The Internet Protocol Journal

November 2016 Volume 19, Number 3

A Quarterly Technical Publication for Internet and Intranet Professionals

In This Issue

You can download IPJ back issues and find subscription information at:
www.protocoljournal.org

ISSN 1944-1134

From The Editor

Publication of this journal is made possible by numerous individuals and organizations. Every year in late August we initiate a sponsorship renewal campaign. This year our funding fell short of our sponsorship target, so we delayed publication of the September issue until now. If you are a subscriber, you should have received an e-mail asking for a donation. I am happy to say that many subscribers responded to that request (see page 42), and with the help of these individuals and our corporate sponsors we are now ready to deliver to you this *November* issue. This will be the third and final issue in 2016, but we hope to return to our regular quarterly publication schedule in 2017. We still need more individual and corporate sponsors, so please ask your company to sign up for a sponsorship or make a donation at **http://tinyurl.com/IPJ-donate**

We are pleased to welcome two new members of our Editorial Advisory Board. David Conrad is the Chief Technical Officer at the *Internet Corporation for Assigned Names and Numbers* (ICANN), and Cullen Jennings is a Cisco Fellow at Cisco Systems, Inc. We are grateful to Fred Baker, who has left Cisco Systems and our Editorial Advisory Board, and we wish him every success in the future.

A large percentage of Internet traffic is electronic mail. In our first article, William Stallings gives an overview of the many enhancements that are designed to make e-mail communication more secure and reliable in the face of an increasing amount of spam and other attack vectors.

Previous articles in IPJ have covered various aspects of *Cloud Computing* and *Software-Defined Networks* (SDNs). In our second article, Russ White and Shawn Zandi take a criticial look at the complexity of these technologies.

As always, we welcome your feedback, suggestions, book reviews, articles, and sponsorship support. You can contact us by sending an e-mail to **ipj@protocoljournal.org** and visit our website for subscription information, back issues, author guidelines, sponsor information, and much more.

—Ole J. Jacobsen, Editor and Publisher
ole@protocoljournal.org

Comprehensive Internet E-Mail Security

by William Stallings, Independent Consultant

For both organizations and individuals, e-mail is pervasive and vulnerable to a wide range of security threats. In general terms, e-mail security threats can be classified as follows:

- *Authenticity-related threats:* Could result in unauthorized access to an enterprise's e-mail system. Another threat in this category is deception, in which the purported author isn't the actual author.
- *Integrity-related threats:* Could result in unauthorized modification of e-mail content.
- *Confidentiality-related threats:* Could result in unauthorized disclosure of sensitive information.
- *Availability-related threats:* Could prevent end users from being able to send or receive e-mail messages.

To assist in addressing these threat categories, the *National Institute of Standards and Technology* (NIST) has issued SP 800-177[1], which recommends guidelines for enhancing trust in e-mail. The document is both a survey of available standardized protocols and a set of recommendations for using these protocols to counter security threats to e-mail usage.

For an understanding of the topics in this article, it is useful to have a basic grasp of the Internet mail architecture, which is currently defined in RFC 5598[2]. The discussion now provides an overview of the basic concepts.

At its most fundamental level, the Internet mail architecture consists of a user world in the form of *Message User Agents* (MUA), and the transfer world, in the form of the *Message Handling Service* (MHS), which is composed of *Message Transfer Agents* (MTA). The MHS accepts a message from one user and delivers it to one or more other users, creating a virtual MUA-to-MUA exchange environment. This architecture involves three types of interoperability. One is directly between users: messages must be formatted by the MUA on behalf of the message author so that the message can be displayed to the message recipient by the destination MUA. There are also interoperability requirements between the MUA and the MHS—first when a message is posted from an MUA to the MHS and later when it is delivered from the MHS to the destination MUA. Interoperability is required among the MTA components along the transfer path through the MHS.

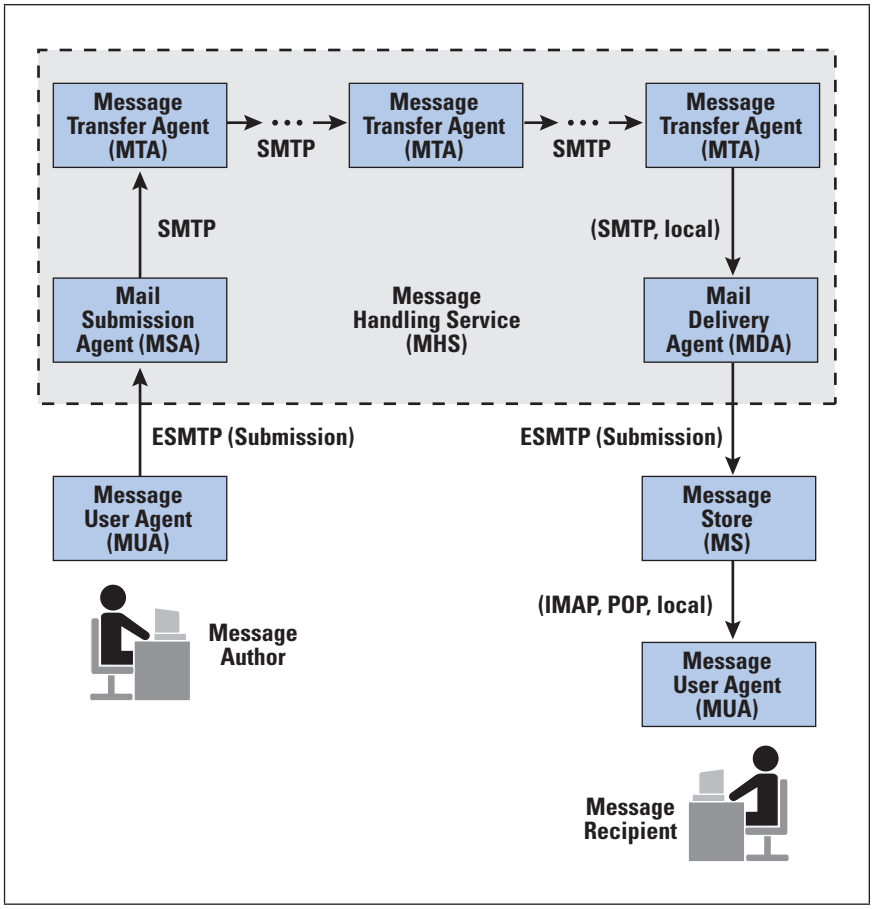


Figure 1: Function Modules and Standardized Protocols Used Between Them in the Internet Mail Architecture

Figure 1 illustrates the key components of the Internet mail architecture, which include the following:

- *Message User Agent* (MUA): Operates on behalf of user actors and user applications. It is their representative within the e-mail service. Typically, this function is housed in the user's computer and is referred to as a *client* e-mail program or a local network e-mail *server*. The author MUA formats a message and performs initial submission into the MHS via a *Mail Submission Agent* (MSA). The recipient MUA processes received mail for storage and/or display to the recipient user.
- *Mail Submission Agent* (MSA): Accepts the message submitted by an MUA and enforces the policies of the hosting domain and the requirements of Internet standards. This function may be located together with the MUA or as a separate functional model. In the latter case, the *Simple Mail Transfer Protocol* (SMTP) is used between the MUA and the MSA.
- *Message Transfer Agent* (MTA): Relays mail for one application-level hop. It is like a packet switch or IP router in that its job is to make routing assessments and move the message closer to the recipients. Relaying is performed by a sequence of MTAs until the message reaches a destination MDA. An MTA also adds trace information to the message header. SMTP is used between MTAs and between an MTA and an MSA or MDA.

- *Mail Delivery Agent* (MDA): The MDA is responsible for transferring the message from the MHS to the *Message Store* (MS).
- *Message Store* (MS): An MUA can employ a long-term MS. An MS can be located on a remote server or on the same machine as the MUA. Typically, an MUA retrieves messages from a remote server using the *Post Office Protocol* (POP) or the *Internet Message Access Protocol* (IMAP).

As will be seen subsequently, an important element in securing e-mail is the use of *public-key cryptography*. In turn, the use of public-key cryptography depends on the use of *Public-key Certificates*. In essence, a public-key certificate consists of a public key plus a user ID of the key owner, with the whole block signed by a trusted third party. A common scheme for the creation and use of public key certificates is by means of a third party known as a *Certificate Authority* (CA). A CA is an entity that is trusted by the user community, such as a government agency or a financial institution. The essential elements in the CA scheme include:

1. Client software creates a pair of keys, one public and one private. The client prepares an unsigned certificate that includes a user ID and the user's public key. The client then sends the unsigned certificate to a CA in a secure manner.
2. A CA creates a signature by calculating the hash code of the unsigned certificate and encrypting the hash code with the CA's private key; the encrypted hash code is the signature. The CA attaches the signature to the unsigned certificate and returns the now-signed certificate to the client.
3. The client may send its signed certificate to any other user. That user may verify that the certificate is valid by calculating the hash code of the certificate (not including the signature), decrypting the signature using the CA's public key, and comparing the hash code to the decrypted signature.

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users. In addition, users can transmit their certificate directly to other users. In either case, when B is in possession of A's certificate, B has confidence that messages it encrypts with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates.

Thus, with many users it may be more practical for there to be numerous CAs, each of which securely provides its public key to some fraction of the users. In practice, there is not a single CA but rather a hierarchy of CAs. This setup complicates the problems of key distribution and trust, but the basic principles are the same.

Several issues with the use of CAs should be mentioned. As can be deduced from the preceding paragraph, a hierarchical CA system can become cumbersome and not scale well. Nevertheless, this scheme is still the preferred one, and it is recommended by SP 800-177. A separate issue is one of security. The global CA ecosystem has become subject to attack in recent years, and has been successfully compromised more than once. One way to protect against CA compromises is to use the *Domain Name System* (DNS) to allow domains to specify their intended certificates or vendor CAs. Such uses of DNS require that the DNS itself be secured with *Domain Name System Security Extensions* (DNSSEC) as described subsequently.

For the reader who needs an introduction or refresher on concepts of public-key cryptography, authentication, and digital signatures, a *Crypto Portal* white paper[3] provides a quick and easy overview. A useful overview of CA and public-key certificate concepts is NIST SP 800-32[4].

Trustworthy E-Mail

The following protocols and standards are described in and recommended by SP 800-177:

- *STARTTLS:* An SMTP security extension that enables an SMTP client and server to negotiate the use of *Transport Layer Security* (TLS) to provide private, authenticated communication across the Internet.
- *Secure Multipurpose Internet Mail Extensions* (S/MIME): Provides authentication, integrity, nonrepudiation (via digital signatures) and confidentiality (via encryption) of the message body carried in SMTP messages.
- *DNS-Based Authentication of Named Entities* (DANE): Designed to overcome problems in the *Certificate Authority* (CA) system by providing an alternative channel for authenticating public keys based on DNSSEC, with the result that the same trust relationships used to certify IP addresses are used to certify servers operating on those addresses.
- *Sender Policy Framework* (SPF): Enables a domain owner to specify the IP addresses of MTAs that are authorized to send mail on its behalf. SPF uses the DNS to allow domain owners to create records that associate the domain name with a specific IP address range of authorized MTAs. It is a simple matter for receivers to check the SPF *text record* (TXT) in the DNS to confirm that the purported sender of a message is permitted to use that source address and reject mail that does not come from an authorized IP address.

- *DomainKeys Identified Mail* (DKIM): Enables e-mail actors (authors or operators) to affix their domain name to the message reliably, using cryptographic techniques, so that filtering engines can develop an accurate reputation for the domain. The MTA can sign selected headers and the body of a message. This signature validates the source domain of the mail and provides message body integrity.
- *Domain-based Message Authentication, Reporting, and Conformance* (DMARC): Publishes a requirement for the author domain name to be authenticated by DKIM and/or SPF, for that domain's owner to request recipient handling of nonauthenticated mail using that domain, and for a reporting mechanism to send reports from recipients back to domain owners. DMARC lets senders know the proportionate effectiveness of their SPF and DKIM policies, and signals to receivers what action should be taken in various individual and bulk attack scenarios.

Figure 2 shows how these components interact to provide message authenticity and integrity. Not shown, for simplicity, is that S/MIME also provides message confidentiality by encrypting messages. Together, these protocols provide a comprehensive Internet e-mail security strategy. This article provides an overview of each.

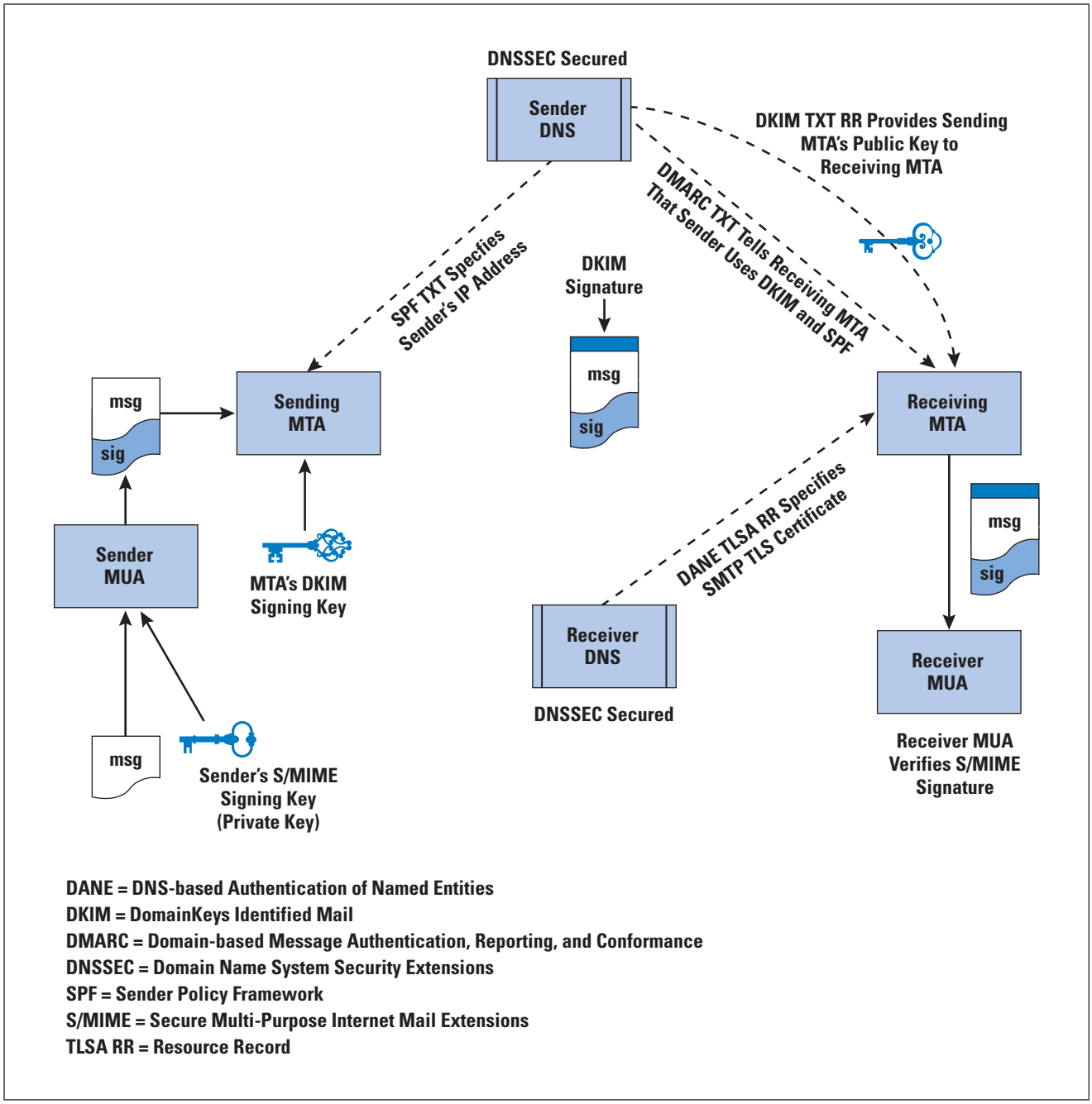
STARTTLS

A significant security-related extension for SMTP is STARTTLS, defined in RFC 3207[5]. STARTTLS enables the addition of confidentiality and authentication in the exchange between SMTP agents. This addition enables SMTP agents to protect some or all of their communications from eavesdroppers and attackers by invoking a *Transport Layer Security* (TLS) session within the SMTP connection. STARTTLS has been widely deployed, and is supported by Amazon, Facebook, Google, Microsoft, Yahoo, and others[6]. A 2014 study by Facebook, which sends several billions of e-mails daily, found that 76% of host names that receive Facebook e-mails support STARTTLS[7].

TLS is a security layer implemented just above TCP. TLS is an Internet Standard that replaces *Secure Sockets Layer* (SSL) with essentially the same functionality[8]. With TLS in place, an application has a TLS *socket address* and communicates to the TLS socket address at the remote application. These addresses are distinct from those used by the same application running directly over TCP. The security functions provided by TLS are transparent to the application and also to TCP. Thus, neither TCP nor the application needs to be modified to invoke the security features of SSL. TLS provides three categories of security, confidentiality, and authentication.

If the client does initiate the connection over a TLS-enabled port, the server may prompt with a message indicating that the STARTTLS option is available.

Figure 2: The Interrelationship of DNSSEC, SPF, DKIM, DMARC, DANE, and S/MIME for Assuring Message Authenticity and Integrity



The client can then issue the STARTTLS command in the SMTP command stream, and the two parties proceed to establish a secure TLS connection. Many e-mail providers and servers now have STARTTLS enabled[9, 10], including Amazon, Comcast, Dropbox, Facebook, Google, Microsoft, and Yahoo.

As described in SP 800-177, STARTTLS may be vulnerable to a *Man-In-The-Middle* (MITM) attack when it is initiated as a request by the server. In this case, the MITM receives the STARTTLS request from the server reply to a connection request, and scrubs it out.

The initiating client sees no TLS upgrade request and proceeds with an unsecured connection. However, SP 800-177 takes the position that some security is better than no security and that until TLS is available everywhere and automatically invoked, TLS-capable servers must prompt clients to invoke the STARTTLS command. TLS clients should attempt to either use STARTTLS initially or issue the command when requested.

S/MIME

Secure/Multipurpose Internet Mail Extension (S/MIME) is a security enhancement to the MIME Internet e-mail format standard[11]. S/MIME is a complex capability that is defined in many documents. The most important documents relevant to S/MIME include the following [12–19]:

- *RFC 5750, S/MIME Version 3.2 Certificate Handling:* Specifies conventions for X.509 certificate usage by S/MIME v3.2.
- *RFC 5751, S/MIME Version 3.2 Message Specification:* The principal defining document for S/MIME message creation and processing.
- *RFC 4134, Examples of S/MIME Messages:* Gives examples of message bodies formatted using S/MIME.
- *RFC 2634, Enhanced Security Services for S/MIME:* Describes four optional security service extensions for S/MIME.
- *RFC 5652, Cryptographic Message Syntax (CMS):* Describes CMS. This syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary message content.
- *RFC 3370, CMS Algorithms:* Describes the conventions for using several cryptographic algorithms with the CMS.
- *RFC 5752, Multiple Signatures in CMS:* Describes the use of multiple, parallel signatures for a message.
- *RFC 1847, Security Multiparts for MIME—Multipart/Signed and Multipart/Encrypted:* Defines a framework within which security services may be applied to MIME body parts. The use of a digital signature is relevant to S/MIME, as explained subsequently.

S/MIME functionality is built into most modern e-mail software and interoperates between them. S/MIME provides four message-related services: authentication, confidentiality, compression, and e-mail compatibility.

Authentication is provided by means of a digital signature. Most commonly RSA with SHA-256 is used. The sequence is as follows:

1. The sender creates a message.
2. SHA-256 is used to generate a 256-bit message digest of the message.
3. The message digest is encrypted with RSA using the sender's private key, and the result is appended to the message. Also appended is identifying information for the signer, which will enable the receiver to retrieve the signer's public key.
4. The receiver uses RSA with the sender's public key to decrypt and recover the message digest.
5. The receiver generates a new message digest for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.

The combination of SHA-256 and RSA provides an effective digital signature scheme. Because of the strength of RSA, the recipient is assured that only the possessor of the matching private key could have generated the signature. Because of the strength of SHA-256, the recipient is assured that no one else could generate a new message that matches the hash code and, hence, the signature of the original message.

Although signatures normally are found attached to the message or file that they sign, it is not always the case: Detached signatures are supported. A detached signature may be stored and transmitted separately from the message it signs. This option is useful in several contexts. A user may wish to maintain a separate signature log of all messages sent or received. A detached signature of an executable program can detect subsequent virus infection. Finally, detached signatures can be used when more than one party must sign a document, such as a legal contract. Each person's signature is independent and is therefore applied only to the document. Otherwise, signatures would have to be nested, with the second signer signing both the document and the first signature, and so on.

S/MIME provides *confidentiality* by encrypting messages using conventional encryption with a secret key, also known as a *symmetric key.* Most commonly, *Advanced Encryption Standard* (AES) with a 128-bit key is used, with the *Cipher Block Chaining* (CBC) mode. The key itself is also encrypted, typically with RSA, as explained subsequently.

As always, one must address the problem of key distribution. In S/MIME, each symmetric key, referred to as a *content-encryption key,* is used only once. That is, a new key is generated as a random number for each new message. Because it is to be used only once, the content-encryption key is bound to the message and transmitted with it. To protect the key, it is encrypted with the receiver's public key.

The sequence can be described as follows:

1. The sender generates a message and a random 128-bit number to be used as a content-encryption key for this message only.
2. The message is encrypted using the content-encryption key.
3. The content-encryption key is encrypted with RSA using the recipient's public key and is attached to the message.
4. The receiver uses RSA with its private key to decrypt and recover the content-encryption key.
5. The content-encryption key is used to decrypt the message.

As Figure 3 illustrates, both confidentiality and encryption may be used for the same message. The figure shows a sequence in which a signature is generated for the plaintext message and appended to the message. Then the plaintext message and signature are encrypted as a single block using symmetric encryption and the symmetric encryption key is encrypted using public-key encryption.

S/MIME allows the signing and message encryption operations to be performed in either order. If signing is done first, the identity of the signer is hidden by the encryption. Plus, it is generally more convenient to store a signature with a plaintext version of a message. Furthermore, for purposes of third-party verification, if the signature is performed first, a third party need not be concerned with the symmetric key when verifying the signature.

If encryption is done first, it is possible to verify a signature without exposing the message content. This option can be useful in a context in which automatic signature verification is desired, as no private-key material is required to verify a signature. However, in this case the recipient cannot determine any relationship between the signer and the unencrypted content of the message.

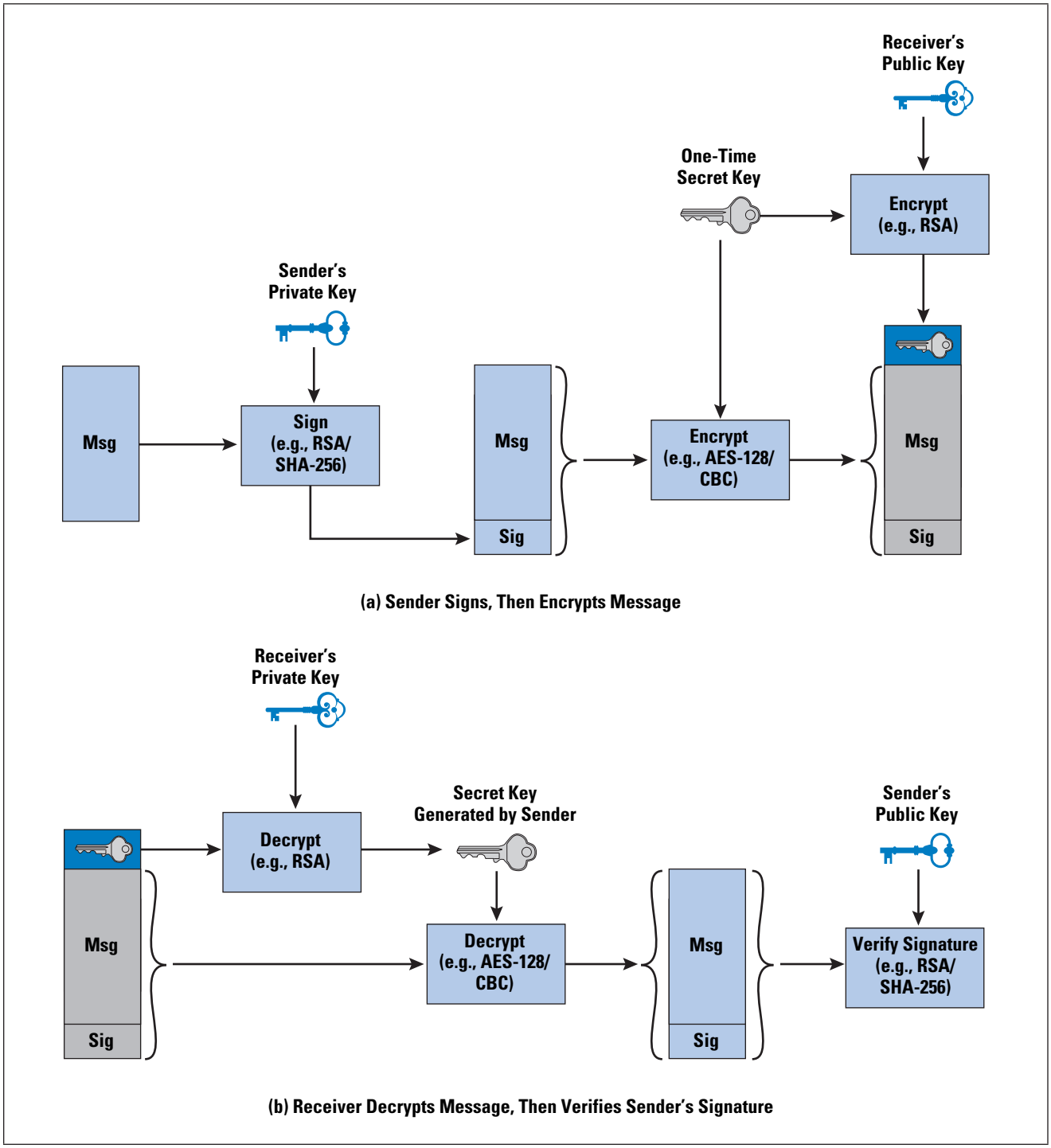
When S/MIME is used, at least part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time symmetric key). Thus, part of or the entire resulting block consists of a stream of arbitrary 8-bit octets. However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction and provide *compatibility,* S/MIME provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters, a process referred to as 7-bit encoding. The scheme typically used for this purpose is Base64 conversion. Each group of three octets of binary data is mapped into four ASCII characters.

S/MIME also offers the ability to *compress* a message. Message compression has the benefit of saving space for both e-mail transmission and file storage. Compression can be applied in any order with respect to the signing and message encryption operations.

RFC 5751 provides the following guidelines:

- Compression of binary encoded encrypted data is discouraged, since it will not yield significant compression. Base64 encrypted data could very well benefit, however.
- If a lossy compression algorithm is used with signing, you will need to compress first, then sign.

Figure 3: Simplified S/MIME Functional Flow



SP 800-177 recommends the use of certificate chain authentication against a known certificate authority. Further, SP 800-177 indicates that users who want more assurance that the public key supplied is bound to the sender's domain may use a work-in-progress DANE-S/MIME mechanism[20], in which the certificate and key can be independently retrieved from the DNS and authenticated per the DANE mechanism described subsequently.

In addition, SP 800-177 notes that MUAs typically use S/MIME private keys to decrypt the e-mail message each time it is displayed, but leave the message encrypted in the e-mail store. This mode of operation is not recommended, as it forces recipients of the encrypted e-mail to maintain their private key indefinitely. Instead, the e-mail should be decrypted prior to being stored in the mail store. The mail store, in turn, should be secured using an appropriate cryptographic technique (for example, disk encryption), extending protection to both encrypted and unencrypted e-mail.

OpenPGP

Pretty Good Privacy (PGP) was developed by Phil Zimmermann as a publicly-available freeware package to enable individuals to exchange secure e-mails without the need to rely on any institution. Efforts began early on to develop Internet standards for PGP[21], culminating in *OpenPGP*. OpenPGP[22, 23] is a proposed Internet Standard for providing authentication and confidentiality for e-mail messages. Although it is similar in purpose and functionality to S/MIME, OpenPGP uses different message and key formats and a different approach to establishing and using certificates. SP 800-177 cites many difficulties with OpenPGP, including lack of usability, scalability issues related to key distribution, and lack of authentication of key owners. Further discussion can be found in [24] and [25]. Accordingly, SP 800-177 recommends the use of only S/MIME and deprecates the use of OpenPGP.

DNS and DNSSEC

As background for the following sections, this section briefly reviews DNS and DNSSEC. The *Domain Name System* (DNS) is a directory lookup service that provides a mapping between the name of a host on the Internet and its numerical Internet address. Four elements comprise the DNS. The domain name space is a tree-structured name space to identify resources on the Internet. The DNS database is a collection of resource records organized into a distributed database; conceptually, each node and leaf in the name-space tree structure names a collection of information (for example, IP address, name server for this domain name) that is contained in *Resource Records* [RRs]). *Name Servers* are server programs that hold information about a portion of the domain-name tree structure and the associated RRs. *Resolvers* are programs that extract information from name servers in response to client requests. A typical client request is for an IP address corresponding to a given domain name.

The DNS database is divided into thousands of separately managed *zones,* which are managed by separate administrators. Using this database, DNS servers provide a name-to-address directory service for network applications that need to locate specific application servers.

Domain Name System Security Extensions (DNSSEC)[26] is used by several protocols that provide e-mail security. DNSSEC provides end-to-end protection through the use of digital signatures that are created by responding zone administrators and verified by a recipient's resolver software. In particular, DNSSEC avoids the need to trust intermediate name servers and resolvers that cache or route the DNS records originating from the responding zone administrator before they reach the source of the query. DNSSEC consists of a set of new resource record types and modifications to the existing DNS protocol.

In essence, DNSSEC is designed to protect DNS clients from accepting forged or altered DNS resource records. It protects these clients by using digital signatures to provide: (1) data origin authentication to ensure that a RR has originated from the correct source; and (2) data integrity verification to ensure that the content of a RR has not been modified. The DNS zone administrator digitally signs every *Resource Record set* (RRset) in the zone, and publishes this collection of digital signatures, along with the zone administrator's public key, in the DNS itself.

In DNSSEC, trust in the public key (for signature verification) of the source is established not by going to a third party or a chain of third parties (as in *Public-Key Infrastructure* [PKI] chaining), but by starting from a trusted zone (such as the root zone) and establishing the chain of trust down to the current source of response through successive verifications of the signature of the public key of a child by its parent. The public key of the trusted zone is called the *trust anchor.*

DANE

DNS-Based Authentication of Named Entities (DANE)[27, 28] is a protocol that provides mechanisms for domains to specify which X.509 certificates, which are commonly used for *Transport Layer Security* (TLS), should be trusted for the domain. DANE enables certificates to be bound to DNS names using DNSSEC. It is proposed in RFC 6698[29] as a way to authenticate TLS client and server entities without a *Certificate Authority* (CA).

Briefly, DANE is an alternative mechanism for securely distributing information about domain names by using DNS. DANE defines a new type of DNS record that enables a domain to sign statements specifying which entities are authorized to represent it. Applications can use these records either to augment the existing system of CAs or to create a new chain of trust, rooted in the DNS.

The rationale for DANE is the vulnerability of the use of CAs in a global *Public-Key Infrastructure* (PKI) system. Every browser developer and operating system supplier maintains a list of CA root certificates as trust anchors. These certificates are called the *root certificates* of the software and are stored in its root certificate store. The PKI scheme allows a certificate recipient to trace a certificate back to the root. So long as the root certificate remains trustworthy and the authentication concludes successfully, the client can proceed with the connection. However, if any of the hundreds of CAs operating on the Internet is compromised, the effects can be widespread. The attacker can obtain the private key of the CA, be issued certificates under a false name, or introduce new bogus root certificates into a root certificate store. There is no limitation of scope for the global PKI, and a compromise of a single CA damages the integrity of the entire PKI system. In addition, some CAs have engaged in poor security practices. For example, some CAs have issued wildcard certificates that allow the holder to issue sub-certificates for any domain or entity, anywhere in the world.

The purpose of DANE is to replace reliance on the security of the CA system with reliance on the security provided by DNSSEC. This protocol is well expressed in RFC 6698:

> "DNS-Based Authentication of Named Entities (DANE) offers the option to use the DNSSEC infrastructure to store and sign keys and certificates that are used by TLS. DANE is envisioned as a preferable basis for binding public keys to DNS names, because the entities that vouch for the binding of public key data to DNS names are the same entities responsible for managing the DNS names in question. While the resulting system still has residual security vulnerabilities, it restricts the scope of assertions that can be made by any entity, consistent with the naming scope imposed by the DNS hierarchy. As a result, DANE embodies the security "principle of least privilege" that is lacking in the current public CA model."

DANE defines a new DNS record type, TLSA, which can be used for a secure method of authenticating *Secure Sockets Layer/Transport Layer Security* (SSL/TLS) certificates. The TLSA provides for:

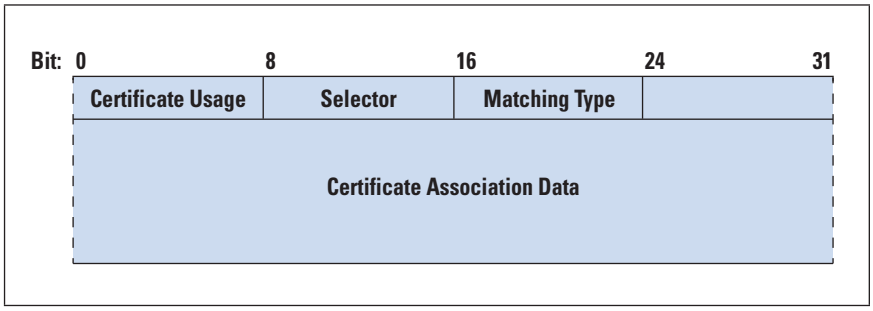
- Specifying constraints on which CA can vouch for a certificate, or which specific PKI end-entity certificate is valid.
- Specifying that a service certificate or a CA can be directly authenticated in the DNS itself.

The TLSA RR enables certificate issue and delivery to be tied to a given domain. A server domain owner creates a TLSA resource record that identifies the certificate and its public key. When a client receives an X.509 certificate in the TLS negotiation, it looks up the TLSA RR for that domain and matches the TLSA data against the certificate as part of the client's certificate validation procedure.

Figure 4 shows the format of a TLSA RR as it is transmitted to a requesting entity. It contains four fields. The *Certificate Usage* field defines four different usage models, to accommodate users who require different forms of authentication. The usage models follow:

- *PKIX-TA (CA constraint):* Specifies which CA should be trusted to authenticate the certificate for the service. This usage model limits which CA can be used to issue certificates for a given service on a host. The server certificate chain must pass PKIX validation that terminates with a trusted root certificate stored in the client.
- *PKIX-EE (service certificate constraint):* Defines which specific end-entity service certificate should be trusted for the service. This usage model limits which end-entity certificate can be used by a given service on a host. The server certificate chain must pass PKIX validation that terminates with a trusted root certificate stored in the client.
- *DANE-TA (trust anchor assertion):* Specifies a domain-operated CA to be used as a trust anchor. This usage model allows a domain-name administrator to specify a new trust anchor—for example, if the domain issues its own certificates under its own CA that is not expected to be in the end users' collection of trust anchors. The server certificate chain is self-issued and does not need to verify against a trusted root stored in the client.
- *DANE-EE (domain-issued certificate):* Specifies a domain-operated CA to be used as a trust anchor. This certificate usage allows for a domain-name administrator to issue certificates for a domain without involving a third-party CA. The server certificate chain is self-issued and does not need to verify against a trusted root stored in the client.

Figure 4: TSLA RR Transmission Format



The first two usage models are designed to coexist with and strengthen the public CA system. The final two usage models operate without the use of public CAs.

The *Selector* field indicates whether the full certificate or just the value of the public key will be matched. The match is made between the certificate presented in TLS negotiation and the certificate in the TLSA RR. The *Matching Type* field indicates how the match of the certificate is made. The options are exact match, SHA-256 hash match, or SHA-512 hash match. The *Certificate Association Data* is the raw certificate data in hex format.

DANE can be used in conjunction with SMTP over TLS, as provided by STARTTLS, to more fully secure e-mail delivery. DANE can authenticate the certificate of the SMTP submission server that the user's mail client (MUA) communicates with. It can also authenticate the TLS connections between SMTP servers (MTAs). The use of DANE with SMTP is documented in RFC 7672[30].

As discussed previously, SMTP can use the STARTTLS extension to run SMTP over TLS, so that the entire e-mail message plus SMTP envelope are encrypted. This option is used if both sides support STARTTLS. Even when TLS is used to provide confidentiality, it is vulnerable to attack in the following ways:

- Attackers can strip away the TLS capability advertisement and downgrade the connection to not use TLS.
- TLS connections are often unauthenticated (for example, the use of self-signed certificates as well as mismatched certificates is common).

DANE can address both these vulnerabilities. A domain can use the presence of the TLSA RR as an indicator that encryption must be performed, thus preventing malicious downgrade. A domain can authenticate the certificate used in the TLS connection setup using a DNSSEC-signed TLSA RR.

DNSSEC can be used in conjunction with S/MIME to more fully secure e-mail delivery, in a manner similar to the DANE functionality. This use is documented in an Internet Draft[21], which proposes a new SMIMEA DNS RR. The purpose of the SMIMEA RR is to associate certificates with DNS domain names.

S/MIME messages often contain certificates that can assist in authenticating the message sender and can be used in encrypting messages sent in reply. This feature requires that the receiving MUA validate the certificate associated with the purported sender. SMIMEA RRs can provide a secure means of doing this validation.

In essence, the SMIMEA RR will have the same format and content as the TLSA RR, with the same functionality. The difference is that it is geared to the needs of MUAs in dealing with domain names as specified in e-mail addresses in the message body, rather than domain names specified in the outer SMTP envelope.

Sender Policy Framework

Sender Policy Framework (SPF) is the standardized way for a sending domain to specify a list of MTAs that are authorized to send on behalf of the domain. The problem that SPF addresses is the following: with the current e-mail infrastructure, any host can use any domain name for each of the various identifiers in the mail header, not just the domain name where the host is located.

Two major drawbacks of this freedom follow:

- It is a major obstacle to reducing *Unsolicited Bulk E-mail* (UBE), also known as *spam*. It makes it difficult for mail handlers to filter out e-mails on the basis of known UBE sources.
- *Administrative Management Domains* (ADMDs) are understandably concerned about the ease with which other entities can use their domain names, often with malicious intent.

However, a basic limitation of SPF is that it forces mail to follow a specific path and breaks when legitimate mail deviates from this path, such as a message that goes through a mailing list.

RFC 7208 defines the SPF[31]. It provides a protocol by which ADMDs can authorize hosts to use their domain names in the **`MAIL FROM`** or **`HELO`** identities. (It is worth noting that this domain name is the return address for error messages, rather than being required to be the same as the author's address.) Compliant ADMDs publish SPF records in the DNS specifying which hosts are permitted to use their names, and compliant mail receivers use the published SPF records to test the authorization of sending MTAs using a given **`HELO`** or **`MAIL FROM`** identity during a mail transaction.

SPF works by checking a neighboring, upstream client MTA IP address against the policy encoded in any SPF record found at the sending domain. The sending domain is the domain used in the SMTP connection, not the domain indicated in the Author From field in the message header as displayed in the MUA. Thus SPF checks can be applied before the message content is received from the sender.

Figure 5 on the following page is an example in which SPF would come into play. Assume that the sender's IP address is **`192.168.0.1`**. The message arrives from the MTA with domain **`mta.example.net`**. The sender uses the envelope **`MAIL FROM`** tag of **`alice@example.org`**, indicating that the message originates in the **`example.org`** domain. But the message header specifies **`alice.sender@example.net`**. The receiver uses SPF to query for the SPF RR that corresponds to **`example.org`** to check if the IP address **`192.168.0.1`** is listed as a valid sender, and then takes appropriate action based on the results of checking the RR.

A sending domain needs to identify all the senders for a given domain and add that information into the DNS as a separate resource record. Next, the sending domain encodes the appropriate policy for each sender using the SPF syntax. The encoding is done in a TXT DNS resource record as a list of mechanisms and modifiers. Mechanisms are used to define an IP address or range of addresses to be matched, and modifiers indicate the policy for a given match. The SPF syntax is fairly complex and can express complex relationships between senders. For more details, see RFC 7208.

Figure 5: Example in Which SMTP Envelope Header Does Not Match Message Header

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: HELO mta.example.net
S: 250 OK
C: MAIL FROM:<alice@example.org>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: To: bob@foo.com
C: From: alice.sender@example.net
C: Date: Today
C: Subject: Meeting Today
. . .
```

If SPF is implemented at a receiver, the SPF entity uses the SMTP envelope **`MAIL FROM:`** address domain and the IP address of the sender to query an SPF TXT RR. The SPF checks can be started before the body of the e-mail message is received, possibly resulting in blocking the transmission of the e-mail content. Alternatively, the entire message can be absorbed and buffered until all the checks are finished. In either case, checks must be completed before the mail message is sent to the end user's inbox.

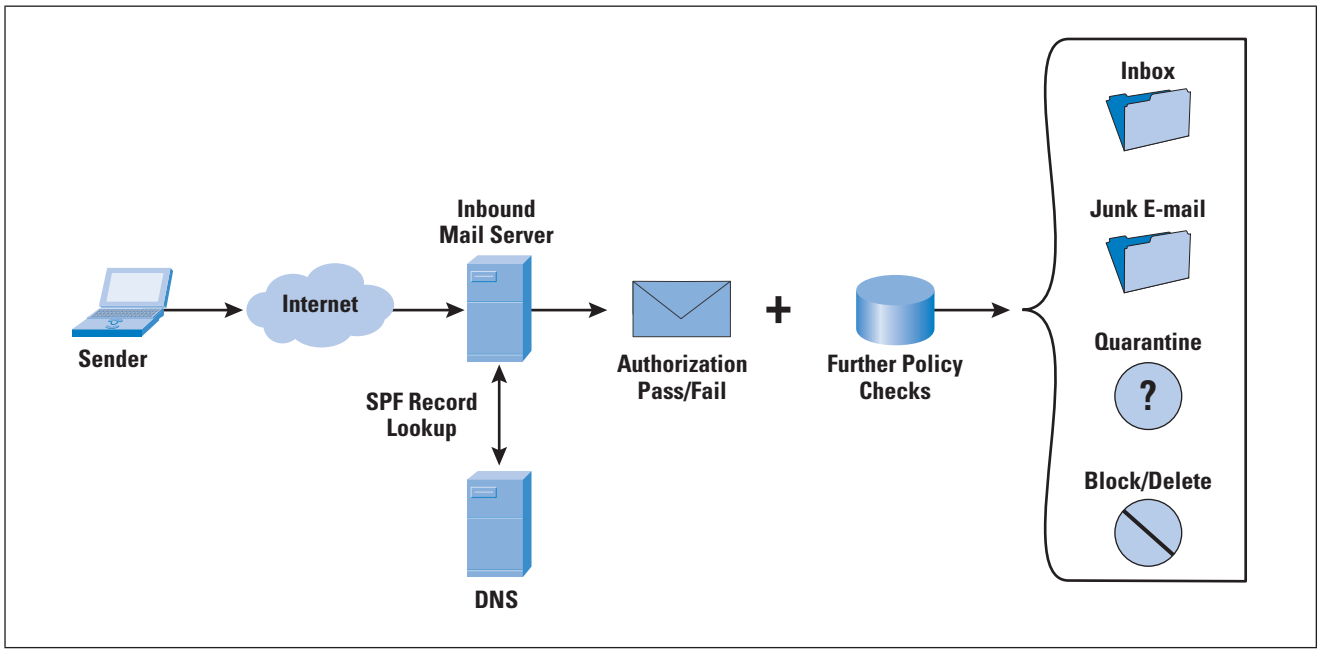
The checking involves the following rules:

1. If no SPF TXT RR is returned, the default behavior is to accept the message.
2. If the SPF TXT RR has formatting errors, the default behavior is to accept the message.
3. Otherwise the mechanisms and modifiers in the RR are used to determine disposition of the e-mail message.

With respect to SPF alone, to say in step 1, preceding, that the default behavior is to accept the message is correct. However, it should be noted that SPF is usually working within a mixture of anti-abuse tools and the aggregate filtering engine typically does not accept a message based on the results of only one of its tools, such as SPF.

Figure 6 illustrates SPF operation. As of 2016, more than 27% of all Internet domains implement SPF[32].

Figure 6: Sender Policy Framework Operation



DKIM

DomainKeys Identified Mail (DKIM) permits a person, role, or organization that owns the signing domain to claim some responsibility for a message by associating the domain with the message[33]. The domain can be an author's organization, an operational relay, or one of their agents. DKIM separates the question of the identity of the signer of the message from the purported author of the message. Assertion of responsibility is validated through a cryptographic signature and by querying the signer's domain directly to retrieve the appropriate public key.

The qualifier *some* in the first sentence of the preceding paragraph is important. In particular, the text directly "covered" by the signature is not vetted for authenticity.

Message recipients (or agents acting in their behalf) can verify the signature by querying the signer's domain directly to retrieve the appropriate public key and thereby can confirm that the message was attested to by a party in possession of the private key for the signing domain. DKIM is an Internet Standard defined in RFC 6376[34]. DKIM has been widely adopted by a range of e-mail providers, including corporations, government agencies, Gmail, Yahoo, and many *Internet Service Providers* (ISPs). As of 2016, an estimated 40% of Internet sites deploy DKIM[35].

An *Administrative Unit* (AU) is that portion of the path of an e-mail message that is under a single administration. DKIM focuses primarily on attackers located outside of the AUs of the claimed originator and the recipients, indirectly, by creating a verifiable signature of valid mail from the administrative unit.

It is with these external AUs that the trust relationships required for authenticated message submission may not exist and do not scale adequately to be practical. Conversely, within these AUs, there are other mechanisms (such as authenticated message submission) that are easier to deploy and more likely to be used than DKIM. External bad actors are usually attempting to exploit the "any-to-any" nature of e-mail that motivates most recipient MTAs to accept messages from anywhere for delivery to their local domain. They may generate messages without signatures, with incorrect signatures, or with correct signatures from domains with little traceability. They may also pose as mailing lists, greeting cards, or other agents that legitimately send or resend messages on behalf of others.

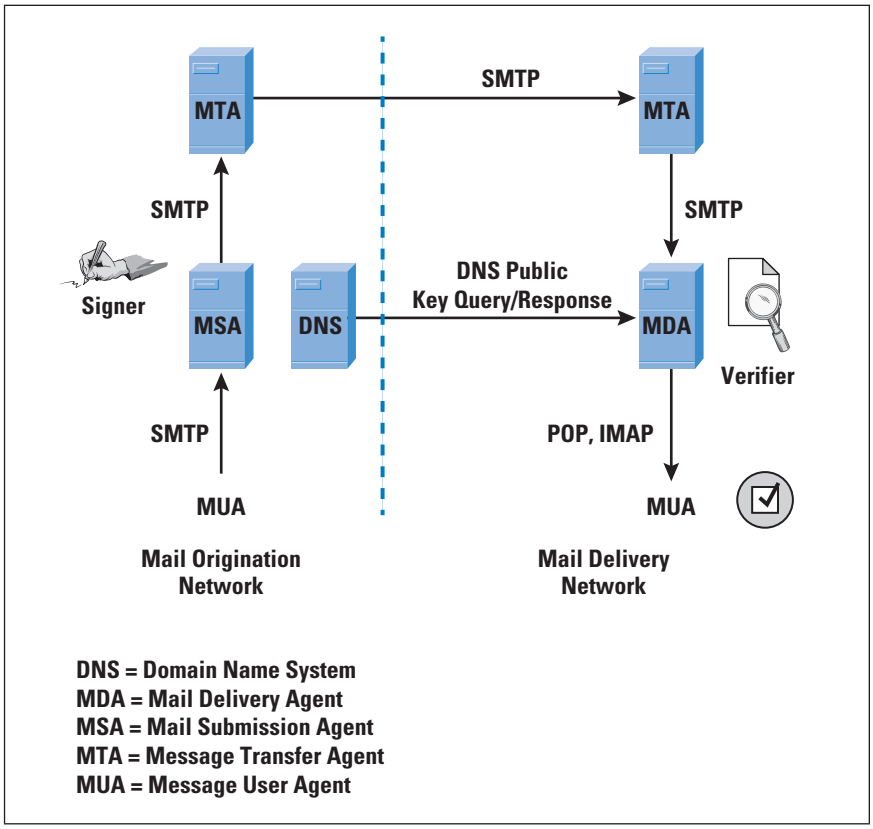
DKIM is designed to provide an e-mail authentication technique that is transparent to the end user. In essence, a user's e-mail message is signed by a private key of the administrative domain from which the e-mail originates. The signature covers none, some, or all of the content of the message and some of the e-mail message headers.

Note that the signature is not validating any of what is signed, as digital signatures usually do. Rather, the choice of what to cover is meant as a means of gluing the **`d=domain name`** to the overall message in a way that is difficult to spoof. At the receiving end, the Message Delivery Agent can access the corresponding public key via a DNS and verify the signature, thus authenticating that the message comes from the claimed administrative domain. Thus, DKIM allows an enterprise to vouch for an e-mail message sent from a domain it does not control. This approach differs from that of S/MIME, which uses the originator's private key to sign the content of the message. The motivation for DKIM is based on the following reasoning:

- S/MIME depends on both the sending and receiving users employing S/MIME. For almost all users, the bulk of incoming mail does not use S/MIME, and the bulk of the mail the user wants to send is to recipients not using S/MIME.
- S/MIME signs only the message content. Thus, RFC 5322[36] header information concerning origin can be compromised.
- DKIM is not implemented in client programs (MUAs) and is therefore transparent to the user; the user doesn't need to take any action.
- DKIM can be configured to apply to all mail from cooperating domains.
- DKIM allows good senders to prove that they did send a particular message and to prevent forgers from forging the DKIM signature.

Figure 7 is a simple example of the operation of DKIM. We begin with a message generated by a user and transmitted into the *Message Handling Service* (MHS) to an MSA that is within the user's administrative domain. An e-mail message is generated by an e-mail client program.

Figure 7: Simple Example of DKIM Deployment



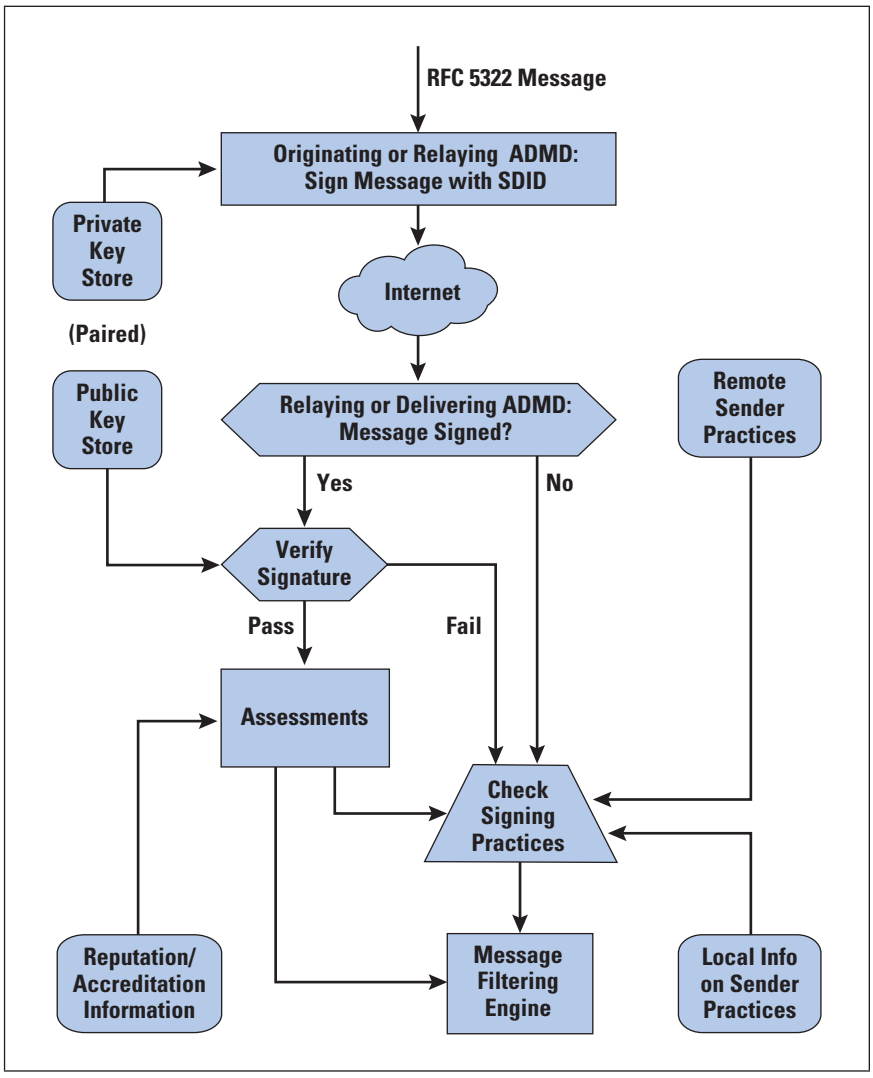
The content of the message, plus selected RFC 5322 headers, is signed by the e-mail provider using the provider's private key. The signer is associated with a domain, which could be a corporate local network, an ISP, or a public e-mail facility such as Gmail. The signed message then passes through the Internet via a sequence of MTAs. At the destination, the MDA retrieves the public key for the incoming signature and verifies the signature before passing the message on to the destination e-mail client. The default signing algorithm is RSA with SHA-256. RSA with SHA-1 also may be used.

Figure 8 on the following page, from RFC 5585[37], provides a more detailed look at the elements of DKIM operation. Basic message processing is divided between a signing *Administrative Management Domain* (ADMD) and a verifying ADMD. At its simplest, this processing is between the originating ADMD and the delivering ADMD, but it can involve other ADMDs in the handling path.

Signing is performed by an authorized module within the signing ADMD and uses private information from a Key Store. Within the originating ADMD, this signing might be performed by the MUA, MSA, or an MTA. Verifying is performed by an authorized module within the verifying ADMD. Within a delivering ADMD, verifying might be performed by an MTA, MDA, or MUA. The module verifies the signature or determines whether a particular signature was required.

Verifying the signature uses public information from the Key Store. If the signature passes, reputation information is used to assess the signer and that information is passed to the message filtering system.

Figure 8: DKIM Functional Flow



If the signature fails or there is no signature using the author's domain, information about signing practices related to the author can be retrieved remotely and/or locally, and that information is passed to the message filtering system. For example, it the sender (for example, Gmail) uses DKIM but no DKIM signature is present, then the message may be considered fraudulent.

The signature is inserted into the RFC 5322 message as an additional header field, starting with the keyword Dkim-Signature. You can view examples from your own incoming mail by using the "View Long Headers (or similar wording) option for an incoming message.

Before a message is signed, a process known as *canonicalization* is performed on both the header and body of the RFC 5322 message. Canonicalization is necessary to deal with the possibility of minor changes in the message made en route, including character encoding, treatment of trailing white space in message lines, and the "folding" and "unfolding" of header lines. The intent of canonicalization is to make a minimal transformation of the message (for the purpose of signing; the message itself is not changed, so the canonicalization must be performed again by the verifier) that will give it its best chance of producing the same canonical value at the receiving end. DKIM defines two header canonicalization algorithms ("simple" and "relaxed") and two for the body (with the same names). The simple algorithm tolerates almost no modification, while the relaxed tolerates common modifications.

DMARC

Domain-Based Message Authentication, Reporting, and Conformance (DMARC), defined in RFC 7489[38], allows e-mail senders to specify policy on how their mail should be handled, the types of reports that receivers can send back, and the frequency of those reports.

DMARC works with SPF and DKIM. SPF enables senders to advise receivers, via DNS, whether mail purporting to come from the sender is valid, and whether it should be delivered, flagged, or discarded. However, neither SPF nor DKIM includes a mechanism to tell receivers if SPF or DKIM is in use, nor do they have a feedback mechanism to inform senders of the effectiveness of the anti-spam techniques. For example, if a message arrives at a receiver without a DKIM signature, DKIM provides no mechanism to allow the receiver to learn if the message is authentic but was sent from a sender that did not implement DKIM, or if the message is a spoof. In essence, DMARC addresses these issues by indicating whether SPF and/or DKIM will be used, what a receiver should do when they aren't, and how receivers should report aggregate results for the domain.

DKIM, SPF, and DMARC authenticate various aspects of an individual message. DKIM authenticates the domain that affixed a signature to the message. SPF focuses on the SMTP envelope, defined in RFC 5321[39]. It can authenticate either the domain that appears in the **`MAIL FROM`** portion of the SMTP envelope or the **`HELO`** domain, or both. These domains may be different, and they are typically not visible to the end user.

DMARC authentication deals with the From domain in the message header, as defined in RFC 5322. This field is used as the central identity of the DMARC mechanism because it is a required message header field and therefore guaranteed to be present in compliant messages, and most MUAs represent the RFC 5322 From field as the originator of the message and render some or all of this content of the header field to end users. The e-mail address in this field is the one used by end users to identify the source of the message and therefore is a prime target for abuse.

DMARC requires that the From address match (be aligned with) an Authenticated Identifier from DKIM or SPF. In the case of DKIM, the match is made between the DKIM signing domain and the From domain. In the case of SPF, the match is between the SPF-authenticated domain and the From domain.

A mail sender that uses DMARC must also use SPF or DKIM, or both. The sender posts a DMARC policy in the DNS that advises receivers on how to treat messages that purport to originate from the sender's domain. The policy is in the form of a DNS TXT resource record associated with the sender's domain name. The sender also needs to establish e-mail addresses to receive aggregate and forensic reports. Because these e-mail addresses are published unencrypted in the DNS TXT RR, they are easily discovered, leaving the poster subject to unsolicited bulk e-mail. Thus, the poster of the DNS TXT RR needs to employ some kind of abuse countermeasures.

Similar to SPF and DKIM, the DMARC policy in the TXT RR is encoded in a series of tag=value pairs separated by semicolons. Once the DMARC RR is posted, messages from the sender are typically processed as follows:

1. The domain owner constructs an SPF policy and publishes it in its DNS database. The domain owner also configures its system for DKIM signing. Finally, the domain owner publishes via the DNS a DMARC message-handling policy.
2. The author generates a message and hands the message to the domain owner's designated mail submission service.
3. The submission service passes relevant details to the DKIM signing module in order to generate a DKIM signature to be applied to the message.
4. The submission service relays the now-signed message to its designated transport service for routing to its intended recipient(s).

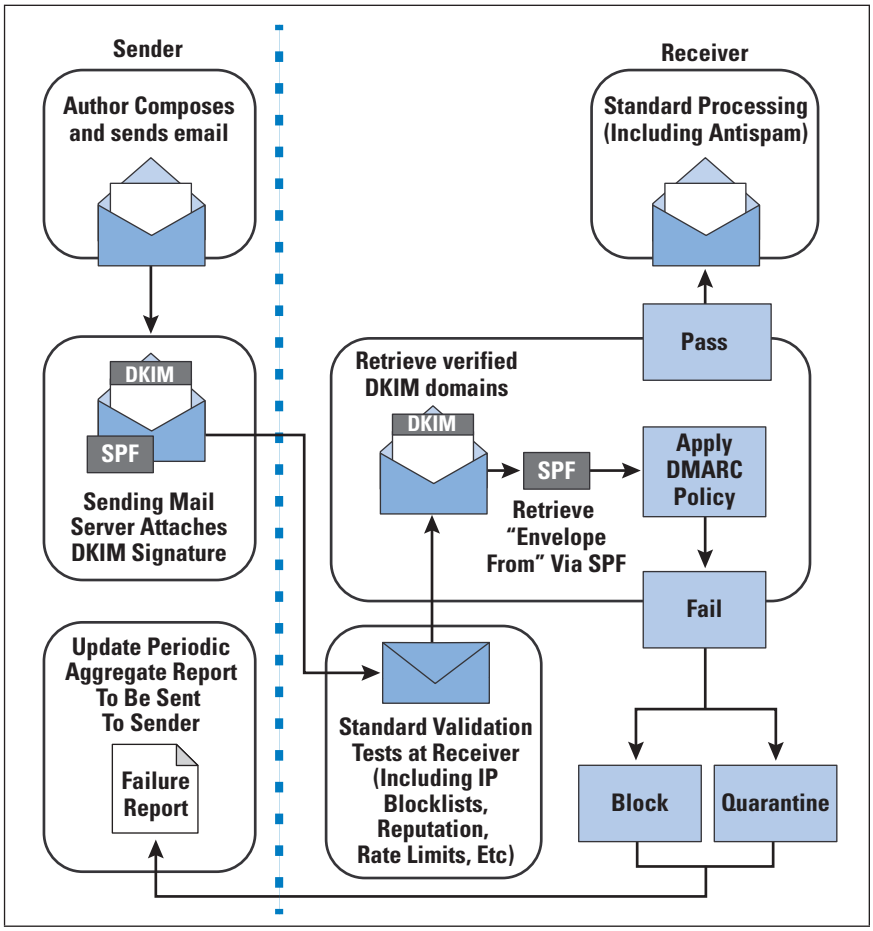
A message generated on the sender side may pass through other relays but eventually arrives at a receiver's transport service. The typical processing order for DMARC on the receiving side follows:

1. The receiver performs standard validation tests, such as checking against IP blocklists and domain reputation lists, as well as enforcing rate limits from a particular source.
2. The receiver extracts the RFC 5322 From address from the message. This address must be a single, valid address or else the mail is refused as an error.
3. The receiver queries for the DMARC DNS record based on the sending domain. If none exists, DMARC processing is terminated.
4. The receiver performs DKIM signature checks. If more than one DKIM signature exists in the message, one must verify.

5. The receiver queries for the SPF record of the sending domain and performs SPF validation checks.
6. The receiver conducts Identifier Alignment checks between the RFC 5321 From and the results of the SPF and DKIM records (if present).
7. The results of these steps are passed to the DMARC module along with the Author's domain. The DMARC module attempts to retrieve a policy from the DNS for that domain. If none is found, the DMARC module determines the organizational domain and repeats the attempt to retrieve a policy from the DNS.
8. If a policy is found, it is combined with the Author's domain and the SPF and DKIM results to produce a DMARC policy result (a "pass" or "fail") and can optionally cause one of two kinds of reports to be generated.
9. Recipient transport service either delivers the message to the recipient inbox or takes other local policy action based on the DMARC result.
10. When requested, Recipient transport service collects data from the message delivery session to be used in providing feedback.

Figure 9, based on one at `DMARC.org`, summarizes the sending and receiving functional flow.

Figure 9: DMARC Functional Flow



DMARC reporting provides the senders feedback on their SPF, DKIM, Identifier Alignment, and message disposition policies, which enables the sender to make these policies more effective. Two types of reports are sent: *Aggregate Reports* and *Forensic Reports*.

Aggregate Reports are sent by receivers periodically and include aggregate figures for successful and unsuccessful message authentications, including:

- The sender's DMARC policy for that interval
- The message disposition by the receiver (that is, delivered, quarantined, rejected)
- SPF result for a given SPF identifier
- DKIM result for a given DKIM identifier
- Whether identifiers are in alignment or not
- Results classified by sender subdomain
- The sending and receiving domain pair
- The policy applied, and whether it is different from the policy requested
- The number of successful authentications
- Totals for all messages received

This information enables the sender to identify gaps in e-mail infrastructure and policy. SP 800-177 recommends that a sending domain begin by setting a DMARC policy of **`p=none`**, so that the ultimate disposition of a message that fails some check is determined by the receiver's local policy. As DMARC aggregate reports are collected, the sender will have a quantitatively better assessment of the extent to which the sender's e-mail is authenticated by outside receivers, and will be able to set a policy of **`p=reject`**, indicating that any message that fails the SPF, DKIM, and alignment checks really should be rejected. From their own traffic analysis, receivers can determine whether a sender's **`p=reject`** policy is sufficiently trustworthy to act on.

A *Forensic Report* helps senders refine the component SPF and DKIM mechanisms as well as alerting them that their domain is being used as part of a phishing/spam campaign. Forensic reports are similar in format to aggregation reports, with these changes:

- Receivers include as much of the message and message header as is reasonable to allow the domain to investigate the failure. Add an *Identity-Alignment* field, with DKIM and SPF DMARC-method fields as appropriate.
- Optionally add a *Delivery-Result* field. Add DKIM Domain, DKIM Identity, and DKIM selector fields, if the message was DKIM signed. Optionally also add DKIM Canonical header and body fields.
- Add an additional DMARC authentication failure type, for use when some authentication mechanisms fail to produce aligned identifiers.

Since its introduction, DMARC has seen rapid acceptance. Thousands of companies use it to prevent billions of messages fraudulently using their Internet domains from reaching inboxes, thereby protecting their customers and employees from phishing and other abuse. Recently, two of the largest mailbox providers in the world—Google and Yahoo—have announced that they are extending that protection to cover more of their Internet domains[40].

Summary

The IETF has developed a suite of protocols that provide comprehensive Internet e-mail security. Many of these protocols have been widely deployed, and the entire suite is recommended by NIST.

Acknowledgment

The author would like to express his gratitude to the reviewer for the many detailed and helpful comments.

References

[1] National Institute of Standards and Technology, "Trustworthy Email," NIST Special Publication 800-177, September 2016.

[2] Dave Crocker, "Internet Mail Architecture," RFC 5598, July 2009.

[3] Crypto Portal, "Cryptology with Cryptool: Practical Introduction to Cryptography and Cryptanalysis," August 2010. **https://www.cryptool.org/images/ct1/presentations/CrypToolPresentation-en.pdf**

[4] National Institute of Standards and Technology, "Introduction to Public Key Technology and the Federal PKI Infrastructure," NIST Special Publication 800-32, February 2001.

[5] Paul Hoffman, "SMTP Service Extension for Secure SMTP over Transport Layer Security," RFC 3207, February 2002.

[6] ZDNet, "Google, Microsoft, Yahoo: We want to stop e-mail snooping by fixing these encryption flaws," March 21, 2016.
http://www.zdnet.com/article/google-microsoft-yahoo-we-want-to-stop-e-mail-snooping-by-fixing-these-encryption-flaws/#!

[7] Facebook, "The Current State of SMTP STARTTLS Deployment," May 13, 2014.
https://www.facebook.com/notes/protect-the-graph/the-current-state-of-smtp-starttls-deployment/1453015901605223/

[8] William Stallings, "SSL: Foundation for Web Security," *The Internet Protocol Journal,* Volume 1, No. 1, June 1998.

[9] Andrea Peterson, "Facebook's security chief on the Snowden effect, the Messenger app backlash and staying optimistic," *The Washington Post,* August 12, 2014.

[10] David Cohen, "Facebook: 95% of Notification Emails Encrypted Thanks to Providers' STARTTLS Deployment," *AdWeek,* August 19, 2014.

[11] Marshall Rose and David Strom, "Secure E-Mail: Problems, Standards, and Prospects," *The Internet Protocol Journal,* Volume 2, No. 1, March 1999.

[12] Sean Turner and Blake Ramsdell, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling," RFC 5750, January 2010.

[13] Sean Turner and Blake Ramsdell, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification," RFC 5751, January 2010.

[14] Paul Hoffman, "Examples of S/MIME Messages," RFC 4134, July 2005.

[15] Paul Hoffman, "Enhanced Security Services for S/MIME," RFC 2634, June 1999.

[16] Russ Housley, "Cryptographic Message Syntax (CMS)," RFC 5652, September 2009.

[17] Russ Housley, "Cryptographic Message Syntax (CMS) Algorithms," RFC 3370, August 2002.

[18] Jim Schaad and Sean Turner, "Multiple Signatures in S/MIME," RFC 5752, January 2010.

[19] Sandy Murphy, Jim Galvin, Steve Crocker, and Ned Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted," RFC 1847, October 1995.

[20] Paul Hoffman and Jakob Schlyter, "Using Secure DNS to Associate Certificates with Domain Names for S/MIME," Internet Draft, work in progress, **draft-ietf-dane-smime-10**, February 24, 2016.

[21] Philip Zimmermann, Derek Atkins, and William Stallings, "PGP Message Exchange Formats," RFC 1991, August 1996.

[22] Hal Finney, Lutz Donnerhacke, Jon Callas, Rodney Thayer, and David Shaw, "OpenPGP Message Format," RFC 4880, November 2007.

[23] Dave Del Torto, Michael Elkins, Raph Levien, and Thomas Roessler, "MIME Security with OpenPGP," RFC 3156, August 2001.

[24] A. Whitten and J. Tygar, "Why Johnny can't encrypt: a usability evaluation of PGP 5.0," *Proceedings of the 8th conference on USENIX Security Symposium* - Volume 8 (SSYM'99), 1999.

[25] Matthew Green, "What's the Matter with PGP?" Cryptography Engineering Blog, August 13, 2014.
http://blog.cryptographyengineering.com/2014/08/whats-matter-with-pgp.html

[26] Miek Gieben, "DNSSEC: The Protocol, Deployment, and a Bit of Development," *The Internet Protocol Journal,* Volume 7, No. 2, June 2004.

[27] ichard Barnes, "Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE)," RFC 6394, October 2011.

[28] Richard Barnes, "Let the Names Speak for Themselves: Improving Domain Name Authentication with DNSSEC and DANE," *The Internet Protocol Journal,* Volume 15, No. 1, March 2012.

[29] Jakob Schlyter and Paul Hoffman, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," RFC 6698, August 2012.

[30] Viktor Dukhovni and Wesley Hardaker, "SMTP Security via Opportunistic DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS)," RFC 7672, October 2015.

[31] Scott Kitterman, "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1," RFC 7208, April 2014.

[32] "SPF-all Domain Survey," **http://spf-all.com/stats.html**

[33] Barry Leiba and Jim Fenton, "DomainKeys Identified Mail (DKIM): Using Digital Signatures for Domain Verification," CEAS 2007—*Fourth Conference on E-mail and Anti-Spam,* August 2–3, 2007.

[34] Murray Kucherawy, Dave Crocker, and Tony Hansen, "DomainKeys Identified Mail (DKIM) Signatures," RFC 6376, September 2011.

[35] "Global DKIM Deployment," **https://eggert.org/meter/dkim**

[36] Peter W. Resnick, "Internet Message Format," RFC 5322, October 2008.

[37] Tony Hansen, Dave Crocker, and Phillip Hallam-Baker, "DomainKeys Identified Mail (DKIM) Service Overview," RFC 5585, July 2009.

[38] Murray Kucherawy and Elizabeth Zwicky, "Domain-based Message Authentication, Reporting, and Conformance (DMARC)," RFC 7489, March 2015.

[39] John C. Klensin, "Simple Mail Transfer Protocol," RFC 5321, October 2008.

[40] Dmarc.org, "Global Mailbox Providers Deploying DMARC to Protect Users," Dmarc Press Release, October 19, 2015.

WILLIAM STALLINGS is an independent consultant and author of numerous books on security, computer networking, and computer architecture. His latest book is *Cryptography and Network Security* (Pearson, 2016). He maintains a computer science resource site for computer science students and professionals at **ComputerScienceStudent.com** and is on the editorial board of *Cryptologia*. He has a Ph.D. in computer science from M.I.T. He can be reached at **ws@shore.net**

Cloudy-Eyed: Complexity and Reality with Software-Defined Networks

by Russ White and Shawn Zandi, LinkedIn

Software-Defined Networks (SDN) are promoted as a way to eliminate the complexity of distributed control planes, increase network responsiveness to specific applications and business requirements, and reduce operational and equipment cost. If this description sounds like the classic "too good to be true" situation, that's because it might just be. Just like you can't build a database that has ideal consistency, accessibility, and partionability, you can't build a cheap network with optimal routing and minimal control-plane state. It's just a reality of the complexity built into the physical shape of the universe that everything has a tradeoff—*cheap, fast, and high quality, choose two.*

When we reach the top of the SDN hype cycle, what will our options be? Perhaps the best place to start in answering this question is by considering why the "big promise" of SDN hasn't been really successful in the real world.

Defining SDN: Then and Now

To really understand the hype and promise of SDNs, it's important to go back to the beginning and consider what the original promise really was. There were originally three crucial elements to the SDN story.

First, SDNs were supposed to remove the intelligence from distributed control planes, replacing them with the centralized calculation of network paths in a controller. While an individual autonomous router has only a localized view of network conditions, a centralized controller can gain a more global view. A global view would allow the controller to more efficiently manage and direct traffic through the network in a way that improves both the efficiency of the network and the performance of applications running across the network.

Second, SDNs were supposed to provide a much more granular level of control—down to the flow level. This added level of control would enable much better policy control in various ways, including the discovery and direction of elephant flows, quality of service on a per-application/per-user basis, and other options.

Third, SDN would enable the network to be programmable, thereby reducing operational costs, enabling a more lean/agile view of the network, and allowing applications to interact directly with the network.

The definition of SDN has changed over the years, broadening so that it now includes just about any network technology that allows programmatic access to information about and control over the network.

An SDN, in more recent terms, seems to include everything from the ability of an application to schedule bandwidth (which is a rather more complicated problem than it seems) to gaining better telemetry data. The centralized controller, flow-based forwarding, and commoditization of hardware are still in scope, but they appear to be mixed in with a much more limited view of the "core components" of the SDN message. Why has the concept of the SDN changed across time?

It's possible to argue that this definitional change is just a matter of the marketing departments at a wide array of vendors grabbing hold of the term, but there seems to be something deeper here. Perhaps the "something deeper" is the original ideals have proven more difficult to achieve than were first thought. A short overview of the challenges of deploying the original SDN ideal might be useful in understanding the historical flow of these changes. Three larger areas are considered in the following sections: centralizing the calculation of network paths, flow-based forwarding, and network programmability.

Centralizing the Calculation of Network Paths

Distributed control planes, such as *Intermediate System-to-Intermediate System* (IS-IS) and *Border Gateway Protocol* (BGP), are often (rightly) seen as one of the most complex components of a network. In fact, entire networks are designed around the operation of these routing protocols, including the consideration of topics like:

- Splitting up failure domains through information hiding
- Managing complex policies through communities, tags, and metrics
- Choosing topologies based on fast convergence characteristics
- The interaction of multiple distributed control planes running on a single network

Further, in order to support the complex processing and data handling of distributed control planes, network devices are typically large, expensive devices, with fast processors and large memory pools. In particular, as the need for policy-driven path selection (which generally means choosing a path through the network that is less optimal than the shortest path from a metrics perspective, but more optimal from a network usage or quality-of-service perspective) increased, the processing power and memory requirements of individual routers ramped up.

If distributed control planes could be eliminated and replaced by a controller (or set of controllers), the complexity of each forwarding device could be reduced dramatically, because the jobs of discovering the local topology and calculating the best path per destination would be offloaded from the individual boxes, and pushed onto the controller. By removing this processing from the routers, small, cheap, lightweight forwarding-only devices could be used instead of the traditional router.

Hence the world could move to *white-box* devices that would be available off the shelf and require little configuration.

Complexity, however, is not so easy to slay. The centralized controller approach presents numerous problems that will, most likely, forever limit it in scale and scope to something smaller than what was originally envisioned, such as two or three controllers providing forwarding information for tens of thousands of switches running at scale. Some of these problems include the relationship between centralized computation and reactive control planes, remote reactions to local topology and reachability changes, and what can fairly be described as the halo effect around software engineering.

Centralized Control and Reactive Forwarding

Distributed control planes, such as IS-IS, are proactive in their discovery of topology and reachability. Before the first packet is transmitted across the network, the routing protocol must discover a set of loop-free paths that can reach every destination in the network. Since this discovery and calculation process typically involves flooding, processing, and managing a lot of information, distributed control planes often rely on information hiding through aggregation to manage the amount and speed of state being carried in the protocol. For instance, in IS-IS intermediate systems in the level 2 flooding domain don't have any information about the topology of the outlying level 1 flooding domains. In a similar way, intermediate systems in a level 1 flooding domain know only about the topology within the flooding domain and which intermediate systems are connected to the level 2 flooding domain.

When the calculation of routes is centralized, there must still be some form of information hiding to scale the control plane. Instead of aggregation at specific topological points in the network, SDN control planes most often opt for moving to a reactive control plane, meaning the forwarding devices discover reachability information only when they receive the first packet in a flow. While this does reduce the amount of forwarding state in any particular device, it also has many drawbacks.

Specifically, reactive control planes disconnect the apparent state of the network from the perspective of any attached device from the actual state of the network. From the host's perspective, the network is up, and therefore there is a path to most destinations that begins with the first packet in a flow. In a reactive control plane, however, there is some amount of lag between the first packet in a flow being transmitted and the path actually being available. One objection to this observation is that the *Domain Name System* (DNS) is also reactive in much the same way. However, end devices generally participate in the DNS system, and hence know the state of their ability to forward in terms of name resolution.

Further, while it's always possible for the network to change state in the middle of a flow being transmitted, reactive control planes suffer from a wider set of causes for these changes. This situation is always true, of course, but while proactive control planes treat a disconnect between apparent and actual states as an error condition to be resolved, reactive control planes treat such a disconnect as a normal state of affairs. In a larger sense, disconnects between the actual and perceived states of the network are seen by attached devices as network instability; the stronger the disconnect, the more unstable the network appears to be. This condition can have an adverse effect on applications and host behavior. Local cache timeouts, cache failures, and other problems need to be included in the more general topology changes and problems common to distributed control planes for path failures.

Centralized Control and Fast Reaction to Changes in the Network

Centralized control planes disconnect local state from recalculation of the best path. If a local node or link fails, information about the state change must be transmitted to a remote device (the controller), which must recalculate a new set of paths, and then distribute those paths throughout the network. These operations can be made very quickly using techniques such as calculating and installing a backup route, but there is no simple way for a centralized controller to react more quickly, and with less chance of an unanticipated failure mode than with a distributed control plane.

The centralized/decentralized decision isn't necessarily a *better-versus-worse* decision, it's just a *different* decision with a *distinct set of tradeoffs*. Each path has its own complexities and problems to address; no set of problems seems to be much less complex to solve than any other set in this case.

The Halo of Software Development

Distributed control planes, as mentioned previously, are very complex, and they require a lot of configuration to deploy, design, troubleshoot, and manage. It seems simpler, in many ways, to just replace all the people who do this configuring, troubleshooting, and managing, with a small team of coders who can build and maintain a controller. The code would be simpler because it's all "in one place," and can be more customized to fit a particular business environment. The reality, however, is far different.

But a single controller simply won't do when it comes to scaling out a network. Even if you could run a network of thousands of routers with a single controller, it goes against every foundational concept of solid system design to do so. There must be at least two controllers, in topologically diverse locations within the network, to provide redundancy in support of overall system resilience. Moving from one controller to two inevitably means providing some way to distribute reachability and policy information between the controllers.

Ultimately, then, a distributed control plane must be built to allow communication between the controllers.[1] Couldn't this distribution just be some standard distributed database? It could, but there's a difference between distributing a database and distributing the meaning contained in the database. To distribute the meaning, you must have an agreed-on format, encoding, and other things. If you examine existing distributed routing protocol specifications, you'll find they spend a lot of time describing not only how to carry information, but also how to specify what sort of information is being carried, and consistent ways to interpret and use that information. To make multiple controller configurations successful (especially across multiple controller vendors), either it all will need to be rebuilt in an inter-controller protocol, or—perhaps simpler—the controllers could just use an existing routing protocol. Regardless of the solution chosen, the problems involved in a distributed control plane haven't been removed from the network, they've just been moved to someplace else in the network.

Further, the distributed protocols the SDN controller is designed to replace are really just other software. The complexity in these protocols comes from the propensity of engineers to push functionality into them to address an ever-expanding array of use cases. As time passes and the larger (or perhaps more obvious) use cases are handled, protocol developers chase smaller problems, finally reaching into large amounts of code for what is really a set of corner cases. But moving the development of the control plane from one place in the network to another place in the network isn't going to solve this problem—the process of accretion of new features and an ever-larger code base and inter-controller protocol specifications to support an ever-increasing set of use cases is going to remain the same.

Flow-Based Forwarding

Standard IP headers contain at least five fields of interest to network devices:

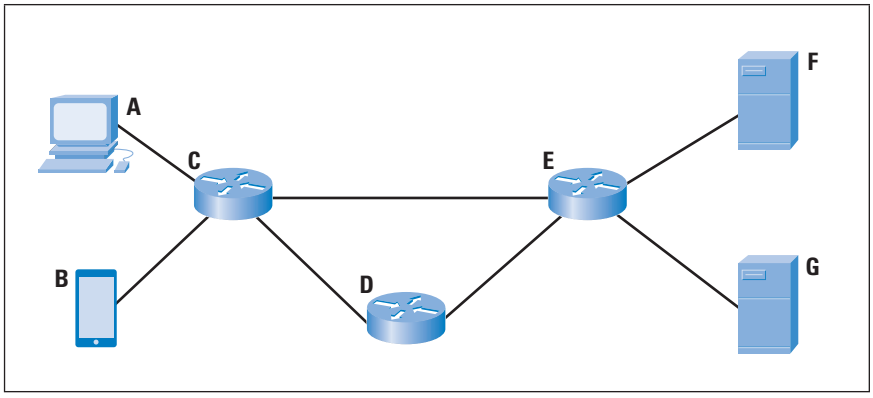
- Source IP address
- Source port number
- Destination IP address
- Destination port number
- Protocol number (or identifier)

Some transport protocols also include more information that might be of interest, such as the *Transmission Control Protocol* (TCP) socket number, which can indicate a particular application, and information about the expected quality of service for this packet (and ultimately flow). Much of this information (and more) could be used to forward traffic into multiple paths through the network based on policy and available bandwidth. However, existing routing protocols are designed to provide reachability, and hence forwarding, information based on the destination address only.

There has long been a desire to forward traffic based on much more than the destination address, so that individual applications can be independently routed through the network, and information other than the destination address can be used to deny access to specific network resources. These requirements have led to a string of work in the area of accounting for the IP source, at the least, when making forwarding decisions.[2]

Figure 1 provides an example.

Figure 1: Flow-Based Forwarding Control Example



In this example, Host A is sending a large file to Server F, while the user at Host B, a small handheld mobile device, is participating in a video conference through Server G. Assuming the two server addresses are shared among numerous different services, destination-based addresses cannot be used to differentiate between the large file transfer and the video conference. In this case, if the network administrator knows about the file transfer, the source addresses of A and B, along with the source and destination protocol information, can be used to differentiate the two traffic streams. This solution would allow the file-transfer traffic to be directed along the [C,D,E] link, while the video conferencing traffic would be directed along the [C,E] link. This traffic separation can be used to allow the video conference traffic to pass along links that aren't being heavily used by the file transfer.

Flow-based forwarding, however, presents many problems.

First, the amount of control-plane state required to forward every flow in a large network individually would be far beyond reasonable. The problem is not just the number of flows, but also the flow setup rate. To put this idea in real terms, if there are 10,000 hosts such as A and B in the illustration, and each attempts to open a different website, and each website requires 20 TCP connections, the network is required to calculate and install 2,000,000 flows in a matter of seconds. Few controllers could handle flow setup rates at this level.[3]

Second, the hardware costs of implementing such a scheme would be very high. The amount of flow state required in each device would be incredibly large—larger than most commodity hardware can support.

In addition, the cost of examining the full header on each packet at each hop in the network to achieve correct routing would be very high as well. The cost includes not only the capital expenditures (CapEx) of acquiring hardware that can support full header examination at every hop, has the table space to hold per-flow tables across millions of flows, and can support the flow setup rate required in a large fabric, but also the operating expenses (OpEx) in terms of power use for such devices. Power drives much more of large-scale design than is often considered; however small the energy cost per packet to examine the entire header at each device, it can still add up, over billions of packets switched, to significant numbers.

Third, the use cases for such flow-based forwarding, in the real world, tend to be rather narrow. Replacing the control plane that manages millions of flows through a large-scale data center fabric to support custom routing for a few thousand flows at any given time doesn't appear to be a good tradeoff in terms of complexity and network manageability.

Of course, SDNs can operate in a mode where most traffic is forwarded based on the packet destination, and the small number of flows that need special routing are handled by examining the full packet header (the five tuples noted previously or deeper), but this solution is a compromise with reality, rather than the original ideal of SDNs. The concluding section of this article considers the more realistic option of compromising with reality, so it is not covered here.

Making the Network Programmable

Finally, SDNs have promised a great deal in terms of network programmability. The breakdown involves three different areas: dynamic provisioning, and dynamic interactions between applications and the network. These topics are considered in the sections that follow.

Dynamic Provisioning

If there's one point virtually every network engineer agrees on, it's that large-scale networks are difficult to provision, monitor, and troubleshoot. It would certainly be a boon to network operations, particularly in large networks, if there were a single, unified interface into every vendor's platform, and every control-plane implementation deployed across the network, to facilitate provisioning and management. While the idea of a single interface is noble, the reality of the market is probably going to intercede—as it has many times in the past—because vendors must be able to differentiate themselves somehow in order to actually sell hardware, software, and services. This reality isn't an indictment of vendor business models, it's just reality as it exists. There are two ways to express this problem.

First, vendors try to differentiate themselves with new features, architectures, and ideas their competitors don't have. New ideas, however, require new models that can be used to configure and manage newly designed and/or modified hardware and software.

If the vendor publishes standardized models for managing these things before they are completed, they lose competitive advantage.

Second, vendors tend to be able to command higher returns on vertically integrated solutions that are easy to deploy and manage *as a unit*. Building vertically integrated solutions, however, tends to thrive on well-integrated, single-vendor interfaces between the parts.

Both of these factors place vendors in the position of trying to balance openness with profit margins. The market demands openness, but it also demands simplicity and innovation, and these goals are sometimes (or even often) contradictory from the vendor's perspective.

The most likely result of these two factors is that SDN interfaces tend to be restricted in their scope and scale to the "lowest common denominator" of available features. Some level of configuration and trace information might be available through vendor-specific extensions, but not on the "common model." Models such as *OpenFlow* tend to start with clean implementations, and then tend to fragment over time as vendors rush to build product. There is little incentive to consider additions to the base work, along with the rework such additions would require on a per-vendor basis, over time.

There is tension around automated provisioning from the network operator's side, as well. On the positive side, dynamic provisioning does take humans out of the repetitive action loop of quickly provisioning network devices and virtual topologies. Thus the speed and accuracy of configuration, provisioning, and fault isolation can be improved dramatically; in other words, automation can reduce the *Mean Time Between Mistakes* (MTBM). However, automating processes also introduce a level of brittleness into the operational cycle that can be undesirable.

Brittleness, in this context, can be seen as a set of systems that react to a wide array of situations with a small set of behaviors. Just as there can be monocultures in bacteria colonies, there can be monocultures in networks. To give a specific example, if every implementation of IS-IS in the network reacts the same way to a given situation, then it's possible for a single defect to cause every router in the network to fail under a single (though perhaps unusual) set of conditions.

The same sorts of situations can arise in provisioning or managing a network; an event that "slips through the cracks" of the automation system, or an attacker who can feel out the perimeters of defense, can take an entire system down very quickly. Another term for this situations is "robust yet fragile":

> *At some point, any complex system becomes brittle—robust yet fragile is one phrase you can use to describe this condition. A system is robust yet fragile when it is able to react resiliently to an expected set of circumstances, but an unexpected set of circumstances will cause it to fail.*[4]

The best ways to counter are to intentionally avoid monocultures where possible, and intentionally inject human decision points in the process. Reducing repetitive human work is good, but removing humans from the entire decision process is bad. This brittleness can end up replacing a large number of smaller failures due to human error and replace them with large systemic failures.

Application Interaction

Combining dynamic provisioning and dynamic policy results in what can be called an *Application Programming Interface* (API) for the network itself. Treating the network as a programmable entity allows applications to directly interact with the network as a system. The general idea can look something like this:

- An application needs a certain amount of bandwidth with specific quality-of-service parameters at a particular time.
- The application uses an interface into a controller to reserve this bandwidth, providing the controller with the impacted endpoints, etc.
- The controller uses some means to build the right network conditions to accommodate the needs of the application.

Another example might be offloading the processing of packets for security reasons into the network. Applications and operating system security are becoming more widely deployed as encryption of data in motion becomes more common. For instance, LinkedIn currently deploys *Transport Layer Security* (TLS) on all external-facing connections, and is in the midst of deploying TLS across the data center fabric among internal applications. This type of encryption reduces the usefulness of firewalls as network appliances (or a "bump in the wire") for blocking various types of attacks. The movement towards application and operating system security, however, means the host must perform all filtering, and must also forward traffic that needs to be forwarded to a honeypot or collection point for further processing. If the network has a policy interface, however, the host could instruct the controller to install policy at any point in the network that makes sense to either block or redirect attacker flows. This model would take security-related packet processing off the host and place it into the network, where specialized hardware can be deployed, and traffic can be optimally redirected or dropped more optimally.

The same objections that can be raised for dynamic provisioning can be raised for direct interaction between applications and the network, such as brittleness. To such interactions can be added the potential for feedback loops between various applications and network conditions (the main reason live measurement of network conditions was removed from the *Enhanced Interior Gateway Protocol* [EIGRP], soon after its first deployments, and replaced with relatively static metrics).

Summary of Network Programmability

Once a dynamic interface to the network as a network is in place, this abstraction can breed complexity beyond what the engineers responsible for maintaining and troubleshooting the network can readily understand. This complexity leads to several different problems, such as the "magic-button effect," where no one really knows why "doing x" solves a particular problem, but since no one can figure it out (and no one has time to figure it out), someone writes a script that "pushes the button" every time "x" happens.

Overall, then, the promise of SDNs in the provisioning space is great—but parallel complexities must be managed. At this point, there is little sense that our understanding of SDN complexity has matured to the point of being able to use the full potential of the technology in the provisioning space.

Conclusion: Looking to the Future

While SDNs aren't poised to "consume the world" in their original form because of issues surrounding centralized controllers, scale, and speed, the concepts involved are beginning to be applied to many different problem spaces. A hybrid-mode approach that allows a more standard distributed control plane to provide forwarding information for the bulk of the traffic based on the destination address, but allows overriding forwarding decisions based on other factors for a small percentage of the traffic, is gaining traction in data center fabrics of all sizes. Programmability is being used in long-haul networks, particularly in conjunction with optical transport, to handle customized forwarding as well. Essentially, the model that's being adopted in the real world is splitting policy from base reachability, leaving the base reachability under the control of distributed control planes, while moving policy-based forwarding into a controller.

Leaving the the proven scalable distributed control plane in place and using SDN to take advantage of the perks such as traffic engineering, bandwidth optimizations, intelligent routing, special policies, and other uses seems to be the most practical path forward. Network operators may find themselves deploying different mixes of SDN-type controls and distributed control planes based on application support and business strategy, but there's little doubt that both distributed control planes and what will be called SDN—programmability layered on top of the distributed control plane—will both continue to be used into the foreseeable future.

SDN is not a product. Rather, it's a methodology or tool; not a destination, goal, or product to sell, or sometimes to market, and should not be considered a target to reach but a strategy to perform certain tasks depending on real needs and if certain requirements apply.

References

[1] See, for instance, Liron Schiff, Stefan Schmid, and Petr Kuznetsov, "In-Band Synchronization for Distributed SDN Control Planes," ACM SIGCOMM *Computer Communication Review,* Volume 46, Number 1, January 2016, **http://www.sigcomm.org/sites/default/files/ccr/papers/2016/January/0000000-0000004.pdf**

[2] Such as the Internet Drafts **draft-baker-ipv6-isis-dst-src-routing**, **draft-baker-ipv6-ospf-dst-src-routing**, and **draft-ietf-spring-segment-routing**.

[3] OpenFlow 1.3 moves towards the proactive installation of forwarding-table information in recognition of the timing issues involved in reactive control planes. This ability does resolve some components of this problem, but not others.

[4] Russ White and Jeff Tantsura, *Navigating Network Complexity: Next-Generation Routing with SDN, Service Virtualization, and Service Chaining,* Addison-Wesley Professional, 2015, ISBN-13: 978-0133989359.

RUSS WHITE has more than 20 years of experience in designing, deploying, breaking, and troubleshooting large-scale networks. Across that time, he has co-authored more than 40 software patents, has spoken at venues throughout the world, has participated in the development of several Internet standards, has helped develop the *Cisco Certified Design Expert* (CCDE) and *Cisco Certified Architect Certification* (CCAR) certifications, and has worked in Internet governance with the *Internet Society* (ISOC). Russ is currently a member of the Architecture Team at LinkedIn, where he works on next-generation data center designs, complexity, and security. His most recent books are *The Art of Network Architecture and Navigating Network Complexity*. Russ holds an MSIT from Capella University; an MACM from Shepherds Theological Seminary; CCIE #2635, CCDE 2007:001, and CCAR certifications, and is currently working on a PhD at Southeastern Theological Seminary. You can find Russ at **http://ntwrk.guru/** and **linkedin.com/in/riw777**

SHAWN ZANDI is a lead infrastructure architect with LinkedIn, where he builds large-scale data center and core networks. Shawn currently lives in San Francisco, California. For the past 15 years, he has worked as network and security architect for consulting firms from Dubai to Silicon Valley. In addition to a bachelor's degree in computer science from ATS University of Technology, Shawn holds more than 40 industry certifications including triple *Cisco Certified Internetwork Expert* (CCIE) and CCDE certifications. He can be reached via **linkedin.com/in/szandi**

Thank You!

Publication of IPJ is made possible by organizations and individuals around the world dedicated to the design, growth, evolution, and operation of the global Internet and private networks built on the Internet Protocol. The following individuals have provided support to IPJ. You can join them by visiting **http://tinyurl.com/IPJ-donate**

Fabrizio Accatino
Scott Aitken
Matteo D'Ambrosio
Danish Ansari
John Bigrow
Axel Boeger
Kevin Breit
Ilia Bromberg
Christophe Brun
Gareth Bryan
Scott Burleigh
Jon Harald Bøvre
Olivier Cahagne
Roberto Canonico
Lj Cemeras
Dave Chapman
Stefanos Charchalakis
Greg Chisholm
Narelle Clark
Steve Corbató
Brian Courtney
Dave Crocker
John Curran
Morgan Davis
Freek Dijkstra
Geert Van Dijk
Karlheinz Dölger
Andrew Dul
Peter Robert Egli
George Ehlers
Torbjörn Eklöv
Peter Eisses
ERNW GmbH
ESdatCo
Mikhail Evstiounin
Paul Ferguson
Christopher Forsyth
Tomislav Futivic
Edward Gallagher
Chris Gamboni
Xosé Bravo Garcia
Serge Van Ginderachter
Greg Goddard
Octavio Alfageme Gorostiaga
Barry Greene
Geert Jan de Groot
Gulf Coast Shots
Martin Hannigan
John Hardin
Headcrafts SRLS
Edward Hotard
Bill Huber
Hagen Hultzsc
Karsten Iwen
David Jaffe
Dennis Jennings
Jim Johnston
Jonatan Jonasson
Daniel Jones
Amar Joshi
Merike Kaeo
David Kekar
Shan Ali Khan
Nabeel Khatri
Henry Kluge
Alexader Koga
John Kristoff
Terje Krogdahl
Bobby Krupczak
Warren Kumari
Darrell Lack
Yan Landriault
Markus Langenmair
Fred Langham
Richard Lamb
Tracy LaQuey Parker
Robert Lewis
Sergio Loreti
Guillermo a Loyola
Hannes Lubich
Dan Lynch
Alexis Madriz
Michael Malik
Yogesh Mangar
Bill Manning
Harold March
David Martin
Timothy Martin
Gabriel Marroquin
Carles Mateu
Juan Jose Marin Martinez
Brian McCullough
Carsten Melberg
Kevin Menezes
Bart Jan Menkveld
William Mills
Charles Monson
Andrea Montefusco
Fernando Montenegro
Tariq Mustafa
Stuart Nadin
Mazdak Rajabi Nasab
Krishna Natarajan
Darryl Newman
Ovidiu Obersterescu
Mike O'Connor
Carlos Astor Araujo Palmeira
Alexis Panagopoulos
Manuel Uruena Pascual
Ricardo Patara
Alex Parkinson
Dipesh Patel
Dan Paynter
Chris Perkins
Rob Pirnie
Blahoslav Popela
Tim Pozar
David Raistrick
Priyan R Rajeevan
Paul Rathbone
Justin Richards
Mark Risinger
Ron Rockrohr
Carlos Rodrigues
Boudhayan Roychowdhury
RustedMusic
Babak Saberi
George Sadowsky
Scott Sandefur
Arturas Satkovskis
Phil Scarr
Jeroen Van Ingen
Schenau
Roger Schwartz
SeenThere
Scott Seifel
Yaron Sheffer
Tj Shumway
Thorsten Sideboard
Helge Skrivervik
Darren Sleeth
Mark Smith
Job Snijders
Peter Spekreijse
Thayumanavan Sridhar
Matthew Stenberg
Adrian Stevens
Clinton Stevens
Viktor Sudakov
Edward-W. Suor
Roman Tarasov
Phil Tweedie
Unitek Engineering AG
John Urbanek
Martin Urwaleck
Betsy Vanderpool
Surendran Vangadasalam
Alejandro Vennera
Luca Ventura
Tom Vest
Dario Vitali
Andrew Webster
Tim Weil
Jd Wegner
Rick Wesson
Peter Whimp
Jurrien Wijlhuizen
Pindar Wong
Bernd Zeimetz

Supporters and Sponsors

Supporters

Diamond Sponsors

Ruby Sponsor

Sapphire Sponsors

Your logo here!

Emerald Sponsors

Corporate Subscriptions

For more information about sponsorship, please contact `sponsor@protocoljournal.org`

The Internet Protocol Journal
NMS
535 Brennan Street
San Jose, CA 95131

ADDRESS SERVICE REQUESTED

The Internet Protocol Journal is published quarterly and supported by the Internet Society and other organizations and individuals around the world dedicated to the design, growth, evolution, and operation of the global Internet and private networks built on the Internet Protocol.

Email: ipj@protocoljournal.org
Web: www.protocoljournal.org

Printed in the USA on recycled paper.