version as of 2019-10-20

hotline cryptography

A group of principals {A, B, C, …} wish to exchange messages with assurance that when A sends a message, B promptly receives it unmodified and knows it came from A. Upon this initial message receipt, B also knows that A is claiming to have sent it to C and eventually A and B expect to receive confirmation directly from C. Anyone outside the group has no technical means to forge messages or see the content of messages, only their lengths.

The principals know each other and have pairwise exchanged keys face-to-face in advance. The group may add anyone known to an existing member, but is intended to be small and fully connected. A sender can pick any subset of the group to send to, and that recipient list is part of the message.

The principals are willing to communicate, but do not totally trust each other. For example, A might lie to B when claiming that the message was sent to C, A might secretly also send to D, B might decline to acknowledge, B might publish the message, and so on. The implementation tries to give some indication of anomalies, to help principals distinguish technical failures from deliberate betrayals, but takes no extreme measures. This messaging might superficially resemble more advanced group communication protocols, but there is no Byzantine fault tolerance, no dependence on central servers or network properties; we only rely on the pairwise authenticated encrypted delivery and acknowledgement listed above.

If an adversary corrupts one of the principals C, only information sent to C should leak. Message plaintext is retained (rather than ephemeral) and may be exported from the cryptographic device, so the ciphers should preferably protect against known and chosen plaintext / ciphertext but do not need to provide forward secrecy. The adversary is assumed to have active control of the network and hence can prevent all message delivery by deletion or jamming, but the cryptographic design should ensure that individual messages cannot be selectively blocked without detection. For the sake of argument, we also assume the adversary is unable to defeat emission security of the devices. The adversary is assumed to have a moderate size quantum computer.

We do not strongly conceal that A and B are communicating. In some deployments, A and B may wish to obscure how much they are communicating and the lengths of individual messages, but in this first version we do not supply background traffic generators or padding schemes; senders can do that themselves.

Messages contain no executable content and require no complicated parsing or external inclusions. We allow message bodies to be either UTF8 text without markup or a simple JPEG. Even display of a malicious message should not be confusable with header lines or non-message content.

Highest priority is put on security of the system, next is reliability, and lower is efficiency. In particular, messages are assumed small enough that there is no need to economize on

bandwidth or processing time. Primary principles toward achieving security are simplicity and open design.

Principals may be online only intermittently, so depend on well-connected relays. The relays do not participate in the message cryptography and learn no more than a network eavesdropper would. To improve reliability, identical ciphertext may be sent multiple times over distinct channels, with repetitions discarded at the relays or at the endpoints. An individual message is unidirectional, so could be sent as UDP packets or broadcast as Morse code and ciphering can be done offline. If A sends to B and to C, messages are separately encrypted and transmitted from A to B and from A to C, in arbitrary order. Keeping to the theme of simplicity and robustness, we do not try to optimize away this cost.

In some deployments, the principal has an attaché who handles the endpoint device. An acknowledgment is understood to mean that the attaché has decrypted and personally read the message, but not yet necessarily shown it to the principal.

As an optional protective measure, entirely independent of the system described above, A and B may exchange revocation passphrases. If either hears the passphrase anytime in the future, whether in an ordinary phone call or in a press announcement or whatever, then they know that the partner's system has been compromised; key material and messages should be erased to the extent feasible. These passphrases and even their existence may be known only by the principals or only by the attachés.

To clarify what we mean above and to sketch how it can be implemented, we descend now into more concrete detail.

When A meets B, she sends him:

- *principal* = globally unique four byte tag, identifying A in message recipient lists
- K_{AB} = random key for encrypting messages from A to B
- *keyID* = four byte tag indicating K_{AB} is in use
- *cipher* = index into a list of approved ciphersuites and protocol versions for K_{AB}
- routing, such as IP address of a relay or HF radio broadcast plan or courier

Also, A records that K_{AB} is for sending (only) to B, and maintains a counter N_{AB} of how many plaintext message bytes have been encrypted under the key. B records his own preferred nickname for entering and displaying *principal* on his device and then sends A his own K_{BA} and other values. The *principal* tag needs to be globally unique and never changed because it is used in stored messages. The *keyID* would not strictly need to be unique, but simplest for B if it is, so B will ask A to pick another random *keyID* in case of collision.

The recommended cipher is AES-256-GCM. To be even more specific: we use a nonce composed of a four byte random value, the four byte *keyID*, and the four byte counter N_{AB}, included as the first twelve bytes of the ciphertext. The counter is in principle already known to the recipient but if it does not match expectations, in this version of the protocol we issue a warning rather than declare synchronization lost and shut down all further communication. Re-keying is mandated well before the counter overflows, and simply overwrites the old keys; no elaborate key management. Recognizing that advances in cryptography or hardware or national pride may dictate alternatives, we allow for other ciphers but do not support version

negotiation; the sender chooses, at initial key exchange, ciphers and key lengths they consider adequate for their needs.

The plaintext begins with a four byte protocol identifier, a random number used to sanity check that compatible versions of the protocol are in use. Then comes a four byte message type code, such as UTF8 or JPEG or special codes to designate ack messages, discussed later. Following the type code is an eight byte "unix time" date in units of seconds, a four byte *principal* indicating the sender, a two byte count *n* of recipients, and *n* four byte fields indicating recipients. Binary integers are unsigned and in network byte order. All remaining bytes in the plaintext are message body.

The date represents time of composition rather than sending, so that exactly the same plaintext goes to all recipients. Senders are not expected to maintain perfect clocks, but we do expect date uniqueness among a sender's messages and monotonicity consistent with the key byte counter. Implausible dates, either violating monotonicity properties or from a clock evidently off by hours or days should be flagged to recipients, for example by displaying with ??? suffix.

We allocate a message type code "ack" for acknowledgments, where the message body consists of text lines of the form sha256: and a hex checksum of the original message plaintext. Other checksums could be allowed as compatible redundant upgrades, but given that the sender and recipients are in a good position to notice malicious collisions, sha256 seems plenty here. The recipients of such ack messages are the sender and other group recipients of the original message. We accept the n^2 cost because our groups are assumed small and again we seek simplicity and robustness. A single ack message is allowed to include checksums for multiple messages sent to that group.

Another message type code "query-ack" asks for a message acknowledgment for a specific checksum, as a way to deal with lost acks. Thus if A sends to B and C and only receives an ack from B, after a suitable delay A may send a query-ack to C, who will re-send an acknowledgment message to A and B or else a "negative-ack" message to A. We envision that as acks arrive, the display of messages will show check marks ✓ next to the recipient nicknames.

Comments are welcome! Please email Eric Grosse, grosse@gmail.com. This formulation of the problem and design arose primarily out of discussions with (in chronological order) Phil Reiner, John Gower, Donna Dodson, Marty Hellman, Dan Boneh, Fred Schneider but they do not endorse this draft and all errors are mine. In particular, I recognize that the choice of GCM is controversial because of the extreme care needed in implementation, and was leaning toward ASCON-80pq but was persuaded for now to stick with the better studied AES.