Cebolla: Pragmatic IP Anonymity

Zach Brown
zab@zabbo.net

Abstract

Cebolla is an intersection of cryptographic mix networks and the environment of the public Internet. Most of the history of cryptographic mix networks lies in academic attempts to provide anonymity of various sorts to the users of the network. While based on strong cryptographic principles, most attempts have failed to address properties of the public network and the reasonable expectations of most of its users. Cebolla attempts to address this gulf between the interesting research aspects of IP level anonymity and the operational expectations of most uses of the IP network.

1 Introduction

The core concept of providing anonymity of commendations through intermediary relays dates back to the early days of the public network. As initially described by Chaum for email [1], anonymity of the sender can be achieved by sending the message to an agent who encapsulates the email and relays it to a second agent, who relays it to a third, who finally delivers the message. Imagining the communication as a conventional paper letter would conjure an image of each agent opening their letter to discover another letter destined for the next agent. The final agent sees the proper letter destined to the recipient. The response travels in the reverse direction, with each agent putting the incoming letter into a new envelope and addressing it to the previous agent. The sender, upon receiving this large envelope, opens as many layers of envelopes as there were intermediate agents to find the original response.

This anonymizing theory can easily be applied to networks when forwarding instructions are included with each datagram. The included instructions increase the size of the datagrams and verifying the instructions can be very expensive. The common solution to these problems is to negotiate and verify the instructions and require that datagrams reference this existing negotiated state. In Cebolla, this negotiated state is asymmetrical. The initiating sender of all the messages negotiates individual instructions with all the forwarding agents. Each forwarding agent only negotiates state with its immediate neighbours in the path.

Cebolla builds on many previous implementations of anonymizing mix networks:

Wei Dai describes an asymmetric anonymizing network, dubbed PipeNet [2]. While very resilient to attack, it is infeasible to run over the public network. Constant cover traffic makes link usage inefficient and prohibitively expensive. Random path selection punishes users with moderate threat expectations that could tolerate narrowing their traffic to topologically close networks. Finally, rampant frame reordering would confuse popular networking protocols. It has never been publicly implemented.

D. Goldschlag and company at the Navy Research Lab have done much work on Onion Routing [3]. While providing much of the ground work in the field, the implementation is not publicly available, and is covered by US patents.

Zero Knowledge productized a mix network with their Freedom product line. The productized nature of the network motivated Zero Knowledge to remain some amount of centralized control of the network, which turned off some potential users. It was never publicly documented fully, nor were comprehensive sources made available, which prevented any third-party implementations of the protocols to conceivably increase the user-base. It has since been discontinued.

Mike Freedman and company push the envelope by introducing a very scalable peer-to-peer anonymizing network with Tarzan [4]. As it happened, Cebolla and Tarzan were developed at about the same time, with different objectives.

Xor-Trees [5] take the concept to an extreme by describing a network with a fully utilized mesh of dedicated links and synchronized key material generators that can be used to mask both the source and destination of messages between command and control centers. The requirements for fully utilized links and synchronized key material make it infeasible for use on the public network.

Cebolla is an attempt to gel these efforts into an implementation that can be readily used by a group of people on the Internet to efficiently protect their communications. The remainder of the paper will focus on defining the environment that Cebolla considers reasonable, and the methodology behind the implementation.

2 Overview

Cebolla is a unix daemon that maintains UDP connections to a set of peers. Many of these peers connecting together builds an overlay network. Through these UDP connections, called links, peers are able to exchange messages which maintain crypto state, discover the topology of the network, negotiate tunnels on behalf of nodes, and transit encapsulated frames through these tunnels.

Tunnels are the construct that allow messages to be forwarded in a way that masks the identity of the sender. A tunnel is a set of forwarding rules that a client gives to nodes that make up the path of the tunnel. The client also shares keys with each node in the path of the tunnel. Clients and servers run the same software; it is initiating the tunnel that makes us call a node a client.

Like other IP tunnels, Cebolla tunnels have networking devices on the nodes at either end of the tunnel. When a frame is routed into a device, the frame is encapsulated and sent down the tunnel. At the other end, frames exit the tunnel and are received by the device at the end of the tunnel.

The steps performed by a client in a typical Cebolla session might look something like:

- A node in the network is discovered. From this node, the client receives a list of all the nodes in the network.
- The client decides on nodes in the network that a tunnel should include.
- The client establishes a UDP link with the first node in the tunnel.
- With this first node, the client negotiates the first part of a tunnel.
- Through the first part of the tunnel, the client negotiates the second part of the tunnel with the second hop. And so on, until the client only has one hop left.

- Through the tunnel, the client finalizes the tunnel.
- By routing through a local device, the client sends frames down the tunnel.
- Headers on the encapsulated frame tell each hop which tunnel to go down, and the headers are re-written at each hop. Crypto may be performed at each hop, if the client so desired.
- As the frame reaches the final hop, it exits the tunnel and is forwarded out the Internet as a normal IP frame.

3 Threat Model

When describing the threats that Cebolla tries to address, we'll adopt some names for roles that are played out on the network:

- **Alice** – the initiator of the communication, who wishes to remand anonymous.
- **Bob** – the intended recipient of the Alice's communication.
- **Neville** – A corrupt node in the Cebolla path who has honestly participated in the protocol with Alice, but who is trying to leverage that to monitor communications.
- **Patrick** – An attacker who controls the flow of encapsulated frames between Alice and Bob, who can conceivably alter them as they pass.
- **Kiddie** – An attacker with connectivity to the same network as the mesh, but with no control of the path that messages take between Alice and Bob.
- **Smith** – An attacker who is able to monitor communications in the mesh at multiple points and perform deep analysis in real-time.

With these participants in mind, Cebolla attempts to make the following guarantees:

- Alice should be able to determine that she is actually communicating with Bob.
- A Neville working alone should not be able to determine the identities of both Alice and Bob.
- Only Alice and Bob should have access to the actual contents of messages.
- Kiddie should not be allowed to degrade Alice and Bob's communication through trivial a expenditure of resources.

Cebolla also makes the following explicit admissions about its lack of privacy guarantees, as well:

- Two or more Nevilles at the right points in the path may collude, with the help of traffic analysis, to discover many things – the identity of both Alice and Bob, the path that their communication takes, and in unbelievably specific circumstances, even the contents of their communication.
- Patrick may sever communication between Alice and Bob at any time.
- Neville can associate frames that probe the edge with streams he transits, possibly giving rise to the ability to find the node at the edge that terminates a particular communication.
- Smith must to be assumed to be the superset of all possible Nevilles – always knowing which Bobs all Alices are currently in communication with.

4 Secret Negotiation

The Cebolla protocols make heavy use of a four-step secret negotiation that builds shared secrets and negotiates optional features. The message exchange is inspired by Photuris [6]. Following the asymmetric nature of Cebolla's anonymity guarantee, more emphasis is given to protecting the information sent by the client initiating a negotiation than by the server responding to it.

4.1 negotiation phases

The negotiation is split up into four phases:

- **initiation**. The initiator sends an initial negotiation request to the server. The request contains a large random initiator ID and lists of the authentication and shared-secret negotiation schemes that the initiator is willing to use during the negotiation. The entirety of the request is sent in the clear and is readable to those who can monitor the channel it is sent over.
- **response**. The responder parses the request and prepares a response packet. The initiator's random ID is echoed back in the response, and the server provides a responder ID as well. The pairing of these, between the responder and initiator, uniquely defines this negotiation instance. The server parses the lists of offered authentication and shared-secret schemes, and chooses one of each to use for the negotiation. Should it not find any suitable, it can return errors. The response includes authentication data and the responder's half of the secret negotiation, as defined by their respective chosen schemes. The entire response is sent in the clear, but the server appends a signature work-a-like, which the initiator may validate using the chosen authentication scheme.
- **configuration request**. The initiator validates the responder's authentication and prepares a packet containing the IDs that identify the exchange. The initiator appends its half of the shared-secret negotiation to the packet, then combines its half with the responder's half in the incoming packet to calculate the shared-secret. From this secret it derives keys that are used to encrypt the initiator's authentication data and a list of negotiable options that are appended to the packet. The initiator then signs its half of the shared-secret and the encrypted data, appending the clear-text signature material to outgoing packet.
- **configuration acknowledgment**. The responder prepares the final packet in the exchange by parsing the incoming configuration request. After the responder verifies the initiator's signature, it combines the halves of the shared secret and decrypts the initiator's authentication data and option list. The responder choses options from the incoming list and puts them in a list in the outgoing packet. The acknowledgment packet is encrypted with the shared secret.

4.2 negotiation state

An important aspect of the negotiation is that the responder does not maintain state for a negotiation until the configuration request has been successfully parsed. The initiator is responsible for issuing retransmissions until it gives up or the negotiation ends in success or error.

The responder must assume that the initiator will receive the sent configuration acknowledgment because it is the last packet in the exchange. The responder must be careful to deal with the possibility of receiving a retransmitted configuration request from the initiator when the acknowledgment is lost in transit.

4.3 negotiation verification

While the initiator ID is simply a large stream of random bytes, the responder ID is built to provide similar functionality for the responder as syn-cookies [7] do for the TCP handshake.

The responder maintains two private secrets that are alternately replaced at regular intervals. The responder ID is calculated by taking a hash of the most recent private secret and the address of the initiator in the medium of the negotiation. An incoming configuration request must have a responder ID that matches the hash of one of the private secrets and the initiator's address for the responder to be sure that it sent a response to this initiator within the interval that the secrets are updated in.

4.4 negotiation resource consumption

Denial of Service are said to occur when an attack drains resources to the point of excluding others from using those resources. Cebolla doesn't address attacks that exhaust incoming bandwidth because they are best addressed upstream, out of Cebolla's reach.

An attacker wishing to exhaust the CPU resources of the responder is more troubling. The attack comes when an attacker overloads the responder with negotiation packets that look valid based on the responder ID. The responder ID's validity is tied to the source IP of the packets. If the attacker generates the stream of packets through legitimate participation in the protocol the responder can limit the attacker's CPU use based on the IP. Limiting can also be used if the attacker resends an infinite stream of identical packets, all of which must still have a valid IP address to pass the responder ID test. An attacker in the path of regular negotiation traffic can resend packets that it observes, throwing disrupting all negotiation on the path.

It would be generous to describe this protection as incomplete. An attacker is still able to use significant resources on the responder through little effort. Mechanisms like hashcash [8] should be employed to require significant expenditure on the part of the attacker to proceed with the negotiation and convince the responder to spend CPU cycles.

5 Links

Links are the backbone of the Cebolla mesh. All communications between nodes, which include clients, occur over these links. Links use symmetric ciphers to guarantee confidentiality of communications and employ message digests to ensure that communications haven't been tampered with.

Link negotiation occurs between nodes over UDP. Link state is associated with a neighbour's source IP address and UDP port. This association builds the concept of a unique link.

The IP address and UDP port of the responder are assumed to be reachable by all clients. The address of the initiator is never used in the protocol. Initiators may build links from behind routers performing NAT without harm. As is expected, the NAT changing its IP and port mapping will confuse the association of that IP-port pair with its link state.

The primary result of the negotiation is a set of dual transmit and receive keys that the partners of the link use to encrypt and verify frames sent to each other. Separate sets of transmit and receive keys are used to prevent attackers from reflecting frames sent from a node back to the node itself.

5.1 link encapsulation

All messages between link partners are described by a link header. It contains a sequence number, a message type, some flags, and a generic ID field that is used by certain message types. The sequence number, described later, protects attackers from replaying valid frames. No flags are currently defined, and the type is obviously used to decide what to do with the frame.

5.2 dueling link headers

This link header is kept at the size of the block cipher used in the link to enable nodes in the middle of the path to save packet space and CPU time under the right threat assumptions. A client may decide that its traffic is adequately protected by a single-layer full-frame encryption and a MAC check only at the end of the path. The routing process in all transit nodes then simply involves decrypting the header, rewriting the ID, encrypting the header, and forwarding the packet.

A single decryption of the block the header resides in wouldn't be enough to give confidence in the resulting header – it could have been modified in transit. Instead of spending bytes and CPU time on a MAC covering the header, we instead maintain a second key that encrypts and decrypts a second copy of the link header. The recipient decrypts the two copies with the two keys, and if they match it has high confidence that either header has not been modified.

6 Tunnel Negotiation

Cebolla builds up tunnels in an iterative process. The first stage is done between the client and its immediate neighbour who it already has negotiated a link with. A tunnel negotiation builds up similar cryptographic state as is built up in a link negotiation. It also assigns tunnel IDs to each participant. The negotiation can include assigning an IP address to the initiator's endpoint on the final hop negotiation. The negotiation of intermediate hops includes a negotiation parameter that specifies the IP address of the next hop to be negotiated.

Tunnel IDs are used by pairs of nodes to associate frames with a tunnel. Each node has a local ID for a hop that connects to another node. This local ID is uniquely generated by each node and transmitted to the other node during the negotiation. When sending frames down a tunnel the sender uses the remote ID to specify the tunnel to the receiver.

If a multi-hop tunnel is being negotiated, the initiator will include an option in the negotiation that will specify the next hop in the path. The negotiated tunnel is not yet ready to be used with real encapsulated frames. The responder in the negotiation will establish tunnel state and mark it as embryonic and store the next hop. The initiator negotiates an additional hop through the embryonic tunnel by building messages intended for the additional hop. The initiator sends these message down the embryonic tunnel as encapsulated negotiation packets. The embryonic tunnel will unencapsulate the messages and forward them over a link that is established to the additional hop.

This process can be repeated for as many hops as the initiator wishes to build.

As of this writing there are no provisions to stop an initiator to a long time building a tunnel that passes through nodes in the network many, many, times. Such a tunnel allows an initiator to send a single packet down the tunnel, resulting in excessive bandwidth and CPU expenditure by the network.

Preventing this behaviour with a simple time-to-live packet header, as used in IP, would give intermediate nodes information about the length of tunnels. This knowledge can be combined with knowledge of the network graph and measurements of streams to gain a very educated guess as to the actual nodes that make up the tunnel.

6.1 tunnel encapsulation

Tunnel headers communicate details of the encapsulated frame between the initiator and the final hop of the tunnel. The current implementation only contains a type field, which is limited to specifying encapsulated IPv4 frames, an unused flags field, and a sequence number.

The level of protection offered by the tunnel is under full control of the negotiator. Each hop negotiation specifies whether block ciphers or MAC digests are applied to payloads passing through that hop.

7 Keying

Cebolla relies heavily on symmetric ciphers to speed up encryption and decryption. Link and tunnel negotiation both build up a shared secret associated with that link or tunnel. Symmetric keys are derived by hashing the shared secret with a known value for each line of keys that will be used. Further keys in a line are derived by hashing the existing keys with the shared secret. For example, the link payload encryption and decryption keys would differ from the dual link header keys by the known value they were hashed with.

Peers must be sure to derive their encryption and decryption keys so that they match their peers'. The initiator's encryption key must match the responder's decryption key. The current implementation achieves this by requiring the responder to swap its key sets after both peers derive their keys with the same code.

7.1 re-keying

Trust in symmetric keys diminishes the longer they are used in the wild. Key rotation, or re-keying, must be done at regular intervals to lessen the success attackers can have at crypt-analyzing the keys. The rotation must be synchronized between either ends of a resource, allowing for dropped messages, to prevent the keys from becoming out of phase.

Cebolla does this by adopting a protocol for rotating the keys that depends on minimal re-keying messages , knowledge of the role of either end in negotiating the initial resource, and feedback based on which keys succeeded in decrypting incoming frames.

The party who decides to start the re-keying protocol first is dubbed the initiator, the latecomer the responder. Both parties maintain two sets of keys for a given resource, primary and secondary. In the quiescent state the primary keys are in use and the secondary keys are undefined. When re-keying is active, the primary keys are used to send messages and to decrypt messages. Decryption is attempted with the secondary keys only when the primary fail.

- The initiator decides to start re-keying. It derives its secondary keys from the first, and sends a re-keying request to the responder.
- The responder sees the re-keying request and derives its secondary keys from the first, and swaps its primary and secondary keys. Its now primarily encrypting and decrypting with the next generation keys. It sends a dummy message, usually implemented as some form of echo request, over the medium.
- The initiator fails to decrypt the message with its primary keys, but succeeds with

the secondary keys. It takes this to mean that the responder has derived the next generation keys. The initiator swaps its primary and secondary keys, and destroys its secondary keys. It is now only using the next generation keys. It sends another dummy message over the medium.

- The responder succeeds in decrypting the message with its primary keys, and takes this to mean that the initiator has completed the re-keying and destroys its secondary keys.

This mechanism is simple to implement and lets traffic flow during the time it takes the re-keying messages to make the round trips between nodes, which may be particularly important over long, fat pipes. There is always a window during which messages will be doubly decrypted by a mismatch in the primary keys of the sender and receiver.

Initiators can cross the streams. If both parties decide to re-key at the same time, their re-key requests can cross in flight. Both will notice this when they go to process a re-keying request and find that they have initiated a re-keying request themselves. Both initiators know the role they played in negotiating the higher level resource (link or tunnel) and fall back to that role when they discover concurrent re-keying negotiation.

As with negotiation, it is the responsibility of the initiator to re-send re-key requests if it thinks that re-keying is not progressing at a reasonable pace.

8 Sequence Numbers

Cebolla uses sequence numbers in a few places to empower the receiver to discard duplicate frames. A typical advancing window approach is used, implemented almost verbatim from the one specified in RFC 2402. We'll briefly summarize.

The sender always increments the sequence number of frames it transmits. There is only one instance of a sequence number in the lifetime of the sequence, which starts at 1. The receiver maintains a window of sequence numbers that will be accepted. As sequence numbers arrive in that window they are marked off. If that sequence number arrives again its frame will be discarded. If a sequence number arrives that is past the window, the window is shifted so that the largest acceptable sequence number is that of the new arrival. This scheme is simple to implement with bitfields and can withstand reordering and large periods of packet loss on the network.

Care must be taken so that the sequence numbers do not wrap. In the case of tunnel and link payload sequence numbers, the sequence is bound to a key context. When the key contexts are cycled, the sequence is reset to 1. This can be forced when the sequence gets close to wrapping before policy would otherwise dictate that key contexts would be cycled.

9 Network Discovery and Topology Flooding

Cebolla uses a topology flooding scheme which is based on OSPF. Clients must be able to discover nodes in the mesh to communicate through. The clients may wish to make complicated decisions about which nodes to trust, and should be able to trust the information they use in making this decision.

At regular intervals, each node broadcasts a Link State Announcement to each neighbour it has an established link with. These announcements describe the node's static attributes, as well as its current connectivity information. These announcements are signed and contain a sequence number.

As a neighbour receives an announcement, it stores the announcement if it is newer than the previous announcement from that neighbour, and sends back an acknowledgment. If the announcement was new, the neighbour then sends the announcement to all its neighbours. The announcement is not forwarded to the neighbour it was just received from. Announcements are re-sent until their receipt is acknowledged.

The announcement collections of each node are sorted and served via the rsync protocol[9], which allows clients to receive deltas of the largely static information efficiently.

The mechanism can be boot strapped by nearly any out-of-band medium that can communicate IP addresses: email, web pages, DNS, etc.

9.1 topology attacks and accountability

Topology discovery raises many risks. A corrupt node could induce bad announcements into the network. A corrupt node could alter the flow of announcements it transits to the rest of the network. A corrupt node could participate honestly with the rest of the nodes in topology flooding and feed bad information only to clients.

Public key signatures bring a simple first layer of confidence to the system. Announcements can be confirmed to come from the same agent as last time, and trust can be established between that agent and any existing public key trust metric system.

The distributed publication of the connectivity announcements gives multiple views into the announcement circulation at multiple points. This lets clients audit the veracity of the announcements, possibly anonymously. Distributed trust metrics can be adopted by using the sequence numbers to check that all views of the mesh are consistent with others. The rate that sequence numbers advance can be checked to make sure that a node isn't delaying announcements. This lets a mesh permit a node's entry into the mesh without centralized admission. Low initial trust increases over time as behavioral audits confirm that the new node is honest.

The announcements can be extended so nodes can describe themselves. Clients can use parameters like software types, administrative domains, underlying network connectivity, and such, to decide which nodes to build a tunnel with.

10 Acknowledgments

Jerome Etienne should rightfully be considered an author of any document describing these protocols—when he didn't outright design them they're only minimally deviating from his work.

Phil Schwan and Mike Shaver's contributions are treasured, as always.

Special thanks are due to Dr. Adam Back, who never fails to make taking on an interesting challenge entertaining.

Ulf Moeller, Mike Freedman, Anton Siglic, Ian Goldberg, and Adam Shostack were kind enough to humour me by explaining cryptographic principles using small words.

11 Availability

Cebolla should be available under the GPL from

`http://www.zabbo.net/cebolla/`

References

[1] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM (USA)*, 24(2), 1981.

[2] Wei Dai. Pipenet. http://www.eskimo.com/˜weidai/pipenet.txt.

[3] D. Goldschlag, M. Reed, and P. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM (USA)*, 42(2):39–41, 1999.

[4] Michael J. Freedman, Emil Sit, Josh Cates, and Robert Morris. Introducing tarzan, a peer-to-peer anonymizing network layer. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, Cambridge, MA, March 2002.

[5] Shlomi Dolev and Rafail Ostrovsky. Xor-trees for efficient anonymous multicast and reception. Technical Report 98-54, 23 1998.

[6] P. Karn and W. Simpson. [rfc 2522] photuris: Session-key management protocol, March 1999.

[7] Dan Bernstein. Syn cookies. http://cr.yp.to/syncookies.html.

[8] Adam Back. Hashcash. http://www.cypherspace.org/hashcash/, May 1997.

[9] Andrew Tridgell. Efficient algorithms for sorting and synchronization. http://citeseer.nj.nec.com/tridgell99efficient.html.

Proceedings of the Ottawa Linux Symposium

June 26th–29th, 2002
Ottawa, Ontario
Canada