Received: 13 September 2024, Revised: 17 October 2024, Accepted: 18 October 20 24, Online: 04 November 2024
DOI: https://dx.doi.org/1 0.55708/js0311002

Secure Anonymous Acknowledgments in a Delay-Tolerant Network

Edoardo Biagioni
Edoardo Biagioni, University of Hawai'i at Mānoa Department of Information and Computer Sciences, Honolulu, HI 96822, USA
*Corresponding author: Edoardo Biagioni, 1680 East-West Road, Honolulu HI 96822, USA, +1-808-956-3891 esb@hawaii.edu

ABSTRACT: TCP and many other protocols use acknowledgments to provide reliable transmission of data over unreliable media. **Secure** acknowledgments offer a cryptographic guarantee that valid acknowledgments for a given message can only be issued by the intended receiver. In the context of an ad-hoc network, **anonymous** acknowledgments make it hard for an attacker to determine which device issued a particular acknowledgment. And unlike TCP, the acknowledgments described here work well even for connectionless communications. This acknowledgment mechanism assumes that message data is protected by secure encryption. The sender of a data message includes in the encrypted part of the message a randomly-generated acknowledgment. Only the intended receiver can decrypt the message and issue the acknowledgment. The acknowledgment is issued by sending it out to its peers, who will forward it until it reaches the sender of the data being acknowledged. Such randomly-generated acknowledgments in no way identify senders and receivers, providing a degree of anonymity. This paper describes the use of such acknowledgments in both ad-hoc networks and Delay-Tolerant Networks. In such networks every peer participates in forwarding data, including both the routing and the end-host functionalities of more conventional networks. In a Delay-Tolerant Network, peers may cache messages and deliver them to other peers at a later time, supporting end-to-end delivery even when peers are only connected intermittently. Caches have limited size, so peers must selectively remove cached messages when the cache is full. As an additional aid to selecting messages to be removed from a cache, peers can remove messages for which they have received a matching ack. This can be done while preserving both security and anonymity, by including in every message, unencrypted, a message ID computed as the hash of the message ack sent encrypted with the message. A peer seeing a new ack can then hash it and discard any cached message whose message ID matches the hash of the ack.

KEYWORDS: Ad-Hoc Networks, Delay-Tolerant Networks, Security, Anonymity, Confidentiality.

1. Introduction

The acknowledgement, commonly abbreviated **ack**, is common in networking protocols that seek to provide reliable transmissions over an unreliable medium. TCP relies on acks to confirm receipt of data transmitted on a connection, and also to grant permission to send further data on that connection. Because of this, acks are an essential part of TCP data transmission. TCP acks each have a 32-bit ack number indicating the sequence number of the next byte of data expected on the connection. In a connection transmitting data in just one direction, acks flow in the direction *opposite* the flow of data.

The expected value of a TCP ack can be computed by anyone observing the flow of TCP traffic, so attackers may create and transmit spoofed acks [1].

This paper describes an ack mechanism that provides:

- a guarantee that a message has been delivered, since only the intended receiver can transmit a valid ack for the message (Sections 3 and 4). The intended receiver is securely identified by its ability to decrypt the message.
- a measure of anonymity, in that the ack message in no way identifies either the sending or the receiving device (Section 3). This offers more protection against traffic analysis compared to protocols where the ack carries source and destination addresses.
- the ability to reliably and securely delete acked messages from message caches, even on peers that have no access to any secret key (Section 3).
- communication to the sender that partial receipt of a message is sufficient for the receiver (Section 5).

2. Background

2.1. Secure Hashing

A hash function is a function that computes a fixed-length bitstring, called the hash, from a variable-length input bitstring. In most applications it is beneficial if the hash bears no resemblance to the input. Such hashes are used, for example, in hash tables to evenly distribute the indices to which keys are assigned.

A secure hash or cryptographic hash [2] is a hash that is hard to invert, which means that, given a hash, it is hard to create an input that hashes to that given hash value. Such hashes have many security applications, and several hash algorithms have been standardized by government

agencies such as the US National Institute of Standards and Technology (NIST) [3].

In brief, the result of hashing a bitstring with a cryptographic hash is the hash value or *digest*. As long as the security of the hash is strong, it would be very hard for an attacker who has no access to the original message, to create a bitstring that hashes to the same hash value. In contrast, anyone with access to the original bitstring can easily hash it and verify that it indeed matches the hash value that was received.

The algorithm described in this paper hashes an acknowledgment and sends in the clear the corresponding hash value, which in this paper is called a *message ID*. In the same message as the message ID, the acknowledgment (ack) is sent encrypted, such that only the intended recipient can decrypt it.

After the receiver has decrypted an ack, it can broadcast it. Any peer receiving this decrypted ack can hash it and verify that the previously seen message ID matches the acknowledgment. As long as the hash is hard to invert, only the intended receiver (and the original sender, who also has access to the unencrypted version of the ack) can broadcast a valid ack.

This is an algorithm that can be used with any encryption algorithm and any secure hash function, and in that sense is very general.

2.2. Delay Tolerant Networks

In an ad-hoc network, every peer communicates wirelessly with every other peer in its range. The network is ad-hoc because its connectivity may change with changing conditions. Each peer contributes what it can to data forwarding as well as to creating and receiving data, behaving as a combination of the roles of host and router in other networks.

Delay-Tolerant Network (DTN) technologies [4, 5, 6] support communication among peers that may only occasionally be in communication range of each other. Such conditions are common among mobile peers that communicate through an ad-hoc network. The purpose of DTNs is to deliver data even in the absence of any simultaneous end-to-end path between sender and receiver.

In the 1970s and 1980s email was often delivered even to peers that were never directly connected to the Internet [7]. For example, a peer H (Host) would from time to time dial up another peer G (Gateway) with better connectivity and which had agreed to cache email to and from H. The connection used a protocol called UUCP (unix-to-unix copy) [8] to download emails addressed to the users of H, and to upload emails originated by users of H. H might then in turn forward emails to other peers that depend on it for connectivity.

To support this intermittent email delivery, G had to save messages addressed to users of H and other peers that connected to H, delivering them on request. G would likewise save outgoing messages originating from H and others until G itself could connect to its upstream peer (or directly to the wider Internet) to deliver these messages.

These techniques allowed email to be delivered even if neither sender nor receiver were ever directly connected to the Internet, as long as they were able to connect to someone else with either an Internet connection, or closer to another host that was connected to the Internet.

DTNs have similar goals as the old uucp email system, but delivering general-purpose messages rather than just email. Intermediate peers in a DTN cache messages and deliver them on request. Typically the intermittent connection is established when one of the peers moves into wireless range of another peer.

To accomplish this delivery, in a DTN all devices are peers, so each device must include all the functionality that in a more conventional network is divided among data sources, data sinks, and routers.

When a sending and a receiving DTN node both have access to the Internet, messages can be delivered directly from the sender to the receiver. When the sender, the receiver, or both are not connected to the Internet, they may still be able to communicate directly with each other over ad-hoc links. If at any given time there is no path between sender and receivers, all peers reachable by the sender cache the message, in case they are able to deliver it later.

As connectivity changes and allows communication with new nodes, peers can forward their cached messages to any new peers, with the goal of eventually delivering each message to its final destination.

In their most extreme form DTNs are only useful for data that is not delay-sensitive. Delay-sensitive communications can still occur over ad-hoc networks as long as the devices have an end-to-end simultaneous path over any combination of the Internet and ad-hoc networks, whereas delay-tolerant communications can be supported even without simultaneous end-to-end connectivity.

2.3. A Useful Delay Tolerant Network: AllNet

One useful application of DTNs is delivery of text messages (chat) among mobile devices. Users carry their mobile devices even to locations with no or intermittent connectivity[1]. It would be desirable for users to be able to communicate with at least their neighbors even in the absence of Internet connectivity. Where such connectivity is intermittent, cached messages can be forwarded once connectivity is available. This way of sending text messages resembles at a high level the uucp email delivery described above, but the details are very different, especially the unpredictable availability of wireless channels compared to the scheduled uucp connections over telephone modems.

The benefit of a chat application is that data requirements are moderate, while the usefulness can be very substantial even in situations where delivery of individual chat messages is delayed.

Creating and using ad-hoc networks and DTNs for such applications is the main goal of the AllNet project [9]. AllNet is designed to work whenever devices can communicate directly among each other even in the absence of cellular or Internet service. Available technologies include point-to-point (infrastructure-less) WiFi and the many variants of Bluetooth, particularly Bluetooth Low Energy (BLE). All

[1]The lack of connectivity may be due wilderness adventure, foreign travel, emergency situations, rural areas, or any other reason, such as being outside of the provider's coverage.

of these are available on popular mobile devices, though in many cases the operating system imposes idiosyncratic restrictions on their usage.

Since peers that might forward a message also have the ability to inspect the message, AllNet encrypts the contents of interpersonal messages to prevent eavesdropping. Communication in AllNet needs to identify neither sender nor receiver. Messages may carry optional addresses, only used to improve efficiency of message delivery.

AllNet authenticates users to each other when they are within direct communication range of each other, or when they both know a secret string that allows them to authenticate to each other over the Internet. Such an exchange creates keys that the parties can use at any time thereafter, with assurance that they really are communicating with each other.

AllNet provides some anonymity of communication both with the addresses being optional, and to a lesser extent by the use of ad-hoc communication. AllNet provides such anonymity without being vulnerable to DDoS amplification attacks [10].

The optional nature of addresses in AllNet supports reasonable tradeoffs between anonymity and performance. A message with 0 significant bits of address is anonymous, and so may be delivered to all peers within reach. On the other hand, intermediate peers that care about performance, including especially bandwidth and battery life, may be more willing to forward messages that carry more significant bits of destination address than messages with fewer bits of destination address, since such messages may be delivered more precisely, ultimately consuming fewer network resources. The sender of a message can then choose fewer bits of address to give greater anonymity, or more bits of address to give greater likelihood of message delivery. This choice may be made dynamically, based for example on network traffic, assuming that less overall traffic implies more chances of delivery for anonymous messages.

Ethical Statement: Like every other security feature, encryption, authentication, and anonymity are intended to protect some people from other people. Like every other security feature, anonymity can be used to shield ethical behavior or unethical behavior. This paper does not attempt to distinguish such uses. In general, purely technical work cannot favor ethical over unethical uses of technology.

As is true for many other protocols (including `https`), encryption, anonymity, and authentication are likely to encourage people to use the technology, and their absence would likely discourage people from using technologies such as AllNet. For an ad-hoc network these security properties are essential since there is no assurance that ad-hoc peers, who might forward all of one's messages, will be friendly.

3. Acknowledgments and Caching in an Ad-Hoc Network

Since DTNs work best with all-to-all delivery of anonymous messages, conceptually each peer in a DTN has to cache all messages. Not only do caches have storage limitations, exchanging cached data with every peer that one encounters may require substantial spectrum and too much energy from a limited battery. For all these reasons a peer that caches messages should be informed when one of its cached messages has been delivered to its final destination. AllNet does this by sending a small ack message that confirms receipt for each data message that has reached its final destination.

There are many differences between TCP acks and AllNet acks:

- an AllNet ack gives evidence that an application, rather than the transport layer, has received the message. In TCP, even data that has been acked by the receiving system may never be delivered if the application crashes or stops reading the socket.
- attackers cannot spoof AllNet acks, since only the intended receiver can decrypt the message and issue the corresponding ack. Technically, the sender of the message could also issue the ack, but a legitimate sender by definition is not an attacker.
- in TCP, acks are normal TCP segments that may or may not carry user data. In AllNet acks are 16-byte (128-bit) random strings. As long as the acks are randomly generated, by the birthday paradox the chances of a collision are small as long as there are substantially fewer than $2^{64} = 18,446,744,073,709,551,616$ acks.
- AllNet acks are anonymous. Nothing in the 16-byte random string identifies either the sender or the receiver.
- since an AllNet ack message may carry multiple ack values, a single ack message can acknowledge data messages from different conversations at once. A TCP cumulative ack may acknowledge multiple segments at once, but all such data segments were sent on the same connection.

Each AllNet peer receiving an ack message caches the acks it contains. The peer also hashes the ack, giving the message ID of the message it is acking. If the message ID matches any message that this peer has originated, that message is marked as acknowledged. Also, any matching data message in the peer's cache no longer needs to be cached.

AllNet peers forward ack messages to their ad-hoc peers and across the Internet. Unlike data messages, acks from different messages and from the ack cache may be combined and forwarded together in a single ack message.

When an AllNet peer retransmits a data message for which it has not received an acknowledgment, it may get a matching ack from a peer other than the final receiver, if that peer has cached an ack for that message.

Ack transmission in AllNet is no more efficient as the delivery of the data message, but acks are much smaller than most data messages, so any overhead is less, and likewise the need to evict acks from caches is less – a same-sized cache can hold many more acks than data messages.

Since acks in most systems (including both TCP and AllNet) are idempotent, meaning that receiving the same ack once has the same effect as receiving it multiple times, duplicate transmission of acks has no consequences beyond the cost of transmission.

Acks are also less important than data messages. In contrast to a dropped data message, the worst possible result of

a dropped ack is the sender not knowing that the message has reached its destination and transmitting a duplicate copy of the message, whereas a dropped data message may have the potentially much more serious result of failure to communicate.

4. Technical Details: Secure Acknowledgments

An attacker can easily generate spoofed TCP acks by observing any part of the connection traffic [1]. As a result TCP is not secure, and higher layers such as TLS must be used to provide some assurance of delivery to the intended party.

AllNet provides by design many of the features provided by the combination of TCP and TLS, but for connectionless decentralized ad-hoc networks and DTNs. The security of these transmissions can only be guaranteed if the intended receiver of a message is the only system able issue the corresponding ack.

In AllNet, each receiver holds one or more cryptographic private keys that it uses to decrypt messages addressed to itself. As mentioned above, AllNet certifies keys based on interpersonal interactions among users, whereas TLS relies on hierarchically issued certificates that are vulnerable to hackers penetrating the systems that issue certificates [11].

A sender generating a data message for a specific receiver includes the ack in the plaintext part of the message before encrypting it, at which point the ack is included in the encrypted part of the message.

The sender then adds to the unencrypted part of the data message the cryptographic hash of the ack.[2] This hash is known as the message identifier or **message ID**. The message ID, like the ack, is very likely to be unique for each message.

The entire process is shown in Figure 1.

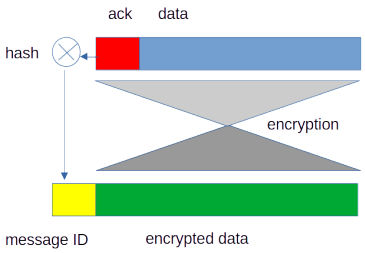

Figure 1: Relationship between ack and message ID

Every peer can see the message ID, but cannot generate a valid ack without access to the receiver's key, so only the intended receiver [3] can generate a valid ack.

A peer that receives an ack can verify whether it matches any of its cached messages or any incoming message by hashing the ack to give the corresponding message ID, then comparing this message ID computed from the ack to the message ID which is in the unencrypted part of every cached or received message.

Since the original sender also has access to the cleartext ack, the sender could also cancel messages that are cached, but this feature is unused both currently and in the foreseeable future.

4.1. Anonymity of Messages and Acknowledgments

Each source and destination address in AllNet messages is defined by a number of significant bits specified as part of the message header. A message whose addresses have no significant bits carries no information about the sender or intended receiver, and as such every peer on the network will try to decrypt it, so that any peer able to decrypt a message is an intended recipient.

The concern with such broadcast messages is the resource usage in having all peers forward and attempt to decrypt every message. The significant bits mechanism of AllNet addresses allows any number of bits to be specified, allowing a sender to specify a small number of bits to reduces the resource usage of the network while still preserving some anonymity. Senders are incentivized to provide as many bits of address as will still retain anonymity, and thereby minimize network resource usage, since packets with more bits of address are more likely to be delivered.

Just as messages can be anonymous, so acks in AllNet are also anonymous in that the ack itself carries no information about the sender and receiver of the ack. In addition, since any peer in the network might have cached the corresponding message, to the extent possible acks are distributed to every peer in the network. This universal distribution makes acks more resistant to traffic analysis than if they were only delivered back to the originator of the data message.

5. Acknowledgments for Messages Larger than the Maximum Transmission Unit (MTU)

The secure acks described so far allow for interesting possibilities when the size of a message is greater than a network's Maximum Transmission Unit (MTU), requiring that the message be sent as a collection of packets instead of a single message. Such large messages can be used to send multimedia data such as audio, images, and video.

The Internet Protocol (IP) uses a mechanism called **fragmentation** [13], and this paper uses the same term. The receiver can reconstruct the larger message once it has received all of the fragments.

In AllNet, each fragment of a larger message carries two encrypted acks, one for the message as a whole and the other for the specific fragment. Correspondingly, the unencrypted part of each fragment contains both a message ID obtained from hashing the message ack, and a fragment ID (which AllNet calls a packet ID) obtained from hashing the fragment ack.

A receiver receiving fragments of a larger message may ack them individually. Once the receiver has received all the fragments of a message, it then issues the ack for the entire message. Each peer receiving a message ack can clear from its cache every fragment it has of the larger message, even if it is only caching some of the fragments.

[2] Normally such hashes produce more than 16 bytes of data, so the sender only includes the first 16 bytes of the hash.

[3] Or anyone who can invert the hash function, which is expected to be challenging for strong cryptographic hash functions such as the SHA-512 hash [12] used by AllNet.

5.1. Acknowledging Partial Transmission

The combination of message acks and fragment acks provides some functionality beyond what traditional TCP segment acks provide. Specifically, if the content can be delivered without delivering all the fragments, then the receiver can issue the message ack immediately even with some fragments still missing.

As one example, email is often sent as both a plain text version and an html version. If either part of the message is received in its entirety, that part can be displayed to the user without having to receive every fragment of the other part, and the receiver can issue the message ack immediately.

When sending image or video data, the resolution of the image or video usually does not need to exceed the resolution of the device. Sending a low-resolution image first, followed by the high-resolution image, allows a low-resolution device to immediately send the message ack, without having to wait for or process the high-resolution image, and likewise allows the sender to only send the low-resolution version. The sender can immediately start sending the high-resolution version if it gets fragment acks for all the fragments of the low-resolution version, but no message ack.

For transmission of more general data (beyond videos and images) there are many schemes for forward error correction (FEC) that involve sending redundant data. If this data can be fragmented appropriately so that the message can be reconstructed even while some fragments are still missing, then a receiver that sends the message ack as soon as it is able to reconstruct the message can avoid having the sender engage in retransmission of any missing fragments.

There are many FEC algorithms. A particularly simple (and inefficient!) FEC scheme simply transmits each fragment 3 times. A receiver that obtains at least one copy of each fragment can immediately issue the message ack, providing reliable transmission without retransmission even in the case of substantial packet loss or transmission delay. Avoiding retransmission is particularly useful in Delay-Tolerant Networks.

If synchronous communication is available, the sender may receive the message ack for such a triply-redundant transmission before sending all three copies of all of the fragments, and can immediately stop transmitting the duplicate/triplicate information. On the other hand in a DTN, a receiver may over time receive a random subset of the fragments, and can then deliver the message to its application and issue the ack as soon as it receives all the fragments needed to reconstruct a complete message. This message ack lets other peers remove from their caches even fragments that the destination has never received.

6. Comparison to TCP Acknowledgments

Section 4 described how AllNet secure acks reliably assure the sender that messages have indeed been received by the intended recipient [4]. This is substantially different from TCP acks, which can be issued by any attacker that knows the sequence numbers in use on the connection and can spoof source IP addresses. A TCP sender has no way to know that such acks are not from the legitimate intended receiver.

Assurance that the ack was issued by the intended receiver is especially valuable in ad-hoc networks and DTNs: since the network is not organized by an authority, there is no reason to believe that intermediate peers are benign. AllNet, as other secure ad-hoc networks, had to be designed assuming that attackers may control at least some of the peers that are forwarding messages and acks.

Further differences between AllNet secure acks and TCP acks follow:

TCP acks carry the sequence number following the last byte that was received. This makes TCP acks cumulative, meaning that a single ack can acknowledge many segment's worth of data, and that loss of a single ack in a continuing stream of data transmission is not likely to lead to retransmission. On the other hand, since TCP acks count bytes rather than messages/packets/segments, and since TCP ack numbers are 32 bits, an ack in the original TCP can acknowledge up to 2^{32} different bytes, or an ack using the Protection Against Wrapped Sequences [14] (PAWS) mechanism can theoretically acknowledge up to 2^{64} different bytes of data.

Unlike TCP acks which count bytes, AllNet secure acks identify packets, so a single 1,000-byte message requires only one distinct secure ack in AllNet but consumes 1,000 sequence and ack numbers in TCP. This is of interest when we consider how many outstanding unacknowledged messages can be handled by each protocol.

The AllNet secure acks are not counters, so cannot be used as cumulative acks (where one ack potentially acknowledges many, many data packets) as in TCP, but one single AllNet message ack can acknowledge many different fragments.

Since there are 2^{128} possible different randomly selected AllNet message acks, the birthday paradox tells us that the chances of collision is extremely low until the number of acks begins to approach $\sqrt{2^{128}} = 2^{64}$.

Unlike TCP, these secure acks might collide with acks from any sender, whereas the TCP connection mechanism ensures that sequence and ack numbers can only overlap within a connection. These differences are summarized in Table 1.

Table 1: Comparison of what TCP and AllNet acks can distinguish.

Protocol	Can reliably distinguish
Original TCP	2^{31} bytes in a window
TCP with PAWS	2^{63} bytes in a window
AllNet	2^{64} simultaneous messages

6.1. Performance Analysis

Given a 64-bit sequence number space for TCP and with reasonable assumptions against delivery of old packets, TCP is guaranteed not to have sequence or ack collisions as long as 2^{63} or fewer bytes are transmitted on a connection within a two-minute Maximum Segment Lifetime (MSL) period, leading to a maximum bandwidth of over 2^{63} / $120s$ = 7×10^{16} bytes/second for each TCP connection.

For the secure acks described in this paper, to stay well away from the birthday paradox we assume that it would be desirable to have no more that about 2^{60} unacknowledged

[4] As long as the recipient's key and the encryption algorithm have not been compromised.

messages in the network at any given time. We further assume message sizes of 1,000 bytes and a maximum message lifetime (as specified by the message expiration option in AllNet) of about a week or 604,800 seconds. Satisfying these assumptions limits the entire network to about $2^{60} \times 1000$ / 604800 = 10^{15} bytes per second.

For communication across the Internet a message lifetime of a week is excessive. Using the same 2 minutes as for TCP, the network can support almost $2^{60} \times 1000$ / 120 = 10^{19} bytes per second.

While this throughput for the entire network cannot be directly compared to the TCP per-connection throughput, it is quite adequate for both the current, largely experimental AllNet, and for any foreseeable developments. Ad-hoc networks are typically small and relatively inefficient [15], and even in the imaginable future are unlikely to scale to large sizes and large amounts of traffic. Therefore for the ad-hoc side of the AllNet communications, even the lower network throughput derived by assuming a 1-week message lifetime is very abundant.

Should the capacity of AllNet ever become an issue due to the limited number of bits in an ack, AllNet could evolve, as TCP already has, to use more acknowledgments bits.

6.2. Performance Results

As in TCP, AllNet acks are sent unreliably, and can therefore be lost. Again as in TCP, the mechanism for requesting an ack retransmission is to retransmit the original message.

Unlike in TCP, for messages that are retransmitted, any intermediate host that has cached the ack can resend it immediately, without the original receiver having to respond. As compared to networks that support TCP and therefore require continuous end-to-end connectivity, for DTNs this increases reliability of ack delivery.

Decrypting a message to generate the ack is more time consuming than the process to generate a TCP ack. TCP has the additional performance advantage of being implemented in the kernel.

With these caveats, we compared the time to return an ack in AllNet between two hosts connected at distant locations across the Internet, with a ping time of 83ms. The first ack (really, SYN+ACK) from the TCP connection establishment 3-way handshake took 125ms (measured using `tcpdump`). Sending a message to an existing contact with AllNet between the same two hosts returned the ack in 154ms (measured using AllNet's `trace` program). These are comparable results.

7. Future Work and Conclusions

It is clear from the above analysis that if AllNet ever becomes as popular as TCP, it will have to be redesigned or extended for higher performance, just as TCP has been and likely will be again in the future. Moving from 16-byte acks to 32-byte acks would address any foreseeable performance limitation due to ack size.

The secure acks described in this paper provide many advantages over conventional acks such as used in TCP. This paper has explored a few, including the guarantee that the ack can only be issued by a receiver that has the correct cryptographic key, the ability to use the combination of message ack and fragment ack to let the sender know how much of the data the receiver actually needs, and the use of the acks to securely enable removing acknowledged messages from peer caches.

Secure acks as described above only work when sent encrypted. Encryption of data messages is pervasive in AllNet, so sending an encrypted ack with the encrypted data of a message requires no additional effort.

Should there be a reason to send the data unencrypted, the ack itself could still be encrypted, as long as keys are distributed in such a way that only the intended recipent can issue the ack.

When encryption is not an option, one could instead imagine having identical ack generators (for example, some kind of secure random number generator) based on identical secret seeds on each pair of sender and receiver, such that the receiver can generate the same sequence of acks as the sender. In such a case, the acks do not need to be transmitted at all. Instead, the sender would include a counter or an identifier for the ack associated with a message, and the receiver can independently generate the ack and hash it to compare the locally generated acks to any received message IDs. The receiver can then issue a valid ack that can be verified by any peer that is caching messages. While these acks do acknowledge receipt, they cannot be used by peers to discard cached messages.

Alternatively, in a scheme somewhat resembling the Bitcoin blockchain [16], and if anonymity is not a concern, a receiver could digitally sign a received message ID with a widely known public key. This scheme requires an infrastructure (perhaps a blockchain?) to record and distribute the public keys, but does not require encryption. Since a shared blockchain requires a persistent connection to the Internet, this scheme is more suitable for systems that can rely on Internet connections than for systems that use ad-hoc and delay-tolerant communications.

We have explored some of the design space for secure and anonymous acknowledgments. While this section has indulged in speculation, the mechanisms in Sections 3 and 4, and in the initial part of Section 5, are fully implemented and live on the AllNet network.

References

[1] Hastings, McLean, "TCP/IP spoofing fundamentals", 1996 International Phoenix Conference on Computers and Communications, doi: 10.1109/PCCC.1996.493637

[2] NIST, "Secure Hash Standard (SHS)", FIPS 180-4, August 2015.

[3] NIST, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", FIPS 202, August 2015. https://csrc.nist.gov/pubs/fips/202/final

[4] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets", SigCOMM, Aug 2003.

[5] Benhamida, Bouabdellah, Challal, "Using delay tolerant network for the Internet of Things: Opportunities and challenges", 2017 8th International Conference on Information and Communication Systems (ICICS), 2017, doi: 10.1109/IACS.2017.7921980

[6] Mallorqui, Zaballos, Serra, "A Delay Tolerant Network for Antarctica", IEEE Communications Magazine, August 2022, doi: 10.1109/MCOM.007.2200147

[7] Partridge, "The Technical Development of Internet Email", IEEE Annals of the History of Computing, vol. 30, no. 2, April-June 2008, doi: 10.1109/MAHC.2008.32

[8] Nowitz, "Uucp Implementation Description", Unix Manual Version 7. https://web.archive.org/web/20180221100921/http://a.papnet.eu/UNIX/v7/files//doc/36_uucpimp.pdf

[9] Biagioni, "Ubiquitous Interpersonal Communication over Ad-Hoc Networks and the Internet", 47th Hawaii International Conference on Systems Sciences), in January 2014, and other papers at https://alnt.org/

[10] Biagioni, "Preventing UDP Flooding Amplification Attacks with Weak Authentication", International Conference on Computing, Networking and Communications (ICNC 2019), February 2019, Honolulu, Hawaii.

[11] Google Security Blog, "An update on attempted man-in-the-middle attacks", August 2011. https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html

[12] Penny Pritzker, "Specifications for the Secure Hash Standard", U.S. Federal Information Processing Standards Publication 180-3, October 2008.

[13] Information Sciences Institute, "Internet Protocol", RFC 791 (section 3.2.1.4), September 1981.

[14] Borman, Braden, Jacobson, Scheffenegger, "TCP Extensions for High Performance", RFC 7323, September 2014.

[15] Gupta, Kumar, "The Capacity of Wireless Networks", IEEE Transactions on Information Theory, vol. 46, March 2000.

[16] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", made public May 24 2009. http://bitcoin.org/bitcoin.pdf

Edoardo Biagioni did his bachelor's degrees at MIT in 1982, diplom at ETH Zürich in 1985, and PhD degree in computer science at the University of North Carolina, Chapel Hill, in 1992.

He is currently an associate professor in Information and Computer Sciences at the University of Hawai'i at Mānoa, where his research includes networking and systems. He has been working on the AllNet project (www.alnt.org) since 2011.