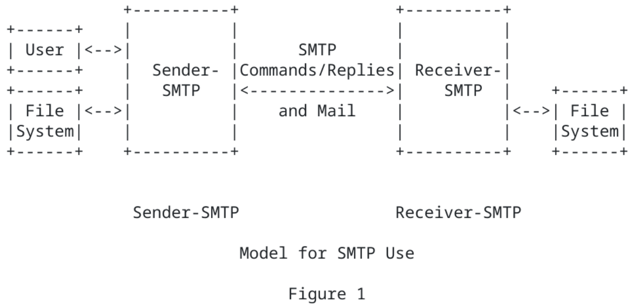
However, SMTP is a host-to-host protocol, which allows unsolicited sending of *RFC 821/822* emails/messages. Its abuse was evident from the beginning and a lot of effort has been invested to make the protocol more 'safe' and 'confidential' – though it isn't up to now.

DKIM is the last answer in email authentication and message integrity verification. It originates back to the year 2006 when *Yahoo* [1] was still a big email vendor and fought against spam waves. Under the driving force of *Yahoo* (which patented *DomainKeys*), *Alt-N* raised the C++ based libdkim [3], which was the starting point of my own **DKIM** implementation in *s/qmail* 4.2.

I will examine the **DKIM** protocol, its cornerstones, implementations, extensions, and current usage in this talk.

SMTP Email: Basically no Regulation

The sketch of RFC 821 SMTP description shows, that there is practically no restriction involved:



```
                +----------+                +----------+
    +------+    |          |                |          |
    | User |<-->|          |      SMTP      |          |
    +------+    |  Sender- |Commands/Replies| Receiver-|
    +------+    |   SMTP   |<-------------->|   SMTP   |    +------+
    | File |<-->|          |    and Mail    |          |<-->| File |
    |System|    |          |                |          |    |System|
    +------+    +----------+                +----------+    +------+


                 Sender-SMTP                Receiver-SMTP

                             Model for SMTP Use

                                  Figure 1
```

Figure: Sketch of RFC 821 SMTP component model for a Mail Transfer Agent (MTA)

- An email address is used for the forwarding and reverse path: `MAIL FROM:<Smith@Alpha.ARPA>`, `RCPT TO:<Jones@Beta.ARPA>`.
- The local part of the email address could be pseudonym, the host part has to identify a domain.

SMTP: A market place

We can consider SMTP email as a evolving market: We have users (consumers) and vendors. The big initial vendors tried to get and improve their market share:

- *AOL*[a]
- *Yahoo*
- *Hotmail*
- *Microsoft* (and much later)
- *Google* (2004/2005)[b]

and in Germany

- *T-Online*,
- *GMX*,
- *Webmail.de*.

[a] *Annalivia Ford* fought 8 years in AOL's spam tranches as 'the angry lady at AOL' [http://blog.annaliviaford.com/]

[b] https://en.wikipedia.org/wiki/History_of_Gmail

Yahoo! Mail	
E-Mail-Dienst	
Betreiber	Yahoo
Registrierung	Ja
https://overview.mail.yahoo.com	

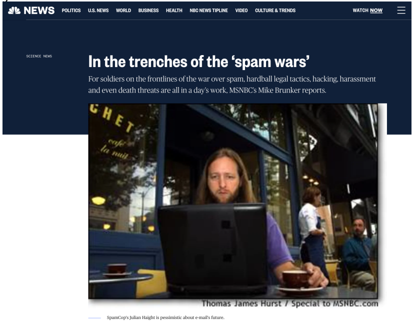
Figure: Yahoo's logo when they were still a leading force for email solutions[a]

[a] https://en.wikipedia.org/wiki/Yahoo!_Mail

↪ You are not an Internet citizen, if you don't have an *email address*.

SMTP abuse: War on Virus and Spam

- The first abuse wave was to send a *worm* over an email. The first known worm was (unintentionally) the '`CHRISTMAS.EXEC`' under IBM mail in 1987 followed by the famous '`I-LOVE-YOU`' (2000) and the '`SQL Slammer`'.
- Mail was/is used to spread viruses and to infect in particular the MS Windows operating systems. The first well known candidates were '`Nimda`' followed by '`Sobig`' and many others.



SCIENCE NEWS

In the trenches of the 'spam wars'

For soldiers on the frontlines of the war over spam, hardball legal tactics, hacking, harassment and even death threats are all in a day's work, MSNBC's Mike Brunker reports.

Thomas James Hurst / Special to MSNBC.com

SpamCop's Julian Haight is pessimistic about e-mail's future.

Figure: Article from NBC news [`https://www.nbcnews.com/id/wbna3078650`]

↪ During this period the 'originators' of this system used throw-away domains for sending the email. This was fuel for *Relay-Blacklist services* like *Spamhaus* and *Spamcop*, monetizing the idea of (S)ORBS – *Spam and Open Relay Blocking System* (2002).

SMTP abuse: War on Virus and Spam (2)

Between the years 2000 and 2010 the situation became really bad and people believed, that SMTP email will soon be dead:

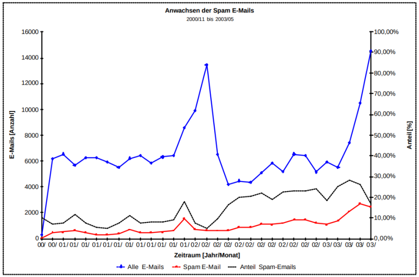
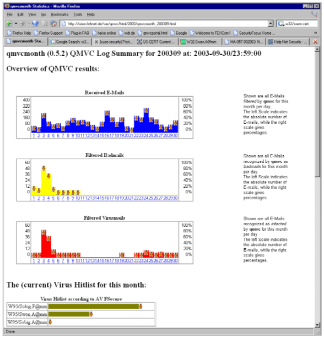


Abbildung 7.2-2: Wachstum der Anzahl der E-Mails und des Spams (Quelle: Cord Beermann)

Figure: Growing numbers of emails (blue line) and spams (red line) from year 2000 to 2003; black line provides ration among both (right axis)


qmvcmonth Statistics - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Received E-Mails
Filtered Badmails
Filtered Virusmails
Done

qmvcmonth (0.5.2) QMVC Log Summary for 200309 at: 2003-09-30/23:59:00

Overview of QMVC results:

Shown are all E-Mails *filtered* by **qmvc** for this month per day. The left Scale indicates the absolute number of E-mails, while the right scale gives percentages.

Shown are all E-Mails recognized by **qmvc** as *badmails* for this month per day. The left Scale indicates the absolute number of E-mails, while the right scale gives percentages.

Shown are all E-Mails recognized as *infected* by **qmvc** for this month per day. The left Scale indicates the absolute number of E-mails, while the right scale gives percentages.

The (current) Virus Hitlist for this month:

Virus Hitlist according to AV FSecure

W95/Sobig.F@mm
W95/Swen.A@mm
W95/Sobig.A@mm

Figure: Email volume at 'WDR.de' at 9/2003 handled by Qmail/Spamcontrol + QMVC; HTML monitoring given by the '*Virulator*'

↪ Figures taken from my (unpublished) book '*Powernetworking mit Qmail & Co.*'.

Spam Counter Means

Apart from *content-based* anti-virus and spam filters, which impact the email throughput drastically, DNS based mechanisms began to become popular:

- *Relay Blacklists* **RBL** (like *(S)ORBS*[1], *Spamhaus*, *Spamcop*): Getting customer complaints, they put the (reverse) IP address of such unwanted SMTP senders in their DNS zone: `1.0.0.127.spamcop.org`. The drawback is, that while doing the DNS query, they can profile the entire SMTP Internet traffic.
  Essentially the RBL service is a *de-legitimization* of the sender.
- Reverse MX[2] (**RMX**) - introduced by *Hadmut Danisch* (2004) - tries the opposite: Provide *additional credentials* in the Internet required for an *authorized* SMTP MTA and making it difficult for a (occasional) spammer. This draft was sabotaged by the big mail providers, but gave rise to the
- *Sender Policy Framework*[3] (**SPF**) standard, which turns out to be complicated failure and is on the list of RFCs to become retired.
- Based on Yahoo's *Domainkeys*[4] patent and also including Microsoft's Sender-ID[5] **DKIM**[6] was initally raised in RFC 4870 (shortly after revised in RFC 4871) and later became RFC 6376. **SPF** + **DKIM** → **DMARC** (RFC 7489).

[1] https://en.wikipedia.org/wiki/Spam_and_Open_Relay_Blocking_System

[2] https://de.wikipedia.org/wiki/Reverse_MX

[3] https://en.wikipedia.org/wiki/Sender_Policy_Framework

[4] https://www.eff.org/de/deeplinks/2006/02/aol-yahoo-and-goodmail-taxing-your-email-fun-and-profit

[5] https://de.wikipedia.org/wiki/Sender_ID

[6] https://en.wikipedia.org/wiki/DomainKeys_Identified_Mail

The DKIM Working Model

The idea of DKIM is in principle quite simple given the following steps:

Sender:
1. *Generate* a public key/private key pair.
2. *Publish* the domain's public key in the DNS.
3. *Sign* every outgoing mail with the adjacent private key and add a DKIM header.

Receiver:
4. *Retrieve* DKIM settings from the email.
5. *Fetch* the public key from the DNS.
6. *Verify* the mail's integrity and by the very same token *authenticate* its submitter.

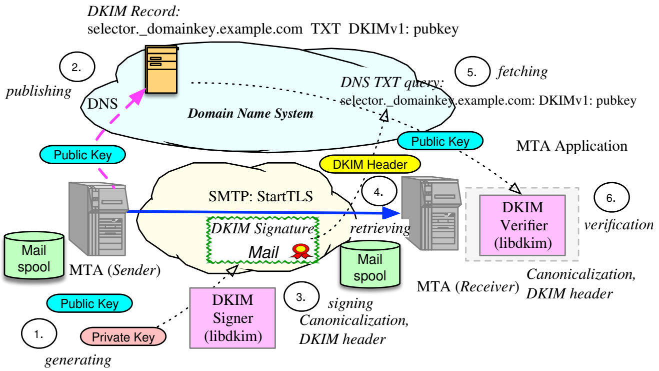


Figure: The major steps involved for using DKIM by two MTAs

How and What to DKIM Sign?

Digital Signatures are by far the most complex operations regarding asymmetrical cryptography:

- We always apply the signature algorithm against the hash value of the digital object (the message M): $h = \hbar(M)$. Therefore, we need to fix the hash function $\hbar$ and how we use the resulting hash value h.
  → We have the choice of `SHA-1` or `SHA-256`.
- We need to define the *signature algorithm*. The algorithm determines the length of the public key. This should be short since it is part of the DNS record, the `RDATA` section respectively.
  Plenty of choices are possible, but they need to be fast as well; both for signing and verification.
  → Here, we have basically the choice of `RSA` or of `ECC` (`ED25519`).
- We need to define how we apply the signature: In case of `RSA`, the resulting signature length is either 1024 or 2048 bit. Our hash input is much smaller, ie. 256 bit. Using this as input would result in very inhomogeneous distribution of the signature.
  → Thus, `PKCS#1-V1.5` is chosen here (RFC 3447).
- Furthermore, we need to determine what we sign. In DKIM we consider:
  1. The *email body* → providing integrity.
  2. A *selection of indicated email headers* including other signatures: → allowing nesting and forwarding of mails.

  → We need **canonicalization**.

DKIM Keys

DKIM keys are usually generated by means of the *OpenSSL* routines.

You have two choices:

1. Generate `RSA` keys (with a length of 1024 or 2084 bit): **genrsa** → generates both public and private key.
2. Generate `ECC` keys (with a length of 255 bit): **genkey** `-algorithm Ed25519` → private key and **pub** → public key.

Let's have a look at an `RSA` key with default length 1024 bit:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjhVSkadKX4qZSOfooKYZBfyRf
UcdA3nvuqJs3dnL70NQ2bpIse+YjWNr7Gckhy/RCnw7JiKB6wWA4AuMceYwh//I7
f2Z2qOC2pfb+qFk2q5ZY/excfm7sK9lv+7mvPxTCCMeDwlR6feVY5wni4rqMLsQS
cry+GMhhZDGwxDt6mQIDAQAB
-----END PUBLIC KEY-----
```

↪ This key – without the leading and trailing line is supposed to be published in the DNS as `TXT` record.
What are its properties and what is wrong with it (and the RFCs don't tell you)?

Let's do a **grep** and count the bytes!

DKIM DNS Interface

The public key is supposed to be published in the DNS as DKIM `TXT` record. The initial DKIM RFC 4871 defines the following structure for a `TXT` record:

- FQDN:
  `<selector>._domainkey.example.com`
- RDATA:
  `v=DKIM1;k=<sigalg>;h=<hash>;s=<service>;t=<type>;p=<BASE64 pubkey>`

Elements:

- `<selector>`: Freely chosen name for key *identification* and key *roll-over*.
- `_domainkey`: Technical label to identify the *DomainKeys* record.
- `k=<sigalg>(*)`: Signature algorithm, thus `RSA` or `ECC`.
- `h=<hash>(***)`: Hash algorithm, allowing `SHA-1` (`RSA` only) or `SHA-2`.
- `s=<service>(**)`: Service typed covered by the DKIM record.
- `t=<type>(**)`: Type of the DKIM record, i.e. `test`.
- `p=<BASE64 pubkey>`: Encoded public key; either `RSA` or `ECC`.

(*) defaults exist; (**) optional; (***) redundant.

↪ This definition has many pitfalls and trapdoors and this RFC should be accompanied by an critical addendum. Before the DNS `TXT` lookup, perform a `CNAME` query!

DKIM DNS Woes: Length Issues

One of the concerns regarding public keys in the DNS as TXT records, is its length:

- Since DNS is mainly transmitted over UDP, an UDP size of 512 byte shall be achieved, in order to avoid to switch to EDNS0 or TCP.
- Given a DNS *query*, this is not an issue.
- For the DNS *response*, this might be of concern:
  - It repeats the query.
  - It includes the answer (the RDATA field as shown).
  - It may include the Authoritative Section (*NS*).
  - It may include the Additional Section (*glue*).
- The length of the RDATA section in DKIM records is given by:
  - The length of the public key → RSA 2084 ~ 430 byte.
  - The number and types of elements (eg. service).
  - This is determining the number of *delimiters*.

Lets have a look:

```
$ dnstxt default._domainkey.fehcom.de
v=DKIM1; k=rsa;  p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjhVSkadKX4qZSOfoo
KYZBfyRfUcdA3nvuqJs3dnL70NQ2bpIse+YjWNr7Gckhy/RCnw7JiKB6wWA4AuMceYwh/
I7f2Z2qOC2pfb+qFk2q5ZYexcfm7sK9lv+7mvPxTCCMeDwlR6feVY5wni4rqMLsQScry+
GMhhZDGwxDt6mQIDAQAB
```

No current DNS implementation obeys/demands the UDP length of 512 byte any longer. Most DNS servers (and resolvers) are happy with 1200 byte and provide a large enough buffer. This is a 'digital dividend' of IPv6!

DKIM DNS Woes: TXT Issues

While programming the DKIM implementation for *s/qmail*, I needed to have a well-understood DNS test-bed. Thus, as a side effect, I enhanced *djbdnscurve6* to natively support not only TLSA but also DKIM records.

It turned out, that the structure of DNS TXT records are not defined in the DNS RFCs!

Given the long byte sequence of the DKIM TXT record, you need a *translation* from the native presentation to the DNS TXT presentation.

How does it look like? (example only)

```
\255 v=DKIM1; k=rsa;  p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjhVSkadKX4  ...
\255 KYZBfyRfUcdA3nvuqJs3dnL70NQ2bpIse+YjWNr7Gckhy/RCnw7JiKB6wWA4AuMceYwh/ ...
\023 MhhZDGwxDt6mQIDAQAB
```

DNS TXT records are broken down into slices ('labels') of maximal 255 byte each. The length of each slice is indicated via the first (unsigned) byte in octal, followed by the remaining bytes. The last slice is typically shorter.

Different DNS implementation may have different maximum label sizes, i.e. not considering the MSB. Probably this behavior is originating from BIND using a fixed buffer size for names of 255 bytes (the maximum domain name label length).

↪ 'Unstructured' i.e. plain DNS TXT records are only accepted for the first 255 byte.

DKIM DNS Woes: DKIM Fields and Token Issues

The DKIM `TXT` record may includes additional particular fields:

`v=DKIM1;k=<sigalg>;h=<hash>;s=<service>;t=<type>;p=<BASE64 pubkey>`

- Each field is separated from the following by a semicolon.
- Additional white spaces are allowed, but neglected. They simply make the `TXT` record bigger.

This can be seen in my published DKIM record:

```
$ dnstxt default._domainkey.fehcom.de
v=DKIM1; k=rsa;  p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjhVSkadKX4qZSOfoo
KYZBfyRfUcdA3nvuqJs3dnL70NQ2bpIse+YjWNr7Gckhy/RCnw7JiKB6wWA4AuMceYwh/
I7f2Z2qOC2pfb+qFk2q5ZYexcfm7sK9lv+7mvPxTCCMeDwlR6feVY5wni4rqMLsQScry+
GMhhZDGwxDt6mQIDAQAB
```

The only **mandatory** fields are:

- `v=DKIM1;`
- `p=pubkey`

↪ If the other information is missing, it defaults to `RSA` with `SHA-1`.

DKIM DNS Woes: Public Key Issue

DKIM RFC 4871 tries to convince us, that in the DNS `TXT` record the public key is given as: `p=<BASE64 pubkey>`.

This is wrong.

If you apply the OpenSSL **genrsa**[7] routines you generate:

- At first the `RSA` public key based upon,
- the 'diced' `p` and `q`, and e=65537, and finally
- using the *Euklid's Algorithm*, the private key `d` is calculated.

The result is a file including all these parameters. As a second step, both the public and the private key are stored in separate files. The output files are ASN.1 encoded, including not only the leading and trailing files, thus you can 'stack' those but also provide a label, telling what kind of ASN.1 object it is:

```
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjhVSkadKX4qZSOfoo
KYZBfyRfUcdA3nvuqJs3dnL70NQ2bpIse+YjWNr7Gckhy/RCnw7JiKB6wWA4AuMceYwh/
I7f2Z2q0C2pfb+qFk2q5ZYexcfm7sK9lv+7mvPxTCCMeDwlR6feVY5wni4rqMLsQScry+
GMhhZDGwxDt6mQIDAQAB
```

↪ Generating ECC keys based on *John Levin*'s RFC 8463, the published Ed25519 key needs to be stripped from its label, which is `MCowBQYDK2VwAyEA`. If you don't obey that, verification will fail.

```
$ openssl asn1parse -inform PEM -in private.key
```

[7] the man page has been removed in the current versions, the existing one does not tell the truth

DKIM Canonicalization

Initially, email (pre RFC 821) was an 7 bit communication protocol. All modern implementations support 8 bit bytes.

The email messages consists of two sections, the

- the email header and
- the email body.

According to RFC 2822 both sections are organized and transmitted in *lines*, with typically a length of < 78 characters and a maximum of 998 bytes. Each *line* is terminated by the ASCII `CRLF` sequence.

The email header can be 'folded' in order to become more readable: Continuation sequences can be indented, typically by a ASCII `HT`. *s/qmail* uses two white spaces.

- A Unix system delivering mails from the Internet to local users needs to 'strip' the '`CR`' after reception.
- SMTP mails to be send, need to be enriched with a '`CR`' before it is delivered.

→ In order to cope with different SMTP MTAs and their message handling, DKIM has chosen to provide some *canonicalization* of the message pre/post signature generation/verification.

r	relax (allow mangling of whitespaces and cases; default),
s	simple (=strict),
t	relax on header, simple on body,
u	simple on header, relax on body.

DKIM Signing and Header information

The DKIM signing process involves the following steps:

1. The message needs to be parsed; missing `CR` (even for empty lines) need to be added.
2. The body (including all its MIME parts) is identified and line-by-line the hash sum is incremented calculated (either `SHA-1`, or `SHA-256`, or both).
   The value is recorded in a header field:
   `bh=base64(hash);`
3. Particular message header are selected, fetched, and their hash sum is generated. The selected header fields are indicated in a new header field:
   `h=Received:Date:From:To:Subject:Message-ID;`
4. The *signature* of the common body hash and hash of the selected header fields is calculated and given as:
   `b=base64(signature)`
5. Further DKIM header fields indicate the signature algorithm and hash function used (`a=`), the way the canonicalization was achieved (`c=`), the expiration time (`x=`), and where to find the public key (in the DNS):
   ```
   DKIM-Signature:  v=1; a=rsa-sha256; c=relaxed/relaxed;
    d=fehcom.de; s=default; x=1695295824; q=dns/txt;
   ```

→ DNS `TXT`: `default._domainkey.fehcom.de`

DKIM signed Message Sample

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
  d=fehcom.de; s=default; x=1695295824; q=dns/txt; h=Received:
  Mailing-List:Precedence:list-help:list-unsubscribe:list-post:
  Delivered-To:Received:Received-SPF:Date:From:To:Subject:
  Message-ID:MIME-Version:Content-Type:Content-Disposition; bh=JO
  BJBgOCD2P2qaKx2RrUz59d3+yJ/BsP8IBHAGYPORc=; b=eAAJdvIsx3dkct1FXE
  vmiDHZDpTySpfmrdPeKDKSWSY89QQKuBOc17KaOETvBFdk9U6favUR2JHazPPrbP
  wTQZwnFeB74iVmJsZQj/uihC2k5MDZ4VWhub/uNNKzJjxtTxzmwdhGdgu6LFXgg4
  9S58UwUw76JhpoA0IDG0hRzJA=
Received: (qmail 3928722 invoked by alias); 14 Sep 2023 11:30:12 -0000
Mailing-List: contact sqmail-help@mail.fehcom.net; run by ezmlm
Precedence: bulk
X-No-Archive: yes
list-help: <mailto:sqmail-help@mail.fehcom.net>
list-unsubscribe: <mailto:sqmail-unsubscribe@mail.fehcom.net>
list-post: <mailto:sqmail@mail.fehcom.net>
Delivered-To: mailing list sqmail@mail.fehcom.net
Received: (qmail 3928715 invoked from network); 14 Sep 2023 11:30:11 -0000
Received-SPF:  pass (brunni.mail.netestate.de: domain of netestate.de
  designates 81.209.177.48 as permitted sender)
  receiver=india167; client-ip=81.209.177.48
  envelope-from=brunni@netestate.de;
Date: Thu, 14 Sep 2023 13:29:48 +0200
From: Michael Brunnbauer <xxx@netestate.de>
To: sqmail@mail.fehcom.net
Subject: Is ucspi-ssl 0.11.6a compatible with OpenSSL 3.0?
Message-ID: <ZQLurAhyDmuqJvXm@netestate.de>
MIME-Version: 1.0
Content-Type: multipart/signed; micalg=pgp-sha1;
        protocol="application/pgp-signature"; boundary="jwenCs685xv/D5bw"
Content-Disposition: inline
X-Evolution-Source: 01344637956320dc13dd427df7b77aba0b6839d2
```

hi all,
https://www.fehcom.de/ipnet/ucspi-ssl.html says that ucspi-ssl 0.12.7 supports OpenSSL 3.1 and 3.0 but I wonder whether ucspi-ssl 0.11.6a already supports OpenSSL 3.0?
Regards,

Michael Brunnbauer

DKIM Verification

For DKIM verification, the following steps need to achieved:

1. The email needs to be received (stored) and
2. potentially, `CR` characters need to be added.
3. From the email DKIM header signature parameters are determined,
4. the DKIM key(s) is/are fetched from the DNS.
5. According to the instructions in the DKIM header, the hash sums are generated – corresponding the canonicalization.
6. The calculated and the provided hash sums are compared.
7. The calculated and the provided DKIM signatures are compared.
8. An un-determined header field as *prepended* to the message, telling the state.
9. The email might be *rejected* (and *bounced*) given the local policy.

$\hookrightarrow$ DKIM key revocation is possible, in case a DKIM `TXT` record is provided, but an empty key is deployed.

DKIM verification Message Sample

```
Return-Path:
 <4814d750.AW8AACTDSzUAAAAAAAAAAGL-usMAASV4DLsAAAAAAApTqgBk8ZBC@a676778.bnc3.mailjet.com>
Delivered-To: netfehcom-hoffmann@fehcom.de
Received: (qmail 3045647 invoked from network); 1 Sep 2023 07:18:27 -0000
X-Authentication-Results: gi.de; dkim=pass; mail.fehcom.net
Received-SPF:  pass (o143.p9.mailjet.com: domain of
 a676778.bnc3.mailjet.com designates 87.253.234.143 as permitted sender)
 receiver=india167; client-ip=87.253.234.143
 envelope-from=4814d750.AW8AACTDSzUAAAAAAAAAAGL-usMAASV4DLsAAAAAAApTqgBk8ZBC@a676778.bnc3.mailjet.com;
Received: from o143.p9.mailjet.com (87.253.234.143)
  de/crypted with TLSv1.3: TLS_AES_128_GCM_SHA256 [128/128]
  DN=none
  by india167 with ESMTPS; 1 Sep 2023 07:18:27 -0000
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/simple; q=dns/txt; d=gi.de;
 i=mitgliederservice@gi.de; s=mailjet; x=1693559906;
 h=message-id:mime-version:from:from:reply-to:to:to:subject:subject:date:date:list-unsubscribe:
 list-unsubscribe-post:feedback-id:organization:x-csa-complaints:x-mj-mid:
 x-mj-smtpguid:x-report-abuse-to:content-type;
 bh=kwBBUEGfqdz8LM/moDIltaJTexdf25N9oX61NoVlpv4=;
 b=E05s68wl81Kwf3is6xECnsMOZaFeRD+PskatQUD74DRNtZapTjcLtJGUp
 ZFWpMqJj/5xZVlcOhzLJIa+n4mKFIODgNTWqeSEwYBVob7oO/u5N2JWix+C1
 Ns9bq0+H7aMzPN9h2O/iD4Iv2IvEpeIeubZkQNQzSzbAJHxFrZ+aBA=
Return-Path:
 <4814d750.AW8AACTDSzUAAAAAAAAAAGL-usMAASV4DLsAAAAAAApTqgBk8ZBC@a676778.bnc3.mailjet.com>
Message-Id: <4814d750.AW8AACTDSzUAAAAAAAAAAGL-usMAASV4DLsAAAAAAApTqgBk8ZBC@mailjet.com>
MIME-Version: 1.0
```

2. DKIM Authenticator

3. SPF Authenticator

1. DKIM Header

Figure: s/qmail's SPF and DKIM verification header

DKIM ECC Keys and Hybrid Signing

It is possible to sign an email with two different DKIM signatures. Here, we have the choice:

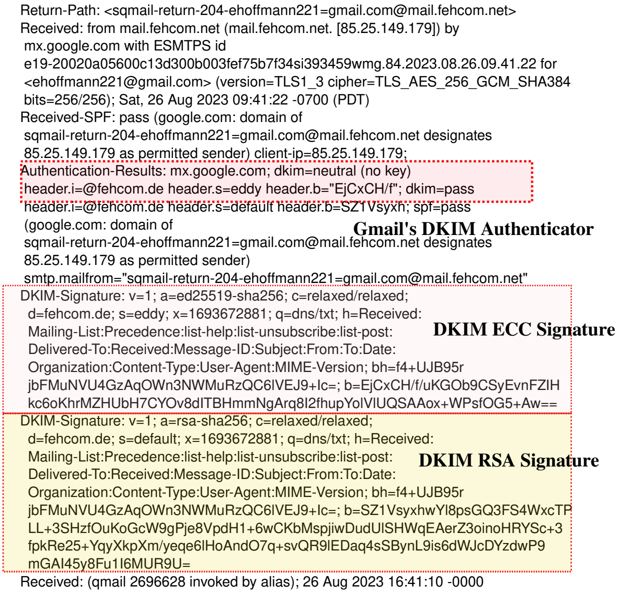
- `RSA` signature with both `SHA-1` and `SHA-256` hash sums. Makes no sense.
- `RSA` signatures with different key sizes (1024/2048 bit). Could be beneficial in case old implementations can only cope with 1024 bits or in case of DNS restrictions.
- `RSA` and `ECC` keys. Again, is helpful for backward compatibility. Until now, Gmail has no knowledge about `ECC` keys.

↪ In case of different DKIM keys different `<selectors>` need to be provided! However, different hash algorithms are happy with one key in the DNS only.

```
$ dnstxt eddy._domainkey.fehcom.de
v=DKIM1; k=ed25519; p=kznHDE2PAWcnHfs8jqMPswPJvYmeCEPiHxIt0kzSefw=

$ dnstxt default._domainkey.fehcom.de
v=DKIM1; k=rsa;  p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjhVSkadKX4qZSOfoo
KYZBfyRfUcdA3nvuqJs3dnL70NQ2bpIse+YjWNr7Gckhy/RCnw7JiKB6wWA4AuMceYwh/
I7f2Z2qOC2pfb+qFk2q5ZYexcfm7sK9lv+7mvPxTCCMeDwlR6feVY5wni4rqMLsQScry+
GMhhZDGwxDt6mQIDAQAB
```

DKIM Hybrid Signing



```
Return-Path: <sqmail-return-204-ehoffmann221=gmail.com@mail.fehcom.net>
Received: from mail.fehcom.net (mail.fehcom.net. [85.25.149.179]) by
 mx.google.com with ESMTPS id
 e19-20020a05600c13d300b003fef75b7f34si393459wmg.84.2023.08.26.09.41.22 for
 <ehoffmann221@gmail.com> (version=TLS1_3 cipher=TLS_AES_256_GCM_SHA384
 bits=256/256); Sat, 26 Aug 2023 09:41:22 -0700 (PDT)
Received-SPF: pass (google.com: domain of
 sqmail-return-204-ehoffmann221=gmail.com@mail.fehcom.net designates
 85.25.149.179 as permitted sender) client-ip=85.25.149.179;
Authentication-Results: mx.google.com; dkim=neutral (no key)
 header.i=@fehcom.de header.s=eddy header.b="EjCxCH/f"; dkim=pass
 header.i=@fehcom.de header.s=default header.b=SZ1Vsyxh; spf=pass
 (google.com: domain of
 sqmail-return-204-ehoffmann221=gmail.com@mail.fehcom.net designates
 85.25.149.179 as permitted sender)
 smtp.mailfrom="sqmail-return-204-ehoffmann221=gmail.com@mail.fehcom.net"
DKIM-Signature: v=1; a=ed25519-sha256; c=relaxed/relaxed;
  d=fehcom.de; s=eddy; x=1693672881; q=dns/txt; h=Received:
  Mailing-List:Precedence:list-help:list-unsubscribe:list-post:
  Delivered-To:Received:Message-ID:Subject:From:To:Date:
  Organization:Content-Type:User-Agent:MIME-Version; bh=f4+UJB95r
  jbFMuNVU4GzAqOWn3NWMuRzQC6lVEJ9+Ic=; b=EjCxCH/f/uKGOb9CSyEvnFZIH
  kc6oKhrMZHUbH7CYOv8dITBHmmNgArq8I2fhupYolVlUQSAAox+WPsfOG5+Aw==
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
  d=fehcom.de; s=default; x=1693672881; q=dns/txt; h=Received:
  Mailing-List:Precedence:list-help:list-unsubscribe:list-post:
  Delivered-To:Received:Message-ID:Subject:From:To:Date:
  Organization:Content-Type:User-Agent:MIME-Version; bh=f4+UJB95r
  jbFMuNVU4GzAqOWn3NWMuRzQC6lVEJ9+Ic=; b=SZ1VsyxhwYl8psGQ3FS4WxcTP
  LL+3SHzfOuKoGcW9gPje8VpdH1+6wCKbMspjiwDudUlSHWqEAerZ3oinoHRYSc+3
  fpkRe25+YqyXkpXm/yeqe6lHoAndO7q+svQR9lEDaq4sSBynL9is6dWJcDYzdwP9
  mGAI45y8Fu1I6MUR9U=
Received: (qmail 2696628 invoked by alias); 26 Aug 2023 16:41:10 -0000
```

Figure: s/qmail's hybrid signing with both ECC and RSA keys

DKIM implementations

DKIM's home page is `dkim.org`. However, it does not reference the two major (public domain) DKIM implementations I'm aware of:

- *ALT-N*'s `libdkim` [`https://libdkim.sourceforge.net/`]
- *Opendkim* [`http://www.opendkim.org/`]

Let's compare them (against *s/qmail*):

Program	**File**	**Version**	**Date**	**Size** (byte)	**Lang**	**Files**	**#words**	**#lines**	**RFCs**
libdkim	libdkim-1.0.21.zip	1.0.21	2013-04-15	55,049	C++	13	57,812	1,439	4871
Opendkim	opendkim-2.10.3.tar.gz	2.10.3	2015-05-12	1,210,224	C	230	945,661	33,437	6376
s/qmail	just DKIM	4.2.25	2023-07-19	145,152 *)	C/C++	9	36,558	967	6376,
s/qmail	sqmail-4.2.25.tgz	4.2.25	2023-07-19	399,360	C/C++	226	393,127	14,700	8463

Table: Software characteristics of some DKIM implementations; *) unpacked

$\hookrightarrow$ The *Opendkim* solution is used by *sendmail*, *postfix*, and *exim* – with the *milter* interface (`dkim-milter`).

Only `libdkim` is popular in the Qmail universe: *Indimail* (*Manvendrah*), *eQmail* (*Kai Peter*). Thus, I picked up `libdkim` for the DKIM implementation in *s/qmail*, though it is C++ code from basically 2005.

C++ DKIM Woes: The code base

Mark Delany designed DKIM [4] and gave birth to *Alt-N*'s `libdkim` based on `C++`:

- `dkim.cpp`
- `dkimbase.cpp`
- `dkimsign.cpp`
- `dkimverify.cpp`

Of course, the *autoconf* bullshit is there. The `C++` code uses mainly `STL`, some `C` interfaces are required providing access to the *libc* DNS stub library (`dnsresolv.cpp`), which – as reported – did not work well.

Pros:

- The code was compact.
- Inline documentation exists.
- The code was well structured.
- `C++` classes require to look at the header files while using 'virtual functions'.

Cons:

- The code includes old '*Allman*' fragments.
- Also `ADMD` (RFC 6541) was present in a preliminary version.
- Of course, *John Levine*'s ECC (`Ed25519`) signatures were missing (RFC 8463).

The code needed complete *refactoring* and *enhancement* to meet current RFCs. The API against `s/qmail` for signing and verifying mails needs to be defined and realized in a way a stable support, easy usage, and *multi-tenancy* can be achieved.

C++ DKIM Woes: Development

The development of DKIM for *s/qmail* was realized in mainly three steps:

1. Understanding and refactoring of the `libdkim` modules.
2. Setting up the API for *s/qmail*: DNS, mail signing, and mail verification.
3. Enhancing the DKIM algorithms to support `Ed25519` signatures.

During this development I had substantial support from *John Levine*, giving me valuable hints about the ECC implementation (no `C` or `C++` implementation did exist yet), *Kai Peter* and *Jörg Backschues* regarding the script to generate the DKIM public/private keys and their DNS deployment. Finally, *Manvendrah* did provide a solution for 'hybrid' `RSA`/`ECC` keys which fortunately could be realized in a more efficient way.

This last step required same analysis with *Valgrind*, showing some memory corruption coupling OpenSSL with `C++` *virtual function destructor* calls in `libdkim`.

Without the support of a skilled beta tester (*Pascal Novus*) I would never be able to setup a solution which was close to fulfill the requirements of a real ISP.

s/qmail's DKIM solution is certainly one of the most advanced solutions. *Manvandrah* took portions of my code for his *IndiMail*. The code this public domain and can be used for other DKIM projects as well, thus is not specific but can be used as a drop-in replacement for any existing `libdkim` installations.

s/qmail: DKIM Integration

In *s/qmail*, `libdkim` is the basic module but now renamed to **qmail-dkim** accompanied by the following modules:

- `libdkim` → **qmail-dkim** as original `C++` module but using *fehQlibs* DNS stub resolver.
- **qmail-dksign** is the `C` written interface to DKIM sign messages. It is located between the *s/qmail* queue and the **qmail-remote** send process.
- **qmail-dkverify** is the `C` interface called via the `QUEUE_EXTRA` mechanism (to be used by AV scanners and others).
- **mkdkimkey.sh** → shell script to generate `RSA` and `ECC` keys + DNS `TXT` record (BIND format).

One important design decision integrating DKIM natively into *s/qmail* was the setup of a staging area for mails to be signed, which is now part of the queue directory:

```
$ ls -la /var/qmail/queue
drwx------   2 qmails  sqmail  512 Jul 29 12:05 bounce
drwxr-x---  25 qmailq  sqmail  512 Dec  9  2022 dkim (used for signing and verification)
drwx------  25 qmails  sqmail  512 Nov 18  2022 info
drwx------  25 qmailq  sqmail  512 Nov 18  2022 intd
drwx------  25 qmails  sqmail  512 Nov 18  2022 local
drwxr-x---   2 qmailq  sqmail  512 Nov 18  2022 lock
drwxr-x---  25 qmailq  sqmail  512 Nov 18  2022 mess
drwx------   2 qmailq  sqmail  512 Sep 15 11:16 pid
drwx------  25 qmails  sqmail  512 Nov 18  2022 remote
drwxr-x---  25 qmailq  sqmail  512 Nov 18  2022 todo
```

↪ These directories typically have subdirectories for efficient mail storage: `./dkim/nm`

s/qmail: DKIM signing

DKIM signing a message needs to have an user interface:

- Given *s/qmail*'s multi-tenancy, different domains should have the possibility to use different DKIM keys and signing policies.
- Since DKIM uses *OpenSSL* (or *LibreSSL*) for signing and verification, DKIM keys and TLS keys and other key material shall be available side-by-side sharing the same access permissions.

Thus we have:

- `/var/qmail/control/dkimdomains` → define storage of DKIM keys and procedures per domain.
- `/var/qmail/control/domaincerts` → define storage of TLS keys and trust store per domain.
- `/var/qmail/ssl/domainkeys/<domain>` → storage of DKIM private keys for signing and in particular key-rollover.

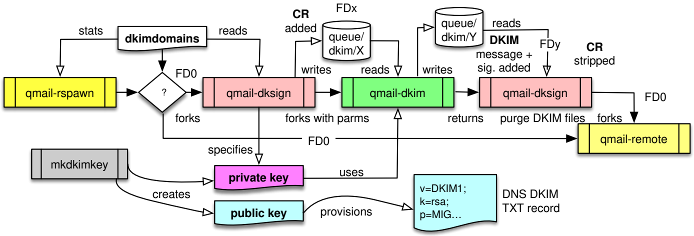


Figure: s/qmail's DKIM signing architecture

s/qmail: DKIM verification

s/qmail's DKIM verification process gave me a lot of headaches:

- How to integrate it into *s/qmail*?
- Do we really need a staging area for canonicalization?
- How to provide the results of the DKIM verification to the user?
- Shall email with wrong signatures being rejected?

The verification process shall be efficient and fast (given DDoS attacks), configurable (verify only mails from particular domains) and reliable.

DKIM verification is done by a call of **qmail-dkverify** using the `QUEUE_EXTRA` mechanism with two environment variables:

- `QMAILQUEUE="bin/qmail-dkverify"` → verify each incoming mail.
- `DKIM="+"` → reject emails, if DKIM verification will fail.

↪ Results for DKIM signed emails are displayed in the received message header:
`X-Authentication-Results:  amazon.de; dkim=pass; mail.fehcom.net`

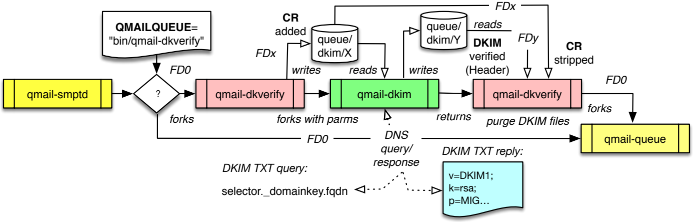


Figure: s/qmail's DKIM verification pipeline

DKIM & s/qmail: Lessons learned

Integration of DKIM into *s/qmail* and given the initial requirements involved about 1 year of work and was a major, in particular SW architectural project:

- `C++` is difficult to read, though `STL` provides some good routines.
- The conglomerate of 2005 `C++` style, `C` (**printf**) combines certainly the *worst* elements of both languages, though it seems to be efficient.
- Read the RFCs very carefully and twice!
- Go in detail through the `ABNF` syntax rules!
- But don't expect things are correctly described.
- Understand each line of code you have to refactor.
- Include useful in-line comments in the code FOR YOURSELF.
- Don't assume your test cases and environment fits everybody.
- Look for qualified beta testers.

DKIM & s/qmail: Lessons learned

Integration of DKIM into *s/qmail* and given the initial requirements involved about 1 year of work and was a major, in particular SW architectural project:

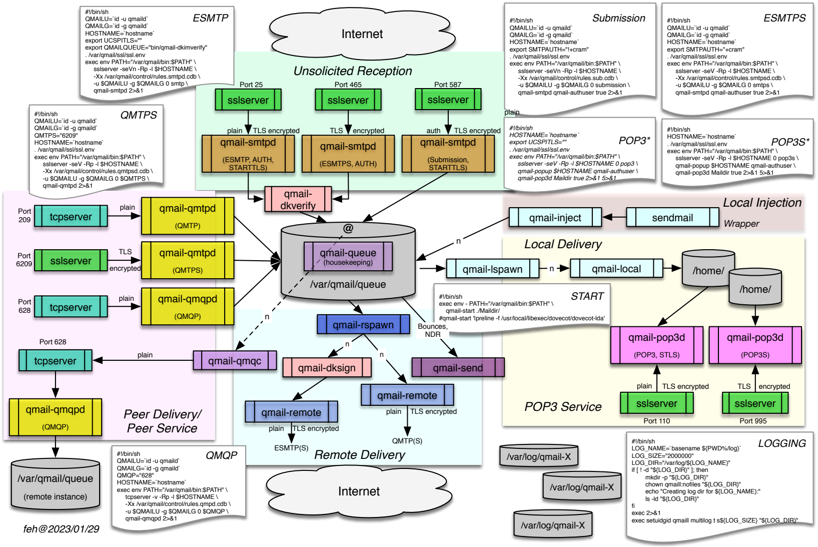
- `C++` is difficult to read, though `STL` provides some good routines.
- The conglomerate of 2005 `C++` style, `C` (**printf**) combines certainly the *worst* elements of both languages, though it seems to be efficient.
- Read the RFCs very carefully and twice!
- Go in detail through the `ABNF` syntax rules!
- But don't expect things are correctly described.
- Understand each line of code you have to refactor.
- Include useful in-line comments in the code FOR YOURSELF.
- Don't assume your test cases and environment fits everybody.
- Look for qualified beta testers.

Now, the final question: Was this project useful and did it brought some benefits?

1. DKIM signing and verifying is almost useless. Each mail has to be process at least twice → huge waste of electrical energy (like virus scanning).
2. Apart from mistakes, each DKIM signed message is verified ok.
3. It helps for my (and my *s/qmail* user's) domain to increase reputation for Gmail et al.
4. It is complete useless in order to suppress spam → You need to accept emails without DKIM signatures anyway.

Happy Binc Imapping!

DKIM & s/qmail: Final Picture



ESMTP
```
#!/bin/sh
QMAILU=`id -u qmaild`
QMAILG=`id -g qmaild`
HOSTNAME=`hostname`
export UCSPITLS=""
export QMAILQUEUE="bin/qmail-dkimverify"
. /var/qmail/ssl/ssl.env
exec env PATH="/var/qmail/bin:$PATH" \
    sslserver -seVn -Rp -l $HOSTNAME \
    -Xx /var/qmail/control/rules.smtpd.cdb \
    -u $QMAILU -g $QMAILG 0 smtp \
    qmail-smtpd 2>&1
```

QMTPS
```
#!/bin/sh
QMAILU=`id -u qmaild`
QMAILG=`id -g qmaild`
QMTPS="6209"
HOSTNAME=`hostname`
. /var/qmail/ssl/ssl.env
exec env PATH="/var/qmail/bin:$PATH" \
    sslserver -seV -Rp -l $HOSTNAME \
    -Xx /var/qmail/control/rules.qmtpsd.cdb \
    -u $QMAILU -g $QMAILG 0 $QMTPS \
    qmail-qmtpd 2>&1
```

Submission
```
#!/bin/sh
QMAILU=`id -u qmaild`
QMAILG=`id -g qmaild`
HOSTNAME=`hostname`
export SMTPAUTH="!+cram"
. /var/qmail/ssl/ssl.env
exec env PATH="/var/qmail/bin:$PATH" \
    sslserver -seVn -Rp -l $HOSTNAME \
    -Xx /var/qmail/control/rules.sub.cdb \
    -u $QMAILU -g $QMAILG 0 submission \
    qmail-smtpd qmail-authuser true 2>&1
```

ESMTPS
```
#!/bin/sh
QMAILU=`id -u qmaild`
QMAILG=`id -g qmaild`
HOSTNAME=`hostname`
export SMTPAUTH="+cram"
. /var/qmail/ssl/ssl.env
exec env PATH="/var/qmail/bin:$PATH" \
    sslserver -seV -Rp -l $HOSTNAME \
    -Xx /var/qmail/control/rules.smtpsd.cdb \
    -u $QMAILU -g $QMAILG 0 smtps \
    qmail-smtpd qmail-authuser true 2>&1
```

POP3*
```
#!/bin/sh
HOSTNAME=`hostname`
export UCSPITLS=""
. /var/qmail/ssl/ssl.env
exec env PATH="/var/qmail/bin:$PATH" \
    sslserver -seV -Rp -l $HOSTNAME 0 pop3 \
    qmail-popup $HOSTNAME qmail-authuser \
    qmail-pop3d Maildir true 2>&1 5>&1
```

POP3S*
```
#!/bin/sh
HOSTNAME=`hostname`
. /var/qmail/ssl/ssl.env
exec env PATH="/var/qmail/bin:$PATH" \
    sslserver -seV -Rp -l $HOSTNAME 0 pop3s \
    qmail-popup $HOSTNAME qmail-authuser \
    qmail-pop3d Maildir true 2>&1 5>&1
```

START
```
#!/bin/sh
exec env - PATH="/var/qmail/bin:$PATH" \
   qmail-start ./Maildir/
#qmail-start '|preline -f /usr/local/libexec/dovecot/dovecot-lda'
```

QMQP
```
#!/bin/sh
QMAILU=`id -u qmaild`
QMAILG=`id -g qmaild`
QMQP="628"
HOSTNAME=`hostname`
exec env PATH="/var/qmail/bin:$PATH" \
    tcpserver -v -Rp -l $HOSTNAME \
    -Xx /var/qmail/control/rules.qmpd.cdb \
    -u $QMAILU -g $QMAILG 0 $QMQP \
    qmail-qmqpd 2>&1
```

LOGGING
```
#!/bin/sh
LOG_NAME=`basename ${PWD%/log}`
LOG_SIZE="2000000"
LOG_DIR="/var/log/${LOG_NAME}"
if [ ! -d "${LOG_DIR}" ]; then
        mkdir -p "${LOG_DIR}"
        chown qmaill:nofiles "${LOG_DIR}"
        echo "Creating log dir for ${LOG_NAME}:"
        ls -ld "${LOG_DIR}"
fi
exec 2>&1
exec setuidgid qmaill multilog t s${LOG_SIZE} "${LOG_DIR}"
```

Figure: s/qmail's Big Picture – Binc IMAP Server to come!

References

- Yahoo DKIM patent: https://www.itprotoday.com/email-and-calendaring/yahoo-publishes-ietf-draft-domainkeys
- dkim.org: https://www.dkim.org/
- Libdkim: https://libdkim.sourceforge.net/index.html
- DKIM Summit: https://www.dkim.org/summit-1/index.html
- SPF: https://datatracker.ietf.org/doc/rfc7208/
- DKIM@Wiki: https://en.wikipedia.org/wiki/DomainKeys_Identified_Mail
- DKIM Signature RFC: https://datatracker.ietf.org/doc/html/rfc6376
- DKIM ECC RFC: https://datatracker.ietf.org/doc/html/rfc8463
- Christmas.exec worm: https://de.wikibrief.org/wiki/Christmas_Tree_EXEC
- I-Love-You worm: https://de.wikibrief.org/wiki/ILOVEYOU
- SQL-Slammer worm: https://en.wikipedia.org/wiki/SQL_Slammer