Opportunistic Keying as a Countermeasure to Pervasive Monitoring

Stephen Kent
BBN Technologies

Abstract

This document was prepared as part of the IETF response to concerns about “pervasive monitoring” (PM) [Farrell-pm]. It begins by exploring terminology that has been used in IETF standards (and in academic publications) to describe encryption and key management techniques, with a focus on authentication and anonymity. Based on this analysis, it propose a new term, “opportunistic keying” to describe a goal for IETF security protocols, in response to PM. It reviews key management mechanisms used in IETF security protocol standards, also with respect to these properties. The document explores possible impediments to and potential adverse effects associated with deployment and use of techniques that would increase the use of encryption, even when keys are distributed in an unauthenticated manner.

1. What’s in a Name (for Encryption)?

Recent discussions in the IETF about pervasive monitoring (PM) have suggested a desire to increase use of encryption, even when the encrypted communication is unauthenticated. The term “opportunistic encryption” has been suggested as a term to describe key management techniques in which authenticated encryption is the preferred outcome, unauthenticated encryption is an acceptable fallback, and plaintext (unencrypted) communication is an undesirable (but perhaps necessary) result. This mode of operation differs from the options commonly offered by many IETF security protocols, in which authenticated, encrypted communication is the desired outcome, but plaintext communication is the fallback.

The term opportunistic encryption (OE) was coined by Michael Richardson in “Opportunistic Encryption using the Internet Key Exchange (IKE)” an Informational RFC [RFC4322]. In this RFC the term is defined as:

> … the process of encrypting a session with authenticated knowledge of who the other party is without prearrangement.

This definition above is a bit opaque. The introduction to RFC 4322 provides a clearer description of the term, by stating the goal of OE:

> The objective of opportunistic encryption is to allow encryption without any pre-arrangement specific to the pair of systems involved.

Later the RFC notes:

> Opportunistic encryption creates tunnels between nodes that are essentially strangers. This is done without any prior bilateral arrangement.

The reference to “prior bilateral arrangement” is relevant to IPsec but not to most other IETF security protocols. If every pair of communicating entities were required to make prior bilateral arrangements to enable encryption between them, a substantial impediment would exist to widespread use of encryption. However, other IETF security protocols define ways to enable encryption that do not require prior bilateral arrangements. Some of these protocols require that the target of a communication make available a public key, for use by any initiator of a communication, but that is not a prior bilateral arrangement.

The definition provided in [RFC4322] is specific to the IPsec [RFC4301] context and ought not be used to describe the goal noted above, as a countermeasure to PM. IPsec implements access controls and thus requires explicit specification of how to process all traffic that crosses an “IPsec boundary” (inbound and outbound). Traffic is either discarded, permitted to pass w/o IPsec protection, or protected using IPsec. The goal of OE is to enable IPsec protected communication without a priori configuration of access control database entries at each end (hence, bilateral). Opportunistic encryption calls for each party to identify the other, using IKE [RFC2409] authentication mechanisms, so it is not an unauthenticated key management approach. Also note that RFC 4322 describes OE relative to IKE, as it should; IPsec implements encryption using ESP [RFC4303]. ESP usually provides data integrity and authentication, as well as confidentiality, thus the phrase opportunistic encryption is unduly narrow relative to the anti-PM goal. OE for IPsec is described in more detail in Section 3.

RFC 4322 also defines anonymous encryption:

> Anonymous encryption: the process of encrypting a session without any knowledge of who the other parties are. No authentication of identities is done.

Thus, in RFC 4322, the term anonymous encryption refers to encrypted communication where neither party is authenticated to the other. Also note that the definition above refers to “the process of encrypting a session …” In fact, it the key management process that causes an encryption session to be authenticated, or not. Thus it is more accurate to refer to “anonymous keying”, rather than anonymous encryption. This document adopts this convention, although most IETF security standards refer to encryption, per se, as being authenticated of not.

It is useful to distinguish two classes of anonymous keying, based on whether one party or both are not authenticated. For example, TLS ([RFC2246], [RFC4356], and [RFCC5246]) typically is used in a fashion that provides server, but not client, authenticated communication. However, TLS also supports two-way authenticated sessions and "pure" unauthenticated sessions, in which neither party asserts an identity during the handshake protocol. Thus TLS offers (pure) anonymous keying and client-anonymous keying. (The same analysis applies to DTLS [RFC6347].)

[insert brief HTTPS [RFC2818] discussion here]

Some security experts distinguish between anonymity and pseudonymity. Pseudonymity [cite] implies use of a identifier, but one that represents a "false name" for an entity. Use of pseudonyms is common in some Internet communication contexts. Many Gmail, Yahoo, and Hotmail mail addresses likely are pseudonyms. From a technical perspective, a pseudonym is an attractive way to provide "unauthenticated" communication. A pseudonym typically makes use of the same syntax as a authentic identity, and thus protocols designed to make use of authenticated identities are compatible with use of pseudonyms, to first order.

"Traceable Anonymous Certificate", is an Experimental RFC [RFC5636] that describes a specific mechanism for a Certification Authority (CA) [RFC5280] to issue an X.509 certificate with a pseudonym. The goal of the mechanisms described in that RFC is to conceal a user's identity in PKI-based application contexts (for privacy), but to permit authorities to reveal the true identity (under controlled circumstances). This appears to be the only RFC that explicitly addresses pseudonymous keying; although it uses the term "pseudonym" extensively, it also uses the term "anonymous" more often, treating the two as synonyms.

Self-signed certificates [RFC6818] are often used with TLS in both browser and non-browser contexts. In the HTTPS (browser) context, a self-signed certificate typically is accepted after a warning has been displayed to a user; the HTTPS [RFC2818] requirement to match a server DNS name against a certificate Subject name does not apply in non-browser contexts. The Subject name in a self-signed certificate is completely under the control of the entity that issued it, thus this is a trivial way to generate a pseudonymous certificate, without using the mechanisms specified in [RFC5636]. Thus support for pseudonymous keying is supported in web browsing, as a side effect of this deviation from [RFC2818]. (There is speculation that most self-signed certificates contain accurate user or device IDs; the certificates are used to avoid the costs associated with issuance of certificates by WebPKI CAs.)

Based on the examples above, one could define an additional term: “pseudonymous keying”. Pseudonymous keying refers to techniques used to distribute keys when an authentication exchange is based on a pseudonym, e.g., a self-signed certificate containing a pseudonym. As with anonymous keying, pseudonymous keying may apply to one or both parties in an encrypted communication. One also can imagine mixed mode communications, e.g., in which anonymous keying is employed by one party and pseudonymous keying is employed by the other.

An examination of about 70 papers published in ACM, IEEE, and other security conference proceedings identified numerous uses of the terms opportunistic and anonymous encryption. Most, though not all, of the papers used opportunistic encryption and anonymous encryption as defined in [RFC4322], but in some papers the terminology was unclear or inconsistent with the [RFC4322] definition.

Another popular source (Wikipedia) uses a somewhat different definition for opportunistic encryption. Wikipedia (http://en.wikipedia.org) provides the following definition:

> Opportunistic encryption refers to any system that, when connecting to another system, attempts to encrypt the communications channel otherwise falling back to unencrypted communications. This method requires no pre-arrangement between the two systems.

This definition shares some aspects of the RFC 4322 definition, but it is not exactly equivalent; it makes no mention of authentication or access control, two essential aspects of opportunistic encryption as per [RFC4322].

The Wikipedia article goes on to state that opportunistic encryption can be employed with other protocols. The article describes the potential for opportunistic encryption based on the use of self-signed certificates with TLS, instead of certificates issued by a “certificate [sic] authority.” This use of the term is at odds with [RFC4322] and with TLS RFCs, which define anonymous encryption differently.

The Wikipedia article further notes that use of self-signed certificates with HTTP [sic] (really HTTPS), will result in warning to users, unless browser extensions are employed, which makes user acceptance of such problematic. It cites use of the STARTTLS extension for SMTP [RFC3207], and use of TLS with IMAP, POP3, and ACAP [RFC2595], as ways of achieving opportunistic encryption for these protocols, when self-signed certificates are employed. Use of a self-signed certificate that does not attest to an authentic identity is an example of pseudonymous keying.

ZRTP [RFC6189] is cited in the Wikipedia article as an example of opportunistic encryption for VoIP. ZRTP uses ephemeral Diffie-Hellman

key management, and thus is more accurately described as offering two-way anonymous encryption. ZRTP also offers an optional mode of operation in which X.509 certificates or OpenPGP-formatted keys are employed to counter MITM attacks. An X.509 certificate used with ZRTP might be self-signed, which could enable pseudonymous keying, or it might be issued in PKI context, which would support authenticated encryption. An OpenPGP-formatted key could offer the same services. In any case, opportunistic encryption is not the most appropriate term to describe ZRTP, based on the description in the RFC cited above.

Anonymous keying is the most accurate term to use when discussing key management capabilities of protocols such as TLS and S/MIME. Access control is not part of these security protocols and there are explicit anonymous key management mechanisms that support 1-way or 2-way anonymity (for TLS). Pseudonymous keying is the most accurate term to describe key management performed using identifiers that are pseudonyms. Most RFCs use the term “anonymous” even when the term “pseudonymous” is more accurate, as noted in the discussion of RFC 5636. This document uses these three terms in a more precise fashion, to avoid confusion and to highlight the security-relevant differences associated with each.

This document proposes using the term “opportunistic keying” (OK) to refer to key management mechanisms consistent with the goals described for countering pervasive monitoring. Since those goals have yet to be formally articulated and agreed upon, this document uses the definition offered earlier, i.e., key management that yields authenticated, encrypted communicated as the preferred outcome, but which degrades (automatically) to encrypted but unauthenticated communications. Also, if this latter state cannot be achieved, e.g., as a side effect of backwards compatibility, plaintext communications results. Authentication of encrypted communication is a function of a key management activity, not the encryption process per se. Thus is makes sense to use the term “keying” (or “key management”) instead of “encryption” when defining the desired mechanism. In the same sense, this document uses the terms anonymous keying and pseudonymous keying.

Most security experts view (two-way) authenticated encryption as the most desirable state for secured communications. Authenticated encryption allows each communicant to detect (and reject) man-in-the-middle attacks. It also seems a good match to what a user expects to be true of a communication, in the absence of attacks. Most IETF security protocols have been designed to achieve this state.

This suggests that OK should attempt to establish an encrypted session (or message transfer) based on authenticated keying, with fallback to unauthenticated (or pseudonymous) keying. Like opportunistic encryption, OK should impose no requirement for a priori, bilateral, arrangements (for realtime communication). If the target of a communication (e.g., a server) makes authenticated key

material available, that material will be used to establish one or more traffic keys that are authenticated. (These keys may offer 1-way or 2-way authentication, depending on details of the protocol context in which OK is employed.) If the key material provided by the target is pseudonymous, the traffic keys may be pseudonymous, or may offer 1-way (client or initiator) authenticated keying. If no key material for a target is available to the initiator of a communication, a traffic key may be created in realtime, e.g., using Diffie-Hellman (or ECDH) key agreement. That key will not authenticate the target of the communication, but it may authenticate the client/initiator, depending on details of the OK mechanism.

For staged delivery (store-and-forward) communication, OK should yield a recipient-authenticated content encryption key (CEK), if key material provided by the recipient is authenticated. If the key material provided by the recipient is pseudonymous, the resulting CEK will be recipient-pseudonymous. In either of these cases, the sender may be anonymous, pseudonymous, or may be authenticated, depending on details of the OK mechanism (under the control of the sender). (In both cases, message integrity is provided, thus tampering with a message en route is detectable.) This document does not address scenarios in which an intended recipient of a message has not made key material available to potential senders a priori. There also is no explicit support for leap-of-faith keying in S/MIME (as offered in SSH, see below) but such support could be added.

If OK can yield either authenticated or unauthenticated encryption, there is an obvious concern that an attacker can try to force the latter outcome, even when the former might have been achieved. Also, if we attempt to “upgrade” existing commutation models that, by default, yield unencrypted communications, there is the potential that an attacker could try to manipulate any negotiation to yield a less secure outcome. It is not apparent that this attack can be avoided entirely, but one can imagine various ways to make it more difficult, and to alert a user to such attacks. For example, in the web context, port 443 vs. port 80 is an explicit request for a secure session. That same strategy could be used with OK, designating new ports for unauthenticated, but encrypted sessions, distinct from plaintext sessions. To avoid imposing unacceptable delays on users, variants of the “happy eyeballs” strategy [RFC6555] might be employed, i.e., attempting to establish authenticated and unauthenticated sessions in parallel. In the IPsec context, mechanisms already exists to accommodate OK, based on configuration parameters, as discussed in Section 3. Extending these sorts of capabilities to store-and-forward communication may be more difficult.

Pseudonymous, encrypted communication is another potential outcome of a key management exchange, as noted above. There is an obvious downside to use of pseudonymous credentials for key management as an alternative to anonymous key management. Pseudonymous credentials often employ the same syntax for identifiers as real credentials, and

thus users may be confused by the subtle distinction. Thus it is preferable to employ anonymous keying when authenticated keying is not possible, of not desired.

2. Additional Terminology

The following definitions are derived from the Internet Security Glossary [RFC4949], where applicable.

Anonymous keying — A key management technique that enables unauthenticated, communication between parties. The communication may be 1-way or 2-way anonymous. If 1-way, the initiator (client) or the target (server) may be anonymous.

Asymmetric cryptography — A type of cryptography in which the algorithms use a pair of keys (a public key and a private key) and use a different component of the pair for each of two counterpart cryptographic operations e.g., encryption and decryption. Some forms of asymmetric cryptography support key agreement for encryption (see below), others support key transport for encryption (see below) and some support only digital signatures, not encryption.

Authentication — The process of verifying a claim that a system or entity has a certain attribute value. In the IETF context, authentication typically refers to verification of an identity claim.

Client-anonymous keying - A key management technique for client/server communication in which the server is authenticated by the client, but the client does not assert its identity and thus is not authenticated by the server. This is an example of 1-way authentication.

(Data) Confidentiality - The security service that prevents information becoming available to unauthorized entities. Encryption is the security mechanism typically used to implement confidentiality.

Content encryption key (CEK) — A symmetric cryptographic key used to encrypt/decrypt the content of an S/MIME message. (Sometimes referred to as a message encryption key.)

(Data) Integrity - The security service that enables a recipient of a message or a packet to determine if the data has been modified or destroyed in an unauthorized manner.

Key agreement algorithm - A key establishment method based on asymmetric cryptography, in which a pair of entities engage in a public exchange of data (public keys and associated data), to generate the same shared secret value. (Thus both entities contribute secret values to the resulting key.) This value is later used to

create symmetric keys used for encryption and/or integrity checking.

Key transport —A key establishment method by which a secret (symmetric) key is generated by one entity and securely sent to another entity. (Thus only one entity contributes secret values to the resulting key.) Key transport may make use of either symmetric or asymmetric cryptographic algorithms.

Leap of Faith (LoF) — In a protocol, a leap of faith typically consists of accepting a claimed peer identity, without authenticating that claim, and caching a key or credential associated with the claim. Subsequent communication using the cached key/credential is secure against a MITM attack, if such an attack did not succeed during the vulnerable initial communication and if the MITM is not present for all subsequent communications.

Man-in-the-Middle attack (MITM) — A form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data to masquerade as one or more of the entities involved in a communication association. Masquerading enables the MITM to violate the confidentiality and/or the integrity of communicated data passing through it.

Opportunistic Encryption — A key management technique that enables authenticated, communication between parties, and that does not require a priori, bilateral arrangements. This term is defined only for IPsec.

Opportunistic Keying (OK) - A key management technique that attempts to establish an encrypted session (or message transfer) based on (1-way or 2-way) authenticated keying, with automated fallback to (1-way or 2-way) unauthenticated keying. If OK is unable to create an encrypted communication, e.g., because the other communicant does not support OK, unencrypted (plaintext) communication results. (See Section 1 for a more detailed discussion.)

Perfect Forward Secrecy (PFS) — For a key management protocol, the property that compromise of long-term keying material does not compromise session/traffic keys that were previously derived from or distributed using the long-term material.

Private key — The secret component of a pair of cryptographic keys used for asymmetric cryptography.

Public key — The publicly disclosed component of a pair of cryptographic keys used for asymmetric cryptography. The phrase “public key data” includes a public key and any additional parameters required to perform computation using the public key.

Pseudonymous keying — A key management technique that enables

pseudonymous communication between parties, e.g., based on use of a self-signed certificate. Pseudonymous keying may be one-way or two-way, depending on details of the key management mechanism employed.

Session - A realtime communication between entities.

Shared secret – A value derived from a key agreement algorithm and used as an input to generate a CEK or traffic encryption key.

Symmetric cryptography – A type of cryptography in which the algorithms employ the same key for encryption and decryption, and the key is not publically disclosed.

Traffic (encryption) key (TEK) – A symmetric key used to encrypt/decrypt traffic carried via an association.

3. IPsec

As noted in Section 1, the term opportunistic encryption was defined in RFC 4322, for the IPsec context. IPsec provides access control for all traffic entering or exiting a system across a nominal boundary established by an IPsec implementation. As a result, one cannot enable secure communications between a pair of IPsec-enabled system without creating appropriate database (SPD and PAD) entries in each system, a priori. Thus the term “opportunistic encryption” in the IPsec context suggests an ability to provide confidentiality for traffic without such a priori IPsec database configuration. OE IPsec preserves the ability of IPsec to implement access control for selected peers, and to employ the authentication methods already defined by IKE [RFC2409]. It offers an additional class of security service for IPsec-enabled peers, one in which traffic can be protected, by default, if both peers have published public keys (in the DNS) in advance.

The specific method for supporting opportunistic encryption proposed in RFC 4322 calls for each system that wishes to enable opportunistic encryption to publish a DNS record containing a public key for that system. (RFC 4322 called for using a TXT record to hold the public key, but later revisions to the implementation of the proposal [cite] call for use of the IPSEECKEY record, defined in [RFC4025]). If two systems publish such records, and they later attempt to communicate, then an ESP [RFC4303] security association (SA) can be established between them, despite the absence of specific SAD/PAD entries for the pair.

RFC 4322 essentially describes an extension to the IPsec SPD model, adding a new class of source/destination access control based on IP address, longest prefix match: “opportunistic tunnel.” Because the extension is expressed only in terms of IP addresses, RFC 4322 calls for storing public keys in inverse DNS trees ([RFC2371], [RFC3596]). A more recent proposal for opportunistic encryption IPsec [Wouters-

pending] calls for using the forward DNS tree (preferably with DNSSEC) for key storage. This proposal, which applies only to end system IPsec implementations, also triggers fetching an OE key based on an application's DNS access, prior to the start of a traffic flow. (RFC 4322 focused on OE in the context of security gateways [RFC4301]. A security gateway would not necessarily see a DNS access and thus could not rely on such an event to trigger a key fetch.) The revised OE proposal, based on (forward) DNS fetches, and implemented on end systems, vs. gateways, can reduce latency and thus make OE IPsec less burdensome to users.

If the public key fetched from the DNS is protected via DNSSEC [RFC4033], the quality of the authentication provided is analogous to that offered by IPsec using other methods, e.g., DANE [RFC6698]. RFC 4322 also allowed use of public keys acquired from the DNS without DNSSEC protection, as a matter of local policy (possibly a per-destination local policy).

RFC 5386 [RFC5386], "Better-Than-Nothing Security: An Unauthenticated Mode of IPsec" (BTNS) described how to provide PAE in the IPsec context. Like OE IPsec, BTNS preserves the ability of IPsec to implement access control for selected peers, and to employ the authentication methods already defined by IKEv1 [RFC2409] and IKEv2 [RFC5996]. It offers an additional class of security service for IPsec-enabled peers, by adding extensions to the SPD and PAD. In this sense, BTNS defines a two-way anonymous keying service for IPsec, complementing the opportunistic encryption service defined by RFC 4322.

RFC 5386 observed that use of anonymous keying makes communication potentially vulnerable to main-in-the-middle (MITM) attacks, as noted earlier in RFC 4322. The BTNS RFC suggests using higher layer mechanisms to detect such attacks, e.g., connection latching via GSS-API [RFC2743] and channel binding as per [RFC5386]. RFC 5386 noted that the BTNS working group was planning to define a "leap of faith" mechanism (ala SSH [RFC4251]) to reduce the window of vulnerability associated with MITM attacks. However the WG did not produce an RFC addressing this topic. BTNS seems very close to the OK goals described in Section 2, but there is no evidence that BTNS has been implemented, much less deployed.

> RECOMMENDATION: BTNS is a very close match to the OK definition provided in Section 1. If the IETF agrees with that definition, BTNS should be revisited and revised, if necessary, with the goal of encouraging widespread implementation and use.

4. TLS & DTLS

Anonymous key exchange (hence anonymous keying) for TLS was described in [RFC2246], [RFC4346], and [RFC5246]. These documents provide a very accurate characterization of the security mechanism, i.e., key

agreement can be performed without authentication of either client or server. (Authenticated key agreement also is supported in TLS.) This terminology may have been adopted because TLS typically is used to authenticate the server, and rarely is used to authenticate the client (e.g., based on use of public key certificates and RSA). In practice, the common (essentially default) use of TLS supports 1-way anonymous keying. A user typically authenticates to a server based on an application mechanism, e.g., passwords. By adopting an explicit mechanism for anonymous keying, TLS avoids confusion between authenticated and unauthenticated sessions.

None of the TLS RFCs explicitly allows use of a self-signed certificate for client authentication. These RFCs also do not describe use of self-signed certificates for server authentication, except for so-called “root certificates” (more formally, “trust anchors” [RFC5280]). There is one obscure comment buried in Appendix A of [RFC5246] that might be interpreted to refer to use of self-signed certificates:

> Note that using non-anonymous key exchange without actually verifying the key exchange is essentially equivalent to anonymous key exchange, and the same precautions apply.

This comment appears in an appendix describing cipher suites, immediately after an enumeration of anonymous cipher suites. The phrase “without verifying the key exchange” is ambiguous. If it refers to not performing certificate validation (terminating with a trust anchor as per [RFC5280]) then it might be an allusion to use of self-signed certificates for anonymous keying. Using the terminology established in this document, this would be considered pseudonymous keying.

Irrespective of the lack of explicit support for self-signed certificates in TLS, of such certificates to represent a server is common, especially when TLS is used with protocols other than HTTPS. Such use is consistent with [RFC6818].

Anonymous keying for TLS is defined explicitly as being based on Diffie-Hellman (or, as per [RFC4492] ECDH) key agreement. (Appendix F of [RFC5246] describes how anonymous RSA key transport could be implemented, using ephemeral RSA keys, but no cipher suites for that mode of operation are defined.) A client can indicate that it wants to create an anonymous session, by specifying only cipher suites of that type (e.g., TLS_DH_anon_WITH_AES_128_CBC_SHA256) in the Client Hello message. If acceptable to the server, the server replies with a Server Key Exchange message (as opposed to a Server Certificate message) that specifies the Diffie-Hellman (ECDH) parameters to be employed. This use of Diffie-Hellman (or ECDH) provides perfect forward secrecy. (If self-signed certificates were used to offer an alternative form of anonymous keying, PFS might not result.)

None of the TLS RFCs mandate support anonymous cipher suites, so compliant servers and clients need not support these capabilities. Nonetheless, support for several of these cipher suites appears to be common.

This approach to specifying Diffie-Hellman parameters differs from IKE. IKE uses IANA-registered sets of such parameters, rather than passing them in a protocol negotiation. For Diffie-Hellman the use of server-specific parameters does not seem to pose problems in terms of interoperability; software and hardware that supports Diffie-Hellman is able to deal with a broad range of groups. It might be preferable, from a security perspective, to use named groups, as IKE does, since such groups can be vetted by the community before publication. For Elliptic Curve cryptography, hardware and software support for arbitrary curves may not be common. TLS added two extensions for elliptic curve cryptography: Supported Elliptic Curves and Supported Point Formats. These two extensions enable a client to specify the curves and point format that it supports. A server replying to a Client Hello message containing these extensions can select a curve and point format that it too supports, to ensure interoperability. The named curves and point formats are IANA-registered values. A client also can declare an ability to support arbitrary curves. In this case, the server selects the curve (and point format) and conveys its choice “verbosely”.

DANE [RFC6698] provides a mechanism for TLS entities to acquire a certificate (or a “raw” public key) based on a domain name. Use of key material acquired via DANE is thus fundamentally not anonymous -- the entity being authenticated must at least be identified by domain name. DANE can provide two types of information about certificates used with TLS: (1) Additional checks to be applied in certificate verification, and (2) additional trust anchors that should be used by the relying party. Information on additional checks simply provides greater assurance to the normal TLS authentication process.

DANE information that provides additional trust anchors can provide some additional means for keying that may be pseudonymous. The certificates that are asserted as trust anchors through DANE can have any content the domain holder wishes. Thus, certificate fields that are normally used for identification (Subject or Subject Alternative Name) need not be checked against the DNS name for the DANE record. The only authentication provided by DANE-asserted certificates is the binding provided by DANE itself, to the domain name under which the DANE records are located. Thus a DANE-supplied trust anchor could contain a pseudonym that could be used as an ID for keying, directly or indirectly.

> RECOMMENDATION: TLS/DTLS already support anonymous encryption, but the cipher suites for this capability are not mandatory to implement. The relevant RFCs should be revised to mandate support for these cipher suites, including both Diffie-Hellman

and ECDH variants. The HTTPS RFC should be updated to reflect OK goals. Use of a different port for OK support should be considered.

5. S/MIME

S/MIME [RFC5751] makes no mention of anonymous keying, opportunistic encryption, or pseudonymous keying. There is no explicit access control context enforced by S/MIME, unlike IPsec, so the term opportunistic encryption is not directly applicable. However, S/MIME makes use of CMS [RFC5652] for message encryption, and CMS explicitly notes support for anonymous keying, in Section 6.2.2:

> "The originatorKey alternative includes the algorithm identifier and sender's key agreement public key. This alternative permits originator anonymity since the public key is not certified."

Thus S/MIME supports anonymous keying, with respect to a sender's identity, in the context of key management. There also is no requirement that an S/MIME message be digitally signed; one may send an "Enveloped-Only" message (Section 3.3 of [RFC5751]). (This is reiterated in Section 2.4.2, which notes that a sender is not required to digitally sign a message.) Thus, unauthenticated, encrypted message transmission is inherently supported by S/MIME.

S/MIME also defines "triple-wrapped" messages [RFC2634]. If a sender employs triple wrapping, the outer layers of the message could conceal the sender's identity, even from intermediate mail relays, but provide 1-way authentication for the ultimate recipient. (The triple wrapping facility also enables access controls to be imposed by the innermost S/MIME headers.)

The e-mail address of the sender of a message is part of the RFC 822 format [RFC822], so anonymous keying for the sender requires use of a mailbox identifier that is not (obviously) linked to the "real world" persona of the sender. This requirement that is trivial to achieve given the large number of mailbox providers that allow users to select arbitrary names. Use of a pseudonymous mailbox ID does not make S/MIME an example of pseudonymous keying, since the key management yields anonymous keying. (RFC 5750 does not require that the sender's e-mail address appear in the certificate, but many S/MIME implementations require its presence to facilitate locating the sender's certificate. Thus a sender might have to resort to pseudonymous keying to achieve interoperability with such clients.)

A sender in S/MIME could choose to employ a self-signed certificate. This would support message signing, distinct from the envelope-only S/MIME option described above, and could be viewed as consistent with a LoF key management model. If the identity asserted in the certificate is a pseudonym, this would be considered pseudonymous encryption for S/MIME. The certificate handling specification for

S/MIME [RFC5750] does not preclude acceptance of self-signed certificates, but it does restrict their use to representing CAs, and it warns of the dangers of accepting such certificates. Thus, strict compliance with [RFC5750] (and [RFC5280]) would require a sender to generate a self-signed CA certificate and issue an end-entity certificate under that CA certificate, for use with S/MIME. This does not represent a significant technical hurdle, but we suspect that most S/MIME implementations will accept self-signed EE certificates, which is consistent with [RFC6818].

For a recipient, achieving anonymous keying is slightly different. S/MIME operates in a staged delivery context, thus each recipient of an encrypted message must make available public key data for a sender to use for key transport or key agreement (see Section X.X below). In the nominal case a recipient would publish its public key data in a directory (e.g., LDAP [RFC4511]), and a sender would retrieve this data to enable encryption of a message key. (The message, or content, encryption, key, is used to encrypt the message content.) The public key of the target (recipient) need not be certified. However, this form of key distribution is rarely used for S/MIME in the public Internet (vs. enterprise) context. Publishing e-mail addresses in a generally-accessible directory is perceived as facilitating spam, and thus this public key distribution approach is discouraged. It would be possible to post a pseudonymous certificate in a repository, but it's not clear that this would be useful for most senders. There is also the possibility that an adversary could monitor repository accesses in an effort to identify potential recipients (and senders).

An alternative key distribution option is for a sender to receive public key data for a recipient as a result of a signed, but not encrypted, message from the recipient. S/MIME calls for implementations to cache capabilities information about senders (Section 2.7.2 of [RFC5751]), to facilitate this form of inband cryptographic data transfer. This represents an alternative way for a prospective recipient to convey public key info. However, this procedure is at odds with the notion of opportunistic encryption and opportunistic keying, as it calls for a priori, per-peer configuration of data to enable later encrypted communication. If the public key data were conveyed in a fashion that does not authenticate the sender, then this would enable later, anonymous keying. However, CMS and S/MIME, make no special provisions for such public key data transfers. A prospective recipient could employ a self-signed certificate to sign a message, and thus convey key material in a way that supports pseudonymous keying. However, this creates an obvious vulnerability that might be worse than retrieving a public key from an untrusted repository.

> RECOMMENDATION: OK support in S/MIME represents a significant challenge. Distribution of recipient key material via a directory systems seems unlikely to be viable, because of concerns about SPAM. It is not clear that most users would want

to receive encrypted e-mail that does not (securely) identify the sender, although the “triple-wrapping” facility of S/MIME could mitigate this concern. Enterprise environments rely on content filtering to reject SPAM and malicious attachments, so true end-to-end encryption is likely to not be acceptable in those contexts. Also, use of “web mail” interfaces precludes use of S/MIME on an end-to-end basis, raising a different set of concerns.

6. SSH

SSH, as described in [RFC4251], is “a protocol for secure remote login and other secure network services over an insecure network.” The SSH transport protocol [RFC4253] typically operates over TCP, providing a tunnel that offers confidentiality and connection-oriented integrity; it also may preform compression.

SSH makes use of asymmetric cryptography to distribute symmetric keys for the encrypted (and integrity-protected) tunnel between a client and a server. The tunnel protocol mandates server authentication, but does not provide client authentication. Thus the tunnel protocol provides client-anonymous keying.

The SSH architecture [RFC4251] specifies an authentication protocol [RFC4252] that operates within the tunnel protocol, providing client authentication. The authentication protocol mandates support for public-key based client authentication, however, the specification states that “All implementations MUST support this method; however, not all users need to have public keys, and most local policies are not likely to require public key authentication for all users in the near future.” In practice, several authentication methods appear to be commonly employed, e.g., passwords, public keys, and SecurID [SecurID cite]. (Public keys seem to be very commonly used in internal, enterprise environments [draft-ylonen-sshkeybcp-01.txt].

The SSH tunnel protocol, like TLS and IKE, employs an algorithm negotiation handshake, initiated by the client, using IANA-registered identifiers. (A client or server also MAY “guess” which algorithm is supported by a the other side, and use that algorithm to initiate a key exchange.) The handshake determines which key management, session encryption, integrity, and compression algorithms will be employed.

SSH implementations usually rely on a leap of faith mechanism to initially convey a server’s public key to a client, but also can make use of certified keys for servers [RFC4521]. Because SSH is most often deployed in an enterprise context, not in the public Internet, the use of an LoF mechanism is a reasonable option. (There is a provision for binding a user-supplied name for a server with the public key accepted during the LoF initialization procedure. This mechanism alerts a user when the public key changes, a good security

practice.) If a server has a certified public key the CA would typically be locally managed, not a public, trusted third party CA as usually employed in browsers. An alternative to using a local CA to issue certificates, is to publish a “key fingerprint” in the DNS, protected with DNSSEC [RFC4255].

Both Diffie-Hellman key agreement and RSA-based key transport mechanism are defined for the tunnel protocol, with the former REQUIRED and the latter RECOMMENDED. Diffie-Hellman groups are specified by reference to IANA-registered identifiers [RFC4250], analogous to the way IKE defines such groups. ECDH key agreement also may be employed, as specified in [RFC5656]. The curves are specified by reference to IANA-registered identifiers.

> RECOMMENDATION: SSH appears to be used primarily in enterprise environments, and for net management by ISPs. To that extent it is not clear that support for OK is perceived as critical by the administrators for these environments.

7. VoIP

Encryption in the VoIP environment is complex, because different protocols are employed for signaling and for media, and different security mechanisms have been defined for each.

Signaling for VoIP, using the “trapezoid” model, is performed using SIP [RFC3261]. That RFC defines several approaches to providing confidentiality, integrity, and authenticity for SIP signaling. The links between a SPI entity and a SIP proxy, or between a pair of SIP proxies, can be secured using IPsec or TLS (Section 26.2.1 of [RFC3261]). The SIP specification mandates support for TLS-based encryption (with one-way or two-way authentication), by SIP proxies, registration servers and redirect servers. Support of this mechanism is RECOMMENDED for SIP UAs. Support for IPsec is optional. Previous discussions of the anonymity and authentication options for the key management mechanisms employed by TLS and IPsec are applicable here. However, it appears that neither TLS nor IPsec is commonly used to protect SIP links.

The SIPS URI scheme (Section 26.2.2 of [RFC3261]) represents an explicit request by a callee to the caller and to each SIP proxy to use TLS to protect each SIP hop. SIPS mandates use of mutual authentication via TLS, using client and server certificates. Support for SIPS also mandated in [RFC3261]. However, there no evidence to indicate that SIPS is widely used (or supported) by the SIP entities that also are required to support TLS.

The SIP specification also describes how to use S/MIME to encrypt some VoIP signaling data on an end-to-end (UA to UA) basis (Section 26.2.4) This mechanism does not offer protection for most of the

signaling data, because that data must be visible to SIP proxies, so this option is not especially relevant to encryption of such data, and attendant user (UA) anonymity. Support for S/MIME use is optional for UAs, but here too, there is no evidence to suggest that this security mechanism is widely deployed.

The encryption mechanisms mandated for VoIP signaling do not tend to offer anonymity, but they also appear to not be used in many (most?) deployments. So the lack of anonymity for these mechanisms is largely irrelevant to the current discussion.

VoIP media is secured using SRTP [RFC3711]. SRTP offers confidentiality and data integrity for VoIP media and media control protocols (RTP and RTCP [RFC3350]), with an optional anti-replay feature. SRTP relies on other protocols to perform key management. RFC 3711 identified three candidate key management techniques, none of which was available as an RFC at the time SRTP was published. Subsequently the three mechanisms were defined: KINK [RFC4430], MIKEY [RFC3830], and mechanisms described in [RFC3711]. Each of these is discussed below.

There also is an ability to transport, via SDP [RFC4566] a symmetric key for use with SRTP, using the "k=" field (Section 5.12). If this method of key management for SRTP is employed it exposes the key to every SIP proxy en route (unless the S/MIME mechanism noted above is employed). It also exposes the key to any passive wiretapper along the route, unless TLS or IPsec is employed to protect the links between SIP entities. It appears that this mechanism, which is vulnerable to MITM attacks at MTAs, is not widely used.

The SDP specification was updated to include additional mechanisms to support key management for media streams (e.g., SRTP) in [RFC4568]. This RFC defined the "crypto" attribute, to signal parameters for media streams security, e.g., symmetric encryption algorithms, integrity algorithms, and key data. The key data represents an encryption key and associated data in plaintext form, analogous to the "k=" field noted above. [RFC4568] includes session parameters labeled "UNAUTHENTICATED_SRTP" and "UNAUTHENTICATED_SRTCP" (Section 6.3.3), but these do not refer to key management for these media stream protocols.

KINK was one of the candidates cited in the SRTP specification, although it was not specifically focused on VoIP. Because KINK is based on Kerberos, it is not appropriate for the scale of the public Internet, and there is no indication that it has been used with SRTP.

MIKEY was developed as a generic key management protocol, for unicast and multicast contexts (including one-to-many and many-to-many). While not specific to the VoIP environment, the MIKEY specification includes several references to how it can be used with SDP and many

references to SRTP. In the VoIP context, MIKEY messages would be transported using SIP.

MIKEY incorporated key agreement (based on Diffie-Hellman), key transport (based on RSA) and "pre-shared" key (based on use of cached, symmetric key material) modes of operation. Support for mutual authentication via pre-shared keys and public key transport is mandatory (Sections 3.1 and 3.2). Support for Diffie-Hellman key agreement is optional (Section 3.3), but that mode of operation assumes use of certificates for authentication. There is one reference to the use of self-signed certificates (page 11), but only in the context of pre-shared keys. The MIKEY specification explicitly notes that protection of the identity of communicating parties was "not a main design goal." RFC 5410 describes how to carry MIKEY is messages in the 3GPP context, in support of broadcast/multicast messaging. This use if codified in a 3GPP security specification [TS33.2436]. Nonetheless, it is not apparent that MIKEY is widely used in the VoIP context.

The third key management method cited in [RFC3711] was later published as "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)" [RFC4567]. This specification focused on key management for SRTP, trying to improve on the vulnerable mechanisms defined in [RFC4568]. It too defines extensions to SDP to carry key management data for protecting a media stream; it is not a specification of a key management protocol per se. (For example, it provides an example of how it could be used to transport MIKEY key management data.) The SDP extensions defined here enable negotiation of a key management protocol, analogous to the negotiation of other media stream features, e.g., codecs, as well as a means to transport key management data. Because this specification does not define a specific key management scheme, it is not directly relevant to supporting OK, anonymous keying or pseudonymous keying.

The current, suggested method to provide keys for SRTP sessions is to use DTLS [RFC5763] in combination with SRTP. This combination is commonly referred to as DTLS-SRTP [RFC6347]. DTLS-SRTP uses a DTLS handshake in the media plane to establish keys which are then used with SRTP. The DTLS endpoints are mutually authenticated via certificates but these certificates may be self-signed (Section 1). Authentication is primarily provided by exchanging digests of the certificates in the SDP, thus tying the media layer authentication to the identity asserted in the signaling layer. Thus pseudonymous keying or one-way anonymous keying is supported as well as mutually-authentication encryption (Section 8.7), depending on which identities are provided via SDP. Ephemeral, self-signed certificates can be used to enable anonymous calls (Section 6.1). DTLS supports perfect forward secrecy using Diffie-Hellman and ECDH.

RECOMMENDATION: OK use needs to be reconciled with the goals of the STIR effort in the IETF. The VoIP encryption landscape is complex, with many options, few of which appear to be widely deployed. A new RFC (or set of RFCs) dealing with security for VoiP probably will be needed if OK is to be successful in this context.

8. IMAP(v4), POP3 and ACAP

The security considerations section (Section 11) of [RFC3501], IMAP (v4 rev 1) mandates support for TLS (1.0) via use of the STARTTLS command [RFC2595], thus an encryption capability is present in implementations that comply with that IMAP RFC. RFC 2595 specifies how to use TLS to protect sessions created for IMAP, POP3 and ACAP. It mandates use of TLS (1.0) with server-based authentication, based on use of a server certificate. It provides clear rules for how to authenticate the server's identity; specifically it requires an authentication check based on the server identity (as used by the client when establishing a connection to the server). The identity represented in the server certificate, presented in the server Certificate message (from the TLS handshake), must match the anticipated server identity. Rules for comparing the server identity to a certificate Subject or Subject Alternative name, including support for "wildcard" names, are provided.

The principal goal of these security mechanisms is to prevent disclosure of a password used by a client to authenticate to a mail server. Server authentication is a central feature of the protocols, and so two-way anonymous keying is not supported by IMAP (or POP3 or ACAP.) However, the use of TLS here does not provide client authentication, so these specifications do provide client-anonymous keying. (Anonymous access by clients is supported, optionally, via the ANONYMOUS SASL [RFC2245] command, but this typically limits the set of mailboxes accessible by a user, and so it does not seem generally applicable.) Pseudonymous keying for a server is not supported, based on normal certificate validation processes.

RECOMMENDATION: Server authentication is a requirement for these protocols, and client authentication is typically effected via passwords, bit via client certificates used with TLS. Thus this set of protocols do not appear to be good candidates for OK, in the general sense. Also, TLS protection of IMAP and POP3 appears to be common, so this may not be a significant vulnerability. Finally, many users access mailboxes via "web mail" interfaces, and in that context encrypted communication is just another HTTP/HTTPS example, as discussed in Section 4.

9. Acknowledgements

[TBS]

10. Security Considerations

[TBS]

11. References

[RFC822] e-mail message format

[RFC2245] Anonymous SASL

[RFC2246] TLS 1.0

[RFC2371] inverse DNS

[RFC2409] IKEv1

[RFC2595] using TLS with IMAP, POP3 and ACAP

[RFC2634] Enhanced Secruiity Services for S/MIME

[RFC2743] GSSAPI

[RFC2818] HTTPS

[RFC2821] SMTP

[RFC3207] STARTTLS for SMTP

[RFC3261] SIP

[RFC3501] IMAP

[RFC3550] RTP & RTCP

[RFC3596] DNS for IPv6

[RFC3830] MIKEY

[RFC4025] DNSSEC

[RFC4033] IPsec key record

[RFC4251] SSH architecture

[RFC4252] SSH authentication protocol

[RFC4253] SSH transport protocol

[RFC4255] SSH key fingerprints in DNS

[RFC4301] IPsec

[RFC4303] ESP

[RFC4322] OE IPsec

[RFC4364] TLS 1.1

[RFC4430] KINK

[RFC4492] ECDH for TLS

[RFC4566] SDP

[RFC4567] Key management for SDP & RTSP

[RFC4568] SDP SDES

[RFC4949] Internet Security Glossary

[RFC4511] LDAP

[RFC5056] channel binding

[RFC5246] TLS 1.2

[RFC5280] PKIX cert format

[RFC5387] BTNS

[RFC5410] MIKEY & 3GPP

[RFC5636] Traceable Anonymous Certificate

[RFC5656] Elliptic Curve crypto for SSH

[RFC5750] S/MIME certificate handling

[RFC5751] S/MIME

[RFC5652] CMS

[RFC5763] DTLS/SRTP

[RFC5996] BTNS

[RFC6189] ZRTP

[RFC6347] DTLS-SRTP

[RFC6555] Happy Eyeballs

[RFC6668] SSH