Chapter 2
A Brief History of Secure Email

Though language is a universal human trait, written language has a much shorter history with our species. It does seem clear from archaeological records that humans have a fascination with symbols. Symbols acquire associated meanings over time, but only those who are familiar with the context understand what those meanings are. We know little about the meaning of the Neanderthal cave drawings other than that they have something to do with animals that could be hunted with primitive weapons.

Eventually symbols became a way of representing words, and written language emerged. There is a paradox with the written word. It does not come naturally to us, it must be taught separately from spoken language, which we seem to pick up without instruction. Written language is mysterious to children, as it is to primitive peoples.

> Whence did the wondrous mystic art arise
> Of painting speech, and speaking to the eyes?
> That we by magic lines are taught,
> How both to color and embody thought?
>
> [Oft quoted historically, but source unknown]

At first, written language must have been secret in itself because so few people knew how to read. As civilization spread, people relied more and more on the written word for record keeping and communication across distance. Reading became common enough that it was not private, but the need for privacy became if anything, even more important. Roman military leaders are credited with the first uses of encryption for communication privacy [16].

But cryptography did not catch on the way written language did, and there is little evidence of its use as Europe endured the Dark Ages. Finally, in the 1200s, along with the rise of trade and increased communication via letters, cryptography found a permanent niche with the "flowering of modern diplomacy." [16, pg. 108] Since then, cryptology grown in importance, finally exploding into practical use on the Internet starting in the 1990s.

Kahn [16], the Codebreakers, page 91: "… cryptology was acquiring a taint that lingers even today—the conviction in the minds of many people that cryptology is a black art, a form of occultism whose practitioner must, in William F. Friedman's

H. Orman, *Encrypted Email*,
SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21344-6_2

apt phrase, "perforce commune daily with dark spirits to accomplish his feats of mental jui-jitsu."

At the dawn of time in the computer era, there was no email and there was no Internet. There were few computers, and there were computer programs, and people used the computers for different tasks, but there were no identifiable "users". Sending a message to another user made no sense if there was only one person at a time using the computer. Yet, because computers were so expensive and so useful, by the 1960's, there were a few computers capable of running software programs for several simultaneous users. This efficiency created a more productive environment for software development. These shared computers sprouted the seeds of email.

One early system was the time-sharing computer developed by System Development Corporation [21]. As part of a demonstration, they arranged to use three computers in three different cities. Their task was to demonstrate that data could be copied from one computer to another using phone lines for communication. The three development teams needed work together, and they came up with the idea of sending messages to each other. At first, they simply sent messages to their computer operator when they needed administrative actions like "let my program run for the next 10 min" and the operator might reply "ok, the program is running now". The messages in those days were printed on teletypes, and the operator probably knew only which teletype would print the message, not which person would read it. The developers added the capability of sending messages from one teletype to another, and this might have been the first demonstration of remote computer messaging.

As the number of computers grew, the number of users increased. The diversity of work done on a single computer increased, and administrators wanted to know who used the computer and individuals wanted to keep their data separated from the data of others. Soon the time-sharing computers began assigning "usernames". The messaging idea quickly extended to user-to-user messages in which the sender was identified by username preceding the message.

In technical terms, a message was a way to send keystrokes from one computer terminal to another, but only if the intended user was at the terminal. So being "logged in" meant sitting at sending and printing device like a teletype, or as became more and more common, a cathode ray tube with text display.

Suppose the person you wanted to communicate with was not logged in? In that case, the messaging software could create a file with the message in it, and when the user did log in, the operating system would let him know that the message was waiting for him. Oddly enough, the idea of attaching the sender's name to the message wasn't always part of the messaging paradigm.

Despite the primitive nature of early messaging, it was useful enough to become an expected attribute of an operating system. Almost as soon as the ARPANET (the precursor to the Internet) was developed, messaging was extended to allow users on different machines to send messages to one another. These early systems were idiosyncratic with respect to the form of an email address, but they allowed people on opposite sides of the country to plan where to go to dinner when developers traveled to meet their colleagues at other facilities.

By the mid 1970's most of the attributes of modern email systems were incorporated into software systems. The "@" form of addressing became universal, and messages had subjects, senders, cc's, and messages could be saved for later reading.

Very little of the information on early computers was private. If you were using a time-sharing system you had to trust your colleagues not to pry or steal or destroy your data. Even after systems like Multics and Unix added access control settings for files and software, people understood that the system administrators had access to all their data.

In 1973 Roger Shell added file encryption to the Multics operating system. A user could choose a key, enter it into the encryption program, and have his file encrypted with that key. The user could decrypt it if he entered the same key as he had for encryption. The cipher was one designed by Shell, and its design was an innovative one for that era, using data dependent rotations, among other things. Multics had user-to-user messaging and email, but encryption was not extended to those functions.

Strong protections for email were considered in the 1970s when an ARPA project sought ways to use the early Internet for classified military messages. The Military Messaging Experiment (MME) [17] defined a formal model of classification labels built into an ARPANET message handling system running on the BBN TENEX operating system. Although the MME did not envision or specify software encryption, Austin Henderson of BBN, on his own initiative, added symmetric encryption to the Hermes message handling system around 1974 [6]. Hermes was part of the normal TENEX system, and many people used it, not just those who were part of MME. This was probably the first time that email was encrypted. The Hermes system prompted the user for a text string to use as starting material for a key, that text was scrambled to form an encryption key K; then message was encrypted with key K. The resulting data was converted to an ascii text format (blocks of 5 letters separated by spaces), and it was sent using the email system. The recipient was prompted for the key, and the whole process was reversed to reveal the plaintext.

Of course, key management had to be done on an ad hoc basis. Symmetric key algorithms require that the sender and receiver share the same key, and it is difficult for two parties to agree on a key without meeting in person or making a phone call.

Fortunately, the key management problem was on the cusp of a huge change.

The Public Key Era Begins

In 1976 a group of researchers developed the idea of public key cryptography [8]. It was a brilliant discovery that established a demarcation point in the history of the field. Before public key, when two parties needed to communicate, they had to have some secure channel to let each other know what their cipher key would be. They might meet in person and tell each other, or they might agree to use some common information from a newspaper (for example, the closing value of the stock market

from the previous day), or they could establish a code, using either a code book or a convention such as 4267 meaning "the 4th word from the top of page 267 in the Merriam Webster dictionary of 1952." Key distribution was the Achilles heel of cryptography.

The astonishing contribution of public key cryptography was to let a person give one key to *everyone* and say "use this key to encrypt messages to me". With symmetric key cryptography, this would be silly because the same key that encrypts messages is the one that decrypts messages. Anyone who had the key could decrypt any messages sent to you. But with public key cryptography, the encryption key is not the decryption key. Only you know the decryption key corresponding to your own encryption key, and only you can decrypt the messages sent to you.

Public key cryptography depends on finding a function that is easy to compute if you know some extra information, but is very hard to compute otherwise. One simple way to do this is with modular exponentiation of large numbers. The word "modular" means using "clock arithmetic". In the explanations of public key methods, the notation "x mod n" means the remainder of "x" divided by "n". Mathematicians have long known that raising a number to a power using modular arithmetic is an interesting operation. For example, if p is an odd prime number and n is an integer, then there are numbers g that have the property that ad n range from 1 to $p-1$, the values of

$$g^n \bmod p$$

are all the numbers from 1 to $p - 1$ in some order. Numbers like g are called generators.

Another interesting property is the commutative property:

$$(g^a)^b = g^{ab} = (g^b)^a \bmod p$$

This leads to the elegant Diffie-Hellman key exchange method. If Alice and Bob and others have agreed to use a very large prime number p and a generating number g as their secret scheme, then they can choose secret numbers and publish public keys. Alice will tell people that her public key is the number $g^a mod\, p$ and Bob will tell people that his public key is the number $g^b mod\, p$, but each of them keeps their exponent as a secret. Then when Bob and Alice want to establish a secret number between themselves, they can both calculate g^{ab} and use that as the basis for enciphering their messages. No one else can calculate the secret exponents, even if they know the public values, because the problem of going from $g^x mod\, p$ to x is very difficult.

Shortly thereafter, a trio of MIT professors found a mathematical algorithm for public key cryptography [27], and the RSA algorithm became one of the most famous mathematical methods of computer science. The algorithm is simplicity itself. Assuming that a message M is a number (in a computer, everything is a number), then the encryption of M is the following calculation:

$$\text{enc(M)} = \text{M}^{\text{e}} \text{mod n}$$

where n is the product of two large primes and e is a number called the encryption exponent. The number n and the exponent e are called the public key. For each e, there is a corresponding secret number d that will decrypt a message. The decryption exponent is the private key.

$$(\text{enc(M)})^{\text{d}} \text{mod n} = \text{M}$$

Not only does RSA enable public key encryption, it also has a flip side that serves as an authentication signature. Reversing the designation of of "public" and "private" for d and e, we obtain:

$$\text{sign(M)} = \text{M}^{\text{d}} \text{mod n}$$

The "encryption exponent" e will "encrypt" the signature and produce the original message, thus verifying that the message was signed by someone who knows secret number d corresponding to the public number e. This is the technical of "verification" in public key systems.

With these wonderful inventions, it seemed a small step to bring cryptography to bear on its natural application in email, but in 1976 there wasn't much interest. The algorithms stressed the computing capabilities of CPUs at the time. Besides, for the most part, people who used the Internet trusted one another, and they were much more interested in promulgating information than in hiding it. This is one of the conundrums that surrounds cryptography: the suspicion that it stirs up, "What are you trying to hide?" In fact, there is very little evidence that Internet users wanted secrecy. Schell's file encryption for Multics [30] is one of the few early examples of an operating system utility for cryptographic protection for data. Nonetheless, interest in privacy and security was building steadily.

In 1980, Peter Deutsch at Bell Labs develop a user-to-user messaging system that used a public key algorithm. He used Merkle's knapsack method [24] for the functions `xsend` and `xget`. If a user wanted to send an encrypted message to another (say Alice wanted to send to Bob), then Alice would use `xsend`, and that program would look in Bob's home directory to see if he had a public key in a file with a well-known name. The `xsend` program would take Bob's public key and use it to encrypt Alice's message, and then the resulting data would be put into Bob's incoming message directory. When Bob wanted to read a message, he would use the `xget` program, and that would use the private part of his key to decrypt the file containing the message. As luck would have it, the knapsack algorithm was flawed, and this early example of secure messaging sank from view.

Interest in cryptography was sparked by two events: the publication of the NIST standard for commercial cryptography the DES cipher [5] (first proposed around 1975), and the aforementioned discovery of public key cryptography. During this time, the Internet was becoming an essential resource for communication in the

scientific community, and email systems grew from being an ad hoc collection of simple messages into an IETF standard with many independent systems for handling an ever-growing volume of messages.

By the mid 1980 s the US DoD was interested in building a secure network for military use. The Secure Data Network System (SDNS) [25] contract was awarded to several companies, including GTE. Ruth Nelson of the GTE's Electronic Defense Communications Division served as chair of the working group that defined an end-to-end encryption architecture. Their specifications for network level security were called SP3 and SP4, and they were supposed to be accompanied by a messaging protocol that would have defined secure email. However, the project's first phase ended with secure messaging remaining undone. Nevertheless, the work did provide means and impetus to explore the use of cryptography on the Internet at a time when cryptography was becoming a controversial political subject, as the US government was leery of letting the technology escape the country.

Applied cryptography on the Internet began with the Military Message Experiment [17], followed by the Secure Data Network System. In parallel, the telecommunications industry planned to be part of the networking industry. It pursued this plan through the International Standards Organization (ISO), and also through the International Telecommunications Union, a United Nations agency specializing in communication and information technologies. The ITU's consultive committee (the CCITT, later named the ITU-T), the industry produced the X.400 and X.500 [29] standards for network messages and directory services. Their joint ISO/ITU work was called the Open Systems Interconnections (OSI) protocols. Security was part of the OSI work, and the standards for encrypted content, such as network packets, were part of X.400. Those standards influenced subsequent work and were adopted for the US Department of Defense information systems. Ultimately, the DOD adopted a secure email system based on the X.400 work. The X.500 specification dealt with directory services, and one of its goals was to develop a directory structure specifically for public keys.

With the impetus of the SDNS contract, the IETF formed a Privacy Task Force (later the Privacy and Security Research Group) that quickly focused on defining standards for secure email. Their starting point was the X.400 and X.500 specifications. One of the questions they confronted was how to assign trust to a public key. While X.400 at that time specified how to use cryptography for networking, it did not yet address the problems of "interpersonal messages"—finding the private key of the recipient, how to identify which keys belonged to which people, etc. The X.500 directory services were meant to solve that problem, and part of the X.500 specification covered exactly how to identify who owned a key.

The X.400 specification was published in 1984 as "The Red Book", describing secure networking over the OSI transport model, and four years later the X.500 services group produced a standard for representing public keys and their owners. This was X.509, the certificate data structure that underlies today's website security and the S/MIME secure email standards. But in those early days of the Internet, X.509 seemed more like a solution looking for a problem.

PKI: What's Around a Name?

There is a fundamental philosophical schism that divides practitioners of authentication. That disagreement is so deep that it split secure email development into two camps.

It begins with an insight that an MIT undergraduate [19] had shortly after the RSA public key algorithm was published. Public keys are a wonderful way to start secure communication, but how can you find out what someone's public key is? Two people could send email to one another, each one showing a number that was their own public key. But how would the recipient be *sure* that it was the right key? The key might be sent by an imposter. How could you really entrust secret information to a key that arrived out of the blue? Short of meeting in person to exchange the information, could two people find a safe way to learn each other's keys?

The insight into a solution could be found in the public keys themselves. If there were a trustworthy person who could take on the task of having users validate their identities, then that trustworthy person could use his own public key to sign the binding between a person's identity and public key. The public key of the trustworthy person (or organization) could be one that was well-advertised on several reputable public websites. The signed data objects would have a binding between an "identity" (such as an ordinary name and/or an email address) and a public key. The signature of the trustworthy entity would mean "The trustworthy entity says 'this number is the public key of bob@example.com'". This simple method of authentication allowed the public keys to be freed from public directories. The keys could be stored anywhere, and they could be fetched without the need for secure communication. The signed objects, once retrieved, could be sent from person-to-person, stored on other directory servers, and yet still be verified by anyone.

But what is a name? It could be a person's ordinary legal name, an email address, or a combination of name and an organization that person belongs to—a family, a business, a school, or any number of other things. But in X.509, names identify people who have a role in a legal entity of some kind. This kind of name structure probably came into being because governments or government funded organizations designed the OSI standards. Of course, even before that, the original funding for secure messaging on the Internet came from the US Department of Defense. Their funding for the Military Message Experiment sought to embed the DoD's practice of assigning classification levels to messages, something that was eventually modeled on computer systems as the Bell-Lapadula model [3]. Besides strict ideas about message classification, the military has longstanding notions of where people belong in a hierarchy. Each member of the military is assigned to an organization within a command hierarchy, and every person has a rank within their organization. Most governments and businesses also have hierarchies, but they are often less strict about their exact memberships and how people and organizations get assigned. Outside of these formal organizations, people have even less formal designations such as friend, cousin, colleague, neighbor. The Internet brought another degree of "fuzzy" relationship to the table: people who never met in person but corresponded through

Internet email lists, bulletin boards, etc. These Internet associates might never know each other's "true names", yet they would be familiar with their opinions, preferences, and style of speech.

The X.509 system specified a way of describing a person's identity and place in a hierarchy. Along with that identifying information, there could be a public key for communicating with them confidentially. In a hierarchical organization, there must be some entity that it is responsible for identifying its members, and that is where public keys found yet another role. If each organization had a public key for signatures, then each person's directory information could be signed by the organization they belonged to. Furthermore, each organization's key could be published and signed by the organization above it in the hierarchy. At the very top of the hierarchy, one organization, the "root" would be the ultimate trusted authority.

The hierarchical structure fit naturally into governmental and corporate organizational charts. This attractiveness led to an expectation that public key technology could be an easy add-on to existing email systems. Looking up an email address and looking up a public key should be tied together into one, simple operation. That fact that this would be simple is embodied in the set of public key management operations that has come to be known as "Public Key Infrastructure" or PKI [1].

There was another aspect of certificate hierarchies that appealed to many people but repelled others. For most purposes we expect to know the "real" identity of an Internet correspondent. There is, after all, a real person sending the email, or at least an identifiable organization. For many purposes, though, the real identity is not important, and we are happy to accept a pseudonym as a reasonable identity. The practice is, of course, well established for publishing. The IETF's Privacy Task Force became the Privacy and Security Research Group (PSRG), and the members participated in discussions about what identity services to provide for secure email. Two camps emerged. One favored the idea that all identities had to be vouched for by some identifiable organization, and all organizations must be part of a hierarchy. Only a few organizations could be trusted to be "roots" of certificate hierarchies. Indeed, some people felt that the very idea of obscured identities was objectionable. Another camp recognized a need for anonymity in political discussion, and for them, the hierarchy structure looked suspiciously like de facto government controls on identity.

Secure Email Begins to Emerge

Privacy Enhanced Mail (PEM) Standardization, Part I

The PSRG moved towards defining its version of secure email [18]. The design team chose the hierarchical X.509 certificate structure as the basis for representing the binding between an identity and a public key. Not everyone was happy with the X.509 decision. Besides the argument over naming hierarchies, licensing issues dogged the requisite public key technology.

From about 1984 through 1992, the PSRG worked on defining secure email. Their work eventually was named "Privacy Enhanced Mail" (PEM). The X.400 messaging standards were the starting point for design, even though those standards were concerned with network packet and stream communication, not email.

The main technical problems to be solved by PEM were:

- to carry encrypted messages over the Internet's normal email protocol, SMTP, without changing any of the servers that forwarded email messages between sites
- use public key methods to ensure that only the intended recipient could read the message
- use public key methods to assure the authenticity of email messages
- ensure that any modifications to the message could be detected by the receiver
- facilitate the transmission of public keys

The first documents covering secure email on the Internet were issued in 1987 by the Privacy Task Force. The document was revised in subsequent years. The first version did not mention certificates, but from 1988 onward certificates were required.

PEM needed to introduce new email headers for naming the sender and receiver. Ordinary "From:" and "To:" headers do not carry enough information to unambiguously determine the public keys, so PEM added their own headers, encapsulated within the PEM message itself. PEM users could send entire certificates chains in the "X-Sender-ID" field, thus solving one of the nagging problems of public key management: how to find the keys. There were no universal directory services on the Internet, and the best way of bootstrapping the keys was to send them in email messages. This was supposed to be a temporary measure while people waiting for directory services. At the time, directory services were widely anticipated, and there were many efforts to build them based on the X.500 specifications or related work in the IETF. Yet even today, the only universal directory service is the Domain Name System (DNS).

PEM headers also carried the information about the public key algorithm, the message digest function, and the symmetric encryption algorithm. The system was "crypto agile" in that it did not have fixed algorithms built into the protocol. Instead, it anticipated that advances in cryptography would lead to changes or variety in algorithms.

To achieve these goals, PEM defined three kinds of secure messages—MIC-CLEAR, MIC, and PRIVATE. "MIC" stood for "message integrity check". MIC-CLEAR was a message that could be read without special software, but it also contained a public key signature tying the sender's identity to the message; MIC, on the other hand, had a more robust message representation encoding that was less susceptible to in-transit modifications that would cause the signature verification to fail. PRIVATE messages were encrypted with the public key of the recipient and then encoded into 80 character lines of ascii characters.There were ultimately four documents that defined PEM

RFC 1421 Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures	1993–02
RFC 1422 Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management	1993–02
RFC 1423 Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers	1993–02
RFC 1424 Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services	1993–02

Beyond these 4 documents, there were also 3 that defined modification detection algorithms: MD2, MD4, and MD5 [28]. Each method had been designed by Ron Rivest, and they reflected different balances between security and speed. None of these are considered secure today, but they were state of the art at the time.

Trusted Information Systems, a small security company in Glenwood, Maryland, developed software that implemented PEM. It ran on PCs and was integrated into the MH email system on Unix. The PEM software issued certificates for a list of users who were enrolled by an administrator, and the certificates were delivered to the users via an initial request message and reply.

The PEM designers deliberately restricted PEM to using only one public key method (RSA), one symmetric cipher (DES, in 3 "modes"), and two hash methods (MD2 and MD5). This made PEM consistent with the IETF's bias towards simplicity in implementation. That simplicity can foster faster validation of the methods by quickly getting independently developed software implementations into the hands of users. Arguably, it also minimizes the possibility of errors in implementations, thus giving greater assurance of security. Further, simple designs are easier to analyze for security problems.

It is interesting to note that even at this early date, the algorithm identifiers in PEM data headers were ASN.1 encoded.

The first implementation of PEM had only a single root key for the Internet Policy Registration Authority, and that key was built into the email client software so that all users could validate certificate chains starting from the root. Each user had a public key certificate; that certificate was attached to any signed message. Two users had to exchange signed email messages in order to learn each other's keys. Once they had done that, they could then send encrypted email. There was no central directory with public keys.

A major advocate for public key cryptography in the mid 1990's was the RSA corporation, the holder of the key patents for implementing the technology. They brought out a secure email product. The company at first worked with the IETF to define a standard for secure email, but then embarked on its own path to define the Public Key Cryptography Standards (PKCS). This left PEM in limbo.

From Out of Nowhere, Pretty Good Privacy (PGP)

In the meantime, political activist and software engineer Phil Zimmerman was one of many people frustrated by the slow pace of email security technology. Zimmerman wanted to facilitate encrypted email, and he wanted to show that it was not rocket science. In the late 1980's, he took the bull by the horns and wrote his own email security application. Taking a dig at the high assurance requirements of the IETF and the PKI requirements, he named his system Pretty Good Privacy. PGP was not derived from the PEM specifications or the X.400 precursors—it was Zimmerman's own design and implementation.

PGP was announced via a USENET group, and once it came onto the scene it become the common man's privacy solution. Although Zimmerman said that it was "an educational tool", it was fully functional open source software that could be used immediately. PEM, on the other hand, was used at only two companies, and there were no plans to commercialize it.

PGP had one major simplification over PEM that was a key point in rapid adoption. The PGP software protected messages with encryption and modification detection, but in a major departure from the IETF standards, it did not use certificates to represent public keys, and it did not use a hierarchy of trust. Zimmerman's simplifying idea was that users could develop trust in public keys through what we today call "social networking" and forgo the complication of certificate authorities and certificate hierarchies. In another departure from the IETF's philosophy of authentication, Zimmerman took pains to make the point that the trust was placed in the key itself, not any external notion of "identity". Pseudonyms for anonymous communication were an explicit goal of PGP.

PGP users could generate a public key immediately from the software distribution and begin using it. The public keys were represented in a simple block of data that could be sent in an email message to another user. Zimmerman encouraged people to put their keys on public servers established for that purpose, and MIT helped out by providing one. Although the system had seemed to have no authentication assurances, it provided a way to create a "web of trust". People could create ad hoc certificates just by signing someone else's key. If Alice convinced Bob that her public key was really one that she controlled, then he could sign her key and put that signed information onto a public server. Carol and Dave might do the same for Alice's key. Then if Ethan were looking for Alice's key, he might see that Carol's key had signed it. If Ethan already trusted Carol and her key, then Carol's signature on Alice's key might convince Ethan to trust Alice's key. Thus the web of trust (which coincided temporally with the "world wide web") began to grow.

One point of contention between PEM and PGP was the privacy of the user's identities. One of Zimmerman's design tenets was to avoid adding any unnecessary information about the email addresses or names of the correspondents in the secured part of the message. Although the unprotected Internet email headers give a lot of information about those identities, Zimmerman wanted to make sure that PGP separated the notions of Internet email identity from PGP identities and keys. If Bob

wanted to use his oddly named email account "alice@example.com" to receive PGP email protected by a key for "bob@example.com", that was fine, and the PGP sending software would not put "bob@example.com" into any unencrypted part of the email.

Having produced PGP as his "proof of concept" for email privacy, Zimmerman felt that his software code base was sufficient for widespread adoption, and he was not interested in producing an IETF standards document. Nonetheless, he was no enemy of the IETF, and he urged the PEM designers to adopt some of his most important design principles. As he presciently wrote in 1991 on the PEM Development mailing list, urging developers to avoid adding unnecessary identifying information into the plaintext parts of messages:

> One of the many reasons PEM is such a useful contribution to the body politic is that without it, the Government can routinely scan the burgeoning flow of email traffic, with far less human effort and far less visibility than they could do with paper mail. With traffic analysis alone, surveillance of political activists and who they associate with can yield useful political intelligence.

During the next several years, PGP became very popular around the world, especially because of Zimmerman's defiance of pressure from the US government to restrict distribution of his software.

The four PEM specifications were final in 1993, but by then PGP was fast becoming the de facto standard for Internet email. At the same time, the IETF's specifications for email messages had changed to allow complex documents to be transmitted, and PEM needed to align itself with the new world of MIME. Complicating things even more was a rift in the The RSA Corporation had been an active participant in the PEM design group through their employee Burt Kalisky, they were about to separate from the IETF and pursue secure email standards through a different industry consortium.

Privacy Enhanced Mail (PEM), Part II: The Tangled Tale of Standardization

PEM did not specify any particular cryptographic methods, but the naming scheme had identifiers for DES, RSA, and MD2 and MD5 as their only examples of symmetric encryption, public key encrypt/signing, and hashing, respectively. The PEM architecture constituted the first definition of a complete, secure email system. As with any pioneer, it encountered numerous problems. The full solution to secure email standardization would not come for several years after the PEM specifications were complete, but the PEM experience was invaluable for motivating the work.

There were two kinds of signing, one that left the text of the message untouched and readable without special software, another that required PEM software to

decode it. This allowed people to send signed email to non-PEM users without worrying about whether or not they had PEM software installed. A side effect of the transmission was to transmit the public key certificate to the recipient, thus bootstrapping secure communication.

PEM also supported sending encrypted email to multiple recipients. Each recipient got an encrypted copy of the symmetric encryption key in the encoded message structure.

The information about who was sending the message, and who it was destined for is information carried in ordinary email headers, but with public key technology, the name is not as important as the key. PEM had its own internal headers that defined the "IDs" of the sender and receivers. Those IDs could be public key identifiers or email names.

PEM was designed to work seamlessly over existing email services, so it needed a way to encode binary data, like signatures or encrypted data, into the ascii character set that Internet email used. The binary data used octets (or bytes) of 8 bits each, but the ascii character set used fewer than half of the 256 possible octet values. The encoding scheme that was chosen was a "3-to-4" map that would take 3 binary octets (24 bits), break them into 4 groups of six bits each, and then map each six bit quantity to a unique ascii character. Long before PEM, a similar method was employed in the Unix utility "uuencode". This is a form of radix-64 encoding, and PEM's variant on it, named "base64", turned out to be a lasting legacy of PEM, though all other details have faded away. Yet PEM was important as the motivator for much needed revisions to what would become known as Public Key Infrastructure (PKI).

Users of PEM needed to let their correspondents know about their public keys, and for this purpose PEM used public key certificates as defined in the ITU-T specification X.509. In these early days of public key technology, the PEM designers envisioned only one hierarchy with a single authority at the root—the Internet Policy Registration Authority(IPRA). The second level authorities would reflect different "policies", such as a state government that certified state agencies and cities, or a business directory service that certified non-profit corporations, or an entity that specified acceptable personal identifications (driver's licenses or passports, etc.) for "persona" identificaton. The certificate authorities were at the third level and below, down to the individual users. Further, the X.509 system for names of certificate authorities assumed a parallel hierarchy (e.g. california.fresno. department-publicworks.northwest-office). As various organizations tried to adopt PEM, the strict rules for certificates became a major impediment. In the real world, businesses merge, government agencies split and reorganized, and names change.

PEM had a second specification entirely devoted to key management. RFC1422 covered a PEM message type for a "Certificate Revocation List" (CRL). These were intended to be a stopgap measure until widespread adoption of the X.509 services occurred. In the meantime, PEM's CRL message was a simple and reliable method for distributing the CRLs to end-users.

X.509 certificates were too difficult to use for large PEM deployments. The second level authorities' policies were difficult to learn or explain, the naming

scheme was unwieldy, CA certificates had no syntactic distinguisher from end user certificates, the revocation scheme did not scale, and the top-down signing scheme had policy and management implications that needed to be addressed by putting more information into the certificates. Administrators found that they needed more information about certificate authorities in order to understand if they were trustworthy. The hierarchy sometimes needed to be shortened by allowing a CA in one branch of a hierarchy to sign the certificate for a CA in a different branch. After a brief attempt to fix things by adding two new fields and calling it "X.509 version 2", the standardization groups retrenched. By 1998 they had developed a major rework of certificates with more fields, more flexible naming, formal definitions of CRLs, cross-signing, and a flexible system for adding new fields through formal extension fields. The X.509 version 3 certificate has proved an enduring legacy of the PEM experience.

One interesting point about X.509v3 today is that it is a standard defined by the ITU-T but separately defined by the IETF. The IETF defines the requirements for certificates used on the Internet for such things as email, secure communication channels, and website security, but these are consistent with the more general ITU-T definition. Nonetheless, in the interests of stability, the IETF publishes its own complete definition of Internet certificates. One standard, two heads!

PEM and MIME

But no sooner had the specs been finished than the task force discovered that they had a problem. While they had been working on secure email, other groups in the IETF had been defining a more general way of sending complex data. People needed to send photos, audio data, and even video, and the IETF had defined a way to send arbitrary data, even disparate kinds of data, in a single email message. This was the MIME standard, and it is what enables today's email attachments.

Encryption turns meaningful data into meaningless data, and that data is not something that humans can read. Email is geared towards a small character set, and the Internet email protocol, RFC822 and its successors, cannot transmit binary data. From this it follows that the data must be encoded into a smaller character set. PEM had solved this problem with "base64" encoding that packed 3 bytes of binary data in 4 bytes of ascii data. Yet this brought up another problem—how would the email handler for the recipient know that the message was encrypted? Email had already faced similar problems, and the usual solution was to add some extra information to the beginning of email. A line might be as simple as:

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
```

Though this seemed logical, it ran into a variety of problems. Email software sometimes tried to be helpful and slightly adjust details of the message formatting, add ing or deleting a whitespace character here or there, and changing the character to indicate end of line, etc. This did not usually change the readability of normal

email, but it played havoc with encrypted email and signed email. Cryptographic techniques would not work unless it could rely on software built to standards that guaranteed that headers could not be added and removed willy-nilly.

Encrypted email is not the only kind of binary data that people exchange, and a different part of the IETF had been working on this problem in the context of multimedia data. The MIME extensions to email were far along, and PEM was out of step with MIME conventions. In fact, MIME is what makes email attachments possible. We can send photos, music, compressed files, complex documents, and all manner of data in email attachments, and we expect it to work without a glitch. What the PEM group faced was the fact that all email clients would soon be supporting MIME, but they were not certain that they would support PEM, especially if its format was unrelated to MIME.

MIME Security Object Security Services (MOSS)

In 1994 the PEM working group decided that they needed to integrate their work with a major addition to Internet email, the Multipurpose Internet Mail Extension (MIME). MIME was becoming the accepted way to encode binary data files in email messages, but PEM's methods worked only for simple text messages. Moreover, a great benefit of MIME was that it could describe separate message parts within a single email message. As a result, MIME was an obvious and convenient mechanism for encoding "privacy enhancements". A marriage was in order.

Up until then, PEM used its own special markers to delineate secure email content:

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
```

and

```
-----END PRIVACY-ENHANCED MESSAGE-----
```

with the expectation (or hope) that email software would not alter the content between the two markers. This simplistic method would not survive into the era of multipart messages with special encodings for binary contents.

A new IETF working group for PEM+MIME was formed, and it defined extensions to MIME for security enhancements based on PEM. This group defined the the standards for MIME Object Security Services (MOSS). MOSS used headers at the start of a message to announce that the message had internal MIME parts with security enhancements. Each security-enhanced MIME part was of either type "signed" or "encrypted". Each enhanced part had two subparts. One subpart was the cryptographic control information about either the signature or the encryption, and the other subpart was either the data to which the signature applied or the encrypted data. MOSS also addressed some of the aspects of key management by defining special MIME parts for requesting and sending public keys.

The MOSS designers sought to harmonize their work with a different secure email system, PGP. In doing so, they compromised on the tenet of PEM that mandated X.509 certificates and names. MOSS enlarged the identifier space to allow email names, the "distinguished name" in a certificate, or any arbitrary character string. This last form of ID opened the door to PGP key digests.

All in all, MOSS was a great improvement to PEM. Security enhancements could be extended to individual message parts using the MIME standards, which were already becoming popular. The victory was hollow, though, because by the time MOSS was published in 1995, it was already destined for the scrapheap.

During the time that the PEM was evolving to MOSS, the RSA corporation was working on its own version of secure email. Their interests were wider than the PEM working group's charter. RSA was interested in algorithm-independent coding of cryptographic parameters, secure representation of different kinds of public keys, extensions of X.509 certificates, certificate management, etc. They decided to work with an industry consortium instead of the IETF.

PKI, PKCS, and S/MIME

The whole subject of public keys, including all the representation and management utilities, comes under the umbrella term Public Key Infrastructure (PKI). Certificates are covered by the X.509 standard, and all the rest is covered by a group of standards that were originally called Public Key Cryptographic Standards (PKCS). They have a complicated history.

Apart from the indicators in an email message to denote that the message had encrypted content, there were a number of other issues that needed to be settled before certificiate-based email protections could reach the level of being well-defined and broadly useful. There were major issues to be dealt with concerning the definition and representation of a certificate, the meaning of terms within a certificate, the representation of keys, names, and signatures, requesting signatures for new certificates, dealing with keys that were unneeded or compromised, securely transferring private keys, etc. The standardization work on these issues bounced around among various groups before settling out into the current situation of cooperative bifurcation between the ITU-T and the IETF.

While the ITU-T remains the authority on X.509 certificates, Internet infrastructure providers rely on the IETF's X.509 version 3 specification. That specification is fully compatible with the ITU-T's definitions, but the IETF maintains its own documents that detail how certificates are represented and used for Internet purposes. The IETF documents also specifiy how Certificate Revocation Lists are represented.

The IETF has developed its own syntax for representing cryptographic data, CMS. That syntax is used for S/MIME data: key identifiers, algorithm identifiers, parameters, etc. The IETF also has its own format for bundling up keys for secure transfer.

The Public Key Cryptography Standards (PKCS) have a complicated history intertwined with many different organizations. PKCS started as an independent,

non-IETF activity, just as the PEM and MOSS standards were being finalized. PKCS were supposed to define the cryptographic representations of data that are essential for interoperable secure email.

The responsibility for PKCS drifted between organizations for several years. During the 1990 s, the RSA corporation and the consortium that it aggregated developed the first set of PKCS specifications. These standards developed more detail and scope for the public key cryptography and data representations than the IETF groups had covered.

Over the years there have come to be 15 different parts to PKCS. Some of them were abandoned, some were brought to fruition, some continue to be developed. The original industry consortium left the work of completing the standards to the Open Systems Interconnection group within the CCITT (which became ITU-T). In 1995 the IETF, with cooperation from NIST, created the "Public Key Infrastructure X.509 (PKIX)" working group to develop refinements (profiles) of the ITU-T PKCS so that Internet services for email and website security and infrastructure security could be built on them. Moreover, the PKIX group saw the need to develop new services, particularly online services, for certificate management and Internet transport. Over time, the "profiles" became independent standards, usually compatible with the ITU-T, but not bound by them.

The PKCS#11 standard is currently under active development under the auspices of the OASIS consortium (https://www.oasis-open.org/).

A summary of PKCS parts (from the OASIS website):

- PKCS #1: mechanisms for encrypting and signing data using the RSA public key cryptosystem. This became the IETF's RFC 3447, issued in 2003.
- PKCS #3: a Diffie-Hellman key agreement protocol.
- PKCS #5: a method for encrypting a string with a secret key derived from a password. This became RFC 2898.
- PKCS #6: certificate extensions; phased out in favor of version 3 of X.509.
- PKCS #7: a general syntax for messages that include cryptographic enhancements such as digital signatures and encryption. This was absorbed into S/MIME and its multitude of documents.
- PKCS #8: a format for private key information. This became RFC 5208, a short description of the format for representing private keys and protected with symmetric encryption.
- PKCS #9: selected attribute types for use in the other PKCS standards. Published as an informational RFC (i.e., not part of other IETF standards).
- PKCS #10: syntax for certification requests. Published as an informational RFC (i.e., not part of other IETF standards).
- PKCS #11: a technology-independent programming interface, called Cryptoki, for cryptographic devices such as smart cards and PCMCIA cards. Currently under development by OASIS.
- PKCS #12: a portable format for storing or transporting a user's private keys, certificates, miscellaneous secrets, etc. In 2014, this standard was transferred from the RSA Corporation to the IETF, becoming RFC 7292. It builds on

PKCS#8 and greatly extends it. [The "p12" file format has been used for bundling key for transporting to email and web clients for some time.]
- PKCS #13: mechanisms for encrypting and signing data using Elliptic Curve Cryptography. Apparently abandoned.
- PKCS #14: pseudo-random number generation. Abandoned.
- PKCS #15: a complement to PKCS #11 giving a standard for the format of cryptographic credentials stored on cryptographic tokens. Standardized by the International Organization for Standardization as ISO/IEC 7816-15:2004

The PKCS#12 standard became the IETF's "Personal Information Exchange" standard for encoding and protecting public/private key pairs when they are moved from one device to another. PKCS#7 survives in the signature attachment filename "smime.p7 s". A few of the other PKCS sections live on as the basis for RFC's on specialized methods of working with hardware devices. The OASIS organization continues work on some of the PKCS subsections.

A certificate is, at its heart, simple to describe: a public key, the identity of the owner, the signer's key, and the signer's signature over those three crucial elements. Yet, the work of obtaining a stable definition has been spread over three decades. The PKIX group published "Internet X.509 Public Key Infrastructure Certificate and CRL Profile" in 1999. The final version was published in 2008. It turns out that using and managing certificates is fraught with complexity, and that complexity has become embodied in the certificate definition.

The original X.509 specification from the X.500 Directory Services specification covered the basic requirements for transmitting, identifying, and authenticating public keys within a certification hierarchy. As security administrators gained experience from trying to use the architecture for large numbers of users and with non-trivial hierarchy depths, they felt that it would be helpful to have more information to facilitate management functions.

A second version of X.509 added two more fields, but the response from administrators was that they needed still more. As the standardization groups worked to find consensus on a small number of fields, they came to realize that the diversity of opinions on how to establish the trustworthiness of certificiate issuer was going to rely on more fields. Some of those fields would be using data that might be unique to only a section of the hierarchy. Version 3 of the standard was forged to satisfy diverse needs by adding the "extension" capability, leaving the original fields as required, adding new fields, and allowing an unlimited number of new fields in the certificate extensions.

The Cryptographic Message Syntax, CMS

The syntax for representations of cryptographic messages was covered in the seventh of the PKCS series, and it was taken under the wing of yet another IETF working group—"networks". The CMS standard [15] is based on a formal

definition of the syntax elements for representing algorithms, algorithm parameters, encrypted data, nonces, etc. That syntax and its representation in terms of bytes and bits became the basis for representing data in certificates and secure email attachments.

The things that go into cryptographic syntax are usually different for each algorithm. The following show how public keys are defined in CMS. The first data structure defines the generic data structure for a public key, and it has an identifier at the beginning signifying that it is an RSA public key type, there are optional parameters, and there is a bit that determines the purpose of the key. The second data structure defines the public key elements (the modulus and the exponent), and the third data structure defines the meaningful names of the usage bits.

```
pk-rsa-pss PUBLIC-KEY ::= {
    IDENTIFIER id-RSASSA-PSS
    KEY RSAPublicKey
    PARAMS TYPE RSASSA-PSS-params ARE optional
    CERT-KEY-USAGE { .... }
 }

RSAPublicKey ::= SEQUENCE {
         modulus           INTEGER,  -- n
         publicExponent    INTEGER   -- e
  }

KeyUsage ::= BIT STRING {
          digitalSignature        (0),
          nonRepudiation          (1), -- recent editions of X.509 have
                                  -- renamed this bit to contentCommitment
          keyEncipherment         (2),
          dataEncipherment        (3),
          keyAgreement            (4),
          keyCertSign             (5),
          cRLSign                 (6),
          encipherOnly            (7),
          decipherOnly            (8) }
```

Whereas previous security standards had used their own conventions for representing the names of algorithms (e.g. “RSA” or “DES”), and their own methods for designating a symmetric key or a public key, CMS presented a uniform method for representing all things cryptographic. This meant that developers needed only one software library to read and write cryptographic information for use with any almost any IETF standard.

The underlying lexicographic representation of CMS elements is an abstract notation called ASN.1. ASN.1 describes numbers, strings, object identifiers, time, etc. The actual representation of ASN.1 in bytes is described by its oft maligned Binary Encoding Rules (BER). BER has several ways to represent numbers and strings, and even the same value can be represented in more than one way.

This thwarts the precision demanded by cryptographic algorithms. Therefore X.509 and CMS use a subset of BER call the Distinguished Encoding Representation (DER). A frequent complaint about BER/DER is that its binary data representation is complicated, and as a result, the CMS data structures cannot be parsed by casual examination of the byte strings.

S/MIME, Secure/Multipurpose Internet Mail Extensions

Another product of the PKCS industry working group was a set of extensions to the IETF MIME protocol. The extensions were simply a set of additional headers indicating that a message part had cryptographic enhancements. These could be encryption or signing or a certificate management function. The body of a protected MIME part was encoded in a format that was the subject of another PKCS work topic denoted as PKCS#7. In 1998, the consortium documented this work in an IETF document as "S/MIME version 2". The document is careful to note that it is not an IETF standard, but it has historic interest. The successor protocol, S/MIME version 3, became in IETF standard in 1998. This was a mere five years after MIME itself was published as a standard.

The IETF's S/MIME working group produced the standards that are the final word on adding certificate-based cryptography to MIME objects. The group needed to pick up from where PEM and MOSS had left off, extending their work with new MIME headers and adding a much more flexible cryptographic representation structure. The working group did a prodigious amount of work in achieving its purpose and extending it. Over its lifetime, the group produced 50 documents covering everything from using X.509v3 certificates to use of Identity-Based Encryption [4], and to avoidance of "small subgroup attacks."

It is important to note that MIME is used for more than just email, and S/MIME can be applied to any MIME object. HTTP can transfer MIME objects, and S/MIME headers are often useful on files that have been signed or encrypted.

Despite, or because of, its simplicity and resemblance to earlier ways of attaching security headers, S/MIME succeeded in becoming widely used. From its first version in 1999 until its most recent revision in 2010, it has kept up-to-date on cryptographic algorithms.

S/MIME defines one new MIME "media" type to indicate that there are security enhancements in the data that follows: application/pkcs7-mime. That media type can be followed by a parameter indicating that the data is encrypted: "smime-type = enveloped-data" or "smime-type = signed-data".

There is a second way to represent signed data, one that largely transparent to those who do not use secure email readers, making it the most commonly used security enhancement. This is the "detached signature" which fits into the MIME

multipart scheme very nicely with the addition of the Content-Type value "multipart/signed":

```
Content-Type: multipart/signed;
   protocol="application/pkcs7-signature"; micalg=sha1
```

In the example above, the MIME part precedes two subparts, one with the message content, which can be any kind of MIME part, and the second with the smime-type of "signed-data".

Most people who receive these multipart signed messages are only aware of the signature part because it appears as an attachment with the filename "smime.p7s". The S/MIME signature on the message is encoded in that attachment, as are the sender's certificates.

The signature hash algorithm, in the example above, is the "micalg" (message integrity algorithm) SHA1. It may seem strange to see the algorithm mentioned in the header, but implementors like to have it revealed before they do any further cryptographic processing. They can apply the algorithm to the preceding MIME part immediately, and then the hash value is ready to compare against whatever is encoded in the signature.

The reader might wonder, where's the cryptography? After all, the headers are merely signals that cryptography is happening somewhere, but where are the details? All the cryptographic details about algorithms and lengths and sizes are in the data following the S/MIME headers, encoded in the Crypographic Message Syntax. S/MIME adds a few data types of its own to CMS, and those, in combination with the additional MIME types, constitute S/MIME.

The file extensions in S/MIME headers are a holdover from the original PKCS specifications, and they serve an advisory role today, because the CMS payload supercedes its function. The "p7s" extension indicates a signature, "p7m" is enveloped data (signed and/or encrypted), "p7z" is compressed data, and "p7c" means that certificates are the only payload.

In earlier versions of S/MIME, there was some support for certificate management, but today's S/MIME leaves most of the work to the protocols designed specifically for that purpose by the PKIX working group. S/MIME does carry certificates as part of a signed message, encoded in the CMS data that is the signature. This supports the "exchange signed messages once; encrypt thereafter" model of person-to-person certificate bootstrapping.

Two IETF Request for Comments (RFC) make up S/MIME version 2: RFC 2311 (http://www.ietf.org/rfc/rfc2311.txt), which established the standard for messages, and RFC 2312 (http://www.ietf.org/rfc/rfc2312.txt), which established the standard for certificate handling. Together, these RFCs provided the first Internet standards-based framework that vendors could follow to deliver interoperable message security solutions.

After all the different attempts to define security headers for email, it is strange that S/MIME is not completely transparent. The answer to the question of "what is an S/MIME message" is best answered from one of its defining documents [26]:

```
Because S/MIME takes into account interoperation in non-MIME
environments, several different mechanisms are employed to carry the
type information, and it becomes a bit difficult to identify S/MIME
messages.  The following table lists criteria for determining whether
or not a message is an S/MIME message.  A message is considered an
S/MIME message if it matches any of the criteria listed below.

The file suffix in the table below comes from the "name" parameter in
the Content-Type header field, or the "filename" parameter on the
Content-Disposition header field.  These parameters that give the
file suffix are not listed below as part of the parameter section.

Media type:  application/pkcs7-mime
parameters:  any
file suffix: any

Media type:  multipart/signed
parameters:  protocol="application/pkcs7-signature"
file suffix: any

Media type:  application/octet-stream
parameters:  any
file suffix: p7m, p7s, p7c, p7z
```

An example of a message encrypted and encoded as an S/MIME message (from RFC 5751):

```
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
       name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

Understanding S/MIME in enough detail to implement it means conquering ASN.1, DER, CMS, X.509v3 certificates, base64 encoding, RSA and DSA public key algorithms and their key encodings, the representation of a public key signature, the SHA1 and SHA2 hash algorithms, the AES symmetric cipher and its encoding, the 3DES symmetric cipher and its encoding, representing the public key encryption of a symmetric cipher key, and the S/MIME headers for email.

OpenPGP: PGP as an IETF Standard

A completely independent concept for secure email took a more direct line from initial concept to its embodiment today as an IETF standard with open source and commercial implementations.

As noted previously, PGP gained immediate popularity, and Zimmerman spoke out as an advocate of privacy technology for the masses. In 1994, with the help of an international team of collaborators, PGP 2.0 hit the streets, and Zimmerman found himself embroiled in legal issues arising from the US government's export limits on cryptography and patent infringement claims related to his use of the RSA public key algorithm. As Zimmerman worked with a team of lawyers to resolve these problems, PGP gained traction as the solution to the private communication needs of Internet users.

PGP 3.0 came out in 1996, and it was a major rework of the system. It added ciphers that avoided patent issues, and it was designed as a software library for developers instead of being simply an "app".

Shortly afterward, the IETF undertook the task of standardizing PGP. The PGP team felt this was necessary to ensure that PGP could be both a commercial product and a freely available application. The IETF working group used "OpenPGP" as their name for the encrypted email system. In sharp contrast to the S/MIME and PKIX documents, OpenPGP is a compact and complete specification for encrypted email.

- RFC 1991 PGP Message Exchange Formats, August 1996. This documented the PGP 2.x implementations that were by then in use around the world.
- RFC 2015 MIME Security with Pretty Good Privacy (PGP), October 1996. The MIME headers for PGP messages were defined here, much as PEM had done with MOSS. It was a simple scheme that allowed PGP to be easily encapsulated with headers indicating encryption or signature or key content.
- RFC 2440 OpenPGP Message Format (obsolete), November 1998. This documented the design of the PGP 3.0 system (aka PGP 5.0).
- RFC 3156 MIME Security with OpenPGP, August 2001. This defines the same MIME headers as in RFC 2015, but it extends them with parameters for more algorithms than PGP supported in the 1990 s.
- RFC 4880 OpenPGP Message Format, November 2007. This replaced RFC 2440; the changes were largely editorial and did not affect interoperability. The doubling of the RFC number from the 1998 version is an amusing footnote on PGP's standardization.

PGP never saw the need for S/MIME. Instead, the designers opted for a scheme more akin to the PEM MOSS headers. The MIME encoding for OpenPGP is simple. An encrypted message has this information in the email message header:

```
Content-Type: multipart/encrypted; boundary=xxx;
   protocol="application/pgp-encrypted"
```

The message will have two internal MIME parts. One simply repeats the encryption information:

```
Content-Type: application/pgp-encrypted
```

and the other has the encrypted, encoded data with a header for arbitrary encoded data followed by PGP's internal header:

```
Content-Type: application/octet-stream

-----BEGIN PGP MESSAGE-----

      ...
```

A signed message is equally simple. The main message header signals the signature data, and names the algorithm that assures data integrity:

```
Content-Type: multipart/signed; boundary=yyy; micalg=pgp-md5;
protocol="application/pgp-signature"
```

The signed message has two MIME parts, one with the actual message, and a second part with the PGP signature preceded by the header:

```
Content-Type: application/pgp-signature
```

A third header type signals that a PGP key block follows:

```
Content-Type: application/pgp-keys
```

PGP, like PEM and S/MIME, uses radix-64 encoding to turn binary data into ascii characters that the email transfer protocol can deal with. PGP uses a slightly different character set and calls the result "ascii armor" (asc). PGP's internal "PGP MESSAGE" delimiters enclose ascii armored data.

The choice of supported ciphers for PGP has always included more variety than the IETF embraced. In its early days, PGP favored ciphers developed outside the United States. OpenPGP today recognizes Blowfish, Twofish, CAST5, AES, and 3DES as symmetric ciphers. The Camellia cipher was added in 2004.

OpenPGP recognizes RSA, ElGamal, and DSA as its public key methods. In 2012 the definitions for elliptic curve cryptography were added. The PGP implementation in the GNU Privacy Guard (GPG) software supports elliptic curves as of release 2.1.

The number of pages of standards devoted to describing OpenPGP is far less than that devoted to S/MIME. The two systems had radically different evolutions from concept to running code, but they are really nothing more than two variants of the same core idea: public key cryptography protecting the secrecy of symmetric cipher keys, and encrypted data embedded in Internet email.

http://www.springer.com/978-3-319-21343-9

Encrypted Email
The History and Technology of Message Privacy
Orman, H.
2015, VII, 100 p. 4 illus., Softcover
ISBN: 978-3-319-21343-9