Hotline Cryptography

Eric Grosse grosse@gmail.com

version 2025-10-19 latest version will always be linked from https://n2vi.com/
This draft has only minor changes since the 2020 version.

This document makes a concrete proposal for securing messages in situations such as world leaders trying to avert war. We seek a system strong enough to resist well resourced adversaries, yet simple enough to be reviewed and adopted by skeptical international competitors. See [1] for more context on this "CATALINK" system. The main technical choices here are: only text and simple images; only symmetric ciphers; face-to-face key exchange; single-message unidirectional transmission.

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. The sender can pick any subset of the group, and that intended recipient list is part of the message. Anyone outside the group has no technical means to forge messages or see the content of messages, only lengths.

The few principals know each other, so the preferred means of key agreement is to have pairwise face-to-face meetings in advance. Any existing member may add new ones, but the group is intended to stay small and fully connected. As soon as A adds D, messages from A can include D but B and C will not be able to reply to D or receive acknowledgments until they also have established a shared key with D by having face-to-face meetings or a work-around solution as described in the implementation section below. There is no need or mechanism for revoking keys; just leave a principal off the conversation.

When A sends to B and C, A and B each expect to receive automatic acknowledgments directly from C but this may take a long time or fail completely, since C may be offline or C may even refuse to send acks. It is left to social pressure to police such behavior, though a simple timeout mechanism corrects for some loss of acks due to network failures.

The principals are willing to communicate, but do not totally trust each other. A good analogy would be diplomatic couriers carrying letters in sealed envelopes between possibly hostile countries. For example, A might lie to B when claiming that the message was sent to C, A might secretly send the message to D, B might publish the message, A might send different messages to B and C but pretend they are the same, and so on. If too much of this chaos were to happen in practice, a new design would be needed. The implementation tries to diagnose 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 and no dependence on central servers or network properties; we only rely on the pairwise authenticated encrypted delivery and the acknowledgment protocol described above.

In our threat model the adversary is assumed to have active control of the network and hence can prevent all message delivery by deletion or jamming, but would have a difficult task to selectively block individual messages without somehow stealing keys. The adversary can replay messages to consume bandwidth but this does not cause confusion at the endpoints since duplicate messages are discarded automatically. If the adversary corrupts one of the principals C, only old and new messages sent to C should leak. Given our hypothesized untrustworthy principals, it would also be possible for A to flood B with messages that pass verification but are only designed to consume storage and screen resources; it would be prudent for the endpoint software to enforce limits against this attack. The adversary is also assumed to have a moderately large quantum computer. If the adversary can execute on my machine as me, it is game over so we ask for so little from the endpoint platform that exceptional resistance to remote compromise is at least possible with enough systems engineering effort. We intend enough disk encryption and tamper-evidence that loss is likely to be detected before it can be exploited, but anyway this document assumes the adversary does not have physical access to endpoint devices or the supply chain and is unable to defeat emission security of the devices. This last might be the most unrealistic assumption; I make it not because I think the threat is unimportant but only to separate out a part of the work that I believe does not affect the cryptographic design and can already be independently solved by national technical means.

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 or health check messages to detect jamming; senders can do that themselves. We also postpone obscuration of MAC address or OS fingerprints or other identifiers of endpoint devices.

Message bodies are either 1) UTF8 text (not necessarily English) without markup or 2) a CCITT (fax) run-length-encoded grayscale images. Perhaps the list of recognized formats will grow over time, but we must guarantee safe delivery and display even of arbitrarily malicious messages. Messages must contain no executable content, require no complicated parsing or external inclusions, and avoid any misleading bitmap display. At the October 2019 workshop, we discussed the relative merits of text and audio/video. Obviously, interactivity and body language can help communication. But workshop participants felt that was mostly true within a single culture and that in the international context, carefully composed and translated text is better. Also, the audio/video system designers said that they often have difficulties initializing or debugging, so they like the idea of a backup channel using pure text. Existing systems demonstrate that it would be wise to define some pre-agreed standard messages, such as the international maritime code of signals or the NATO-Russia INCSEA coded messages, which would seek to provide core pre-translated meanings to which crisis-specific additional text could be added.

Principals may be available only intermittently, so they own and operate always-on, well-connected relay devices at the network edges. Little is assumed about the network except

the ability to deliver ciphertext from one relay to another; there are no sessions or management systems or central servers. The relays need only provide occasional delivery of ciphertext to endpoint devices, in extreme cases by physical delivery of storage media. To improve reliability, identical ciphertext is likely to be sent multiple times in distinct ways between relays. 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 overhead cost.

In some deployments, the principal has an attaché who carries and operates the endpoint cryptographic display device. An acknowledgment is understood to mean that the attaché has decrypted and personally read the message, but not necessarily shown it to the principal yet. The principal is encouraged to follow up promptly with an explicit reply as well, quoting a translation or paraphrase to avoid misunderstanding, even while internal thought and discussion are ongoing so that a full response is not immediately ready.

Highest priority is put on security, next is reliability, and lowest is efficiency. Messages are assumed small enough that there is no need to economize on bandwidth or processing time. We seek security through simplicity and open design. The principals are assumed to be from hostile countries with stringent security reviews that the hotline system must pass. Context for this hotline proposal is communication between world leaders trying to avoid war. For a purely civilian scenario, imagine a multinational enterprise incident response team in the early hours when they fear all of their existing infrastructure and credentials have been compromised. In short, the system should resist attack by very talented and well resourced adversaries.

Finally, before diving into protocol details, let's sketch how keys get exchanged, especially with the complications caused by a pandemic:

preparation

Alice and Bob agree informally to use CATALINK. They say which ID and ciphers they plan to use and where/how to meet. If any of this is in dispute, it gets negotiated ahead of time. If Alice can't travel but trusts attaché Adam, he uses a "provisional endpoint" to act as Bob for the first step of the key exchange below, receiving Alice's proposal, ready to pass along.

key exchange

To establish terminology, let's first imagine Alice and Bob meet face-to-face and connect their endpoint using a single ethernet cable, with nothing else attached. Alice "proposes" by sending a packet containing her ID, cipher, key, and routing information. Bob "acks" by simply saying so (in plaintext) on the isolated cable. Similarly, Bob proposes and Alice acks.

For the pandemic scenario, instead Alice and Bob send delegates Adam and Betty, who act similarly but using their provisional endpoint and return home.

confirmation

When Adam returns home, he and Alice again connect the real endpoint to the provisional endpoint, and Alice finally receives Bob's proposal, which Adam has already accepted on her behalf. At this point Alice has fully exchanged keys and routing information. She routes an

encrypted "accepted" message to Bob using her real endpoint, as confirmation that all is well. The provisional endpoint is erased, ready for future use.

trust

Alice is trusting Adam here to be a secure physical courier. If he ever lets the provisional endpoint out of his control, the system could be subverted. During his trip Adam can assist key exchanges with several principals (Bob, Carol, and Doug) using a single provisional endpoint, with the software keeping things straight using the previously agreed global ID for each principal.

It seems likely that additional message type codes beyond the ones specified below will be needed in these pandemic scenarios for updating routing information. This will be done carefully so that it clearly can only affect availability, not confidentiality or integrity.

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

When A meets B, she sends him:

- *principal* = 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, who 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 unique among the set of (at most a few hundred) principals 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. Although none of these values is published to the world, they may be known to untrusted principals so only K_{AB} is considered an essential secret.

The default cipher is xchacha20poly1305, so that we can avoid worrying about rekeying because of byrte counts. 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 gets to choose, at initial key exchange, ciphers and key lengths; the recipient may refuse.

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 fax 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 the same timestamp 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. No specific action needs to be taken by the recipient in this case, just extra skepticism. We're not assuming a secure NTP service, we're just adding an inexpensive GPS receiver to Broker and letting Puck confirm its clock has not drifted to far off from Broker.

We allocate a message type code "ack" for acknowledgments, where the message body consists of the four byte sender ID and the eight byte timestamp in the sent message. Such ack messages are individually sent to the sender and other group recipients of the original message. Note that, unlike in a previous version of this doc, there is no message content checksum, so that a suspicious sender could enable message watermarking per recipient. We accept the n^2 cost because our groups are assumed small and we seek simplicity and robustness.

Another message type code "query-ack" asks for a message acknowledgment for a specific message, 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. No particular guidance is provided on the timeout for sending query-ack; ten minutes would be a plausible value, perhaps with exponential backoff. A lot of these query-ack messages may accumulate for a principal that is away for an extended period, so automated processing is essential.

Prototype software may be found at `https://github.com/n2vi/hotline` and `…/puckfs`.

`puck.go` and is intended to be small and straightforward enough for direct inspection. I personally prefer point-and-click via the `acme` editor from the Plan 9 system, but the tool can also be used directly from a command line shell on the puck, which is what we call our hotline endpoint:

`hotline fetch`	download any pending messages from the network
`hotline list`	summarize all downloaded and saved messages
`hotline send < mess`	upload a new message

Messages are saved as plain text or faxes, to be viewed using your favorite editor and image viewer. Again this would be point-and-click in acme. The file `mess` could be prepared locally or drafted in the Department of State and imported to the puck; it has the form

```
to Bob
to Charlie
```

```
доверяй, но проверяй
```

We skip here the commands to initialize and the network implementation, since in our prototype uses a "broker" placeholder allowing us to test pucks while waiting on a prototype resilient network implementation.

We need a work-around for how a new principal joins an existing network, especially in these pandemic times when physical visits may be difficult. Suppose that Alice, Bob, and Charlie are in an existing fully-bilaterally-connected network. Suppose next that Alice meets Dan directly and they arrange keys and are successfully communicating. Now Alice wants to add Dan to the larger network. Alice sends a message to Bob saying

```
hotline introduction 123456789 Dan
```

and Bob executes that, thereby adding a *principal* entry in Bob's puck-local database. But Bob does not have a shared secret with Dan yet, and Alice ought not participate in setting one. Instead, Bob and Dan agree remotely on some random string using existing channels that they trust, perhaps couriers augmented by the Signal app (before war breaks out so while existing networks are still operational) and then Dan executes locally on his puck:

```
hotline rekey Bob 'random string'
```

and similarly for Bob. Now Bob and Dan can exchange messages. Repeat this for Charlie.

I close with answers to Frequently Asked Questions:

In some diplomatic situations, it matters immensely who sends the first message or places the call. Reflecting existing practice, our protocol does not attempt to fudge this but rather assumes it is handled by lower officials or by "Track 2" intermediaries. Thus it is expected that there will be several parallel hotlines or groups of principals, each group treated independently. There are conceivable technical solutions involving distributed asynchronous "ready-to-receive" protocols, but after discussion it was decided that those introduce undesirable diplomatic uncertainty and complexity. Maybe it suffices if A and B keep a light conversation going so the first serious message is not such a dramatic event?

There is no automatic key renegotiation. If the byte-limit is reached or if a compromise is signaled somehow, principals have to re-establish keys from scratch in a new face-to-face meeting. That is harsh, but years of experience hardening an important identity system (see `landing.google.com/advancedprotection`) taught me account recovery is the weak link. My expectation is that the attaché operating the hotline endpoint for the principal, "carrying the football" in NC3 parlance, is highly professional and does not carelessly lose keys. The byte-limit is so large and messages so small that regular updates should never be needed. Just re-key every couple years when you meet up at the UN or G7 or whatever. Frankly, my advice would be to set up more than one secure endpoint, each with its own keys, and keep the backup in a safe; only get it out for testing and in extremis. Or instead of having just one attaché

carrying the football, have two attachés carrying independent footballs. If one is on vacation, the other will do. If one is compromised, tell all correspondents and only use the backup. Your correspondents may find it a hassle that you have more than one key, although mostly they will just be replying to a message and so I hope it is manageable. Since an Alice-Bob pairing may have more than one key, Alice may use nicknames Bob-p1 and Bob-p2. Anyway, let's not complicate the story or initial deployment with this wrinkle. Nothing in our design prevents a prudent principal from adopting the practice in later years when desired.

There is also no cryptographic key commitment, since in the hotline context going back to the Cuban missile crisis plausible deniability is generally regarded as a feature rather than a bug. But perhaps re-keying becomes harder when couriers are involved. This needs more thought.

Sender and receiver see the same message content, as a consequence of authenticated encryption, but cannot prove this to anyone else. This ability to repudiate past messages is deliberate and could be useful but is not a primary functional requirement.

If only the pucks hold keys, there is little to prevent an adversary from flooding the network with junk messages or packets. I am considering a keyed hash as a defensive measure, with the keys being part of the "routing information" that Alice and Bob exchange.

As an optional protective measure, A and B may exchange revocation passphrases. If either hears such a distress 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. Even the existence of these passphrases may be known only by the principals or only by the attachés. This proposal is highly tentative, as it could well lead to more harm than good through untimely catastrophic failure of the system or risk to personnel. Anyone with practical experience is especially requested to give us advice.

From our understanding of how hotlines are used, message plaintext is assumed to be retained rather than ephemeral and may be exported from the cryptographic device. Therefore the authenticated encryption should protect against known and chosen plaintext / ciphertext but does not need to provide forward secrecy. It would be possible for A and B to hash K_{AB} after each message, or use a symmetric-key ratchet, but I did not see a need. Double ratchet would not be feasible because the assumed network has no fine-grain interaction, only one-way messages with time delay.

The keyID is included in the nonce to enable endpoints to conserve bandwidth by fetching only messages intended for them from a relay that may hold messages for other endpoints as well. In principle, an endpoint could download everything and try all keys. The keyID is only meaningful to A and B and anyone doing enough traffic analysis to deduce who A and B are. The intent is that only one keyID is valid at a time for A and B. If they want to have multiple keys for redundancy, it is wiser to do that at the level of fully independent endpoint devices and

independent principal tags as well. It would be possible to change the app to do more complicated key management without changing the protocol, but that invites operator error. At initial and subsequent key exchange, new key material overwrites old; obsolete keyID could appear and be ignored if old messages get replayed.

Ack messages do not themselves get ack'd. If A sends a message to B and C but never hears back from C, then A will send a query-ack. If A does get the ack from C but B does not, there is no correction mechanism. This seemed good enough, but ought to be part of the prototype trial with real principals in made-up scenarios to find out how much they care. The protocol does not include an automatic negative-ack mechanism, though the byte counter does make it technically possible for the endpoints to highlight some failures and let the operators clarify manually. In this, I may be showing an American bias toward two-party conversations. Include Europeans in the prototype trial, since they may have more nuanced views about multi-party play.

When all is working normally, at the time B first views a message sent to many principals, some recipients will have checkmarks and some not. If B views the message again tomorrow, normally almost all will have checkmarks, but perhaps not if some are off on vacation; that is taken as a mild signal that not everyone is actively in the conversation. Only if there seems to be a persistent problem would B raise the matter with A or with C or with technical advisors. It frequently seems to happen in the real world of email that months later C claims to A or B that he never saw the message. Acks help with this. Or maybe not; deniability does have its advantages.

Message acknowledgment is well known to be effectively unsolvable in the network threat model. The acks here are hoped to be good enough to be useful to principals without introducing excessive complexity. Only testing will reveal if this hope is justified.

One could fairly criticize that I have documented above the easiest parts of the hotline. An associated document "hotline systems architecture" should expand upon the plan for very secure endpoint hardware (the "puck") and the plan for the relay device (the "broker") and redundant networking. By "very secure" here I mean much less susceptible to being compromised than any of the other computers you have. As an extremely tentative hint: imagine the puck has a licensed simple RISC-V processor, maybe borrowing from OpenTitan, built at an older-tech trusted fab. No WiFi or USB or cameras or microphones, just keyboard connector and HDMI and the ethernet that comes with that. Key exchange and communication between the puck and broker is over CAT5. The puck will boot over eSPI from Oreboot firmware and run something like seL4 with small components on top plus a minimal network stack, all written in Go. My inspiration is Ken Thompson's story "I had a file system, all I needed to add was an editor, a compiler, and a kernel." I think of it a bit like Qubes, only using capabilities instead of virtual machines for the isolation of network stack. There is no multi-user sharing of the puck, so side-channel protection is not a prime concern. Controlled sharing off-puck of message text is important to keep the operator from working around security; I'm going to try it with Upspin. In deployment, the puck should be so small and cheap that you can update it by grinding it up and

getting a new one. The brokers can be more ordinary computers but using a mix of multi-homed IP and spread spectrum digital mode terrestrial radio and more to get the desired network resilience. There also needs to be a design for the practical operation, testing, training, and deployment.

With existing certified-secure technology, it sometimes happens that government officials want to securely communicate with partners that they can't totally trust and their best option is to use obsolete crypto devices from years ago, still secure enough for the purpose, a bit clunky, but not a catastrophic loss if one gets into enemy hands. Because of the open source and comparatively low cost we target, one can imagine a variant of our hotline system being able to meet that need, though wise to revisit key exchange and lack of forward secrecy.

This formulation of the problem and design arose primarily out of discussions with (in chronological order) Phil Reiner, Peter Hayes, John Gower, Donna Dodson, Marty Hellman, Dan Boneh, Fred Schneider, an MIT LL team led by David A. Wilson, and Ulfar Erlingsson, but they do not endorse this draft and all errors are mine. In particular, the choice of GCM is controversial because of extreme care needed in implementation, and I was leaning toward ASCON-80pq but was persuaded to stick with the better studied AES. I also wish to thank Deborah Gordon, Ron Minnich, Doug Randall, Devabhaktuni Srikrishna, Adam Wick and anonymous participants in the Tech4GS / Nautilus / Stanley / CISAC workshop, October 21-22 2019.

[1] Nautilus Institute, Stanley Center For Peace and Security, and Technology for Global Security, “Last Chance: Communicating at the Nuclear Brink, Scenarios and Solutions Workshop, Synthesis Report”, NAPSNet Special Reports, May 23, 2020, https://www.tech4gs.org/uploads/1/1/1/5/111521085/last_chance_final_report-1__1_.pdf