CLL: A Cryptographic Link Layer for Local Area Networks

Yves Igor Jerschow, Christian Lochert, Björn Scheuermann, and Martin Mauve
{jerschow, lochert, scheuermann, mauve}@cs.uni-duesseldorf.de

Institute of Computer Science, Heinrich Heine University, Düsseldorf, Germany

Abstract. Ethernet and IP form the basis of the vast majority of LAN installations. But these protocols do not provide comprehensive security mechanisms, and thus give way for a plethora of attack scenarios. In this paper, we introduce a layer 2/3 security extension for LANs, the *Cryptographic Link Layer (CLL)*. CLL provides authentication and confidentiality to the hosts in the LAN by safeguarding all layer 2 traffic including ARP and DHCP handshakes. It is transparent to existing protocol implementations, especially to the ARP module and to DHCP clients and servers. Beyond fending off external attackers, CLL also protects from malicious behavior of authenticated clients. We discuss the CLL protocol, motivate the underlying design decisions, and finally present implementations of CLL for both Windows and Linux. Their performance is demonstrated through real-world measurement results.

1 Introduction

Ethernet and the Internet Protocol (IP) are the main building blocks for the vast majority of modern Local Area Networks (LANs). However, these protocols, and thus virtually all installed LANs, do not provide comprehensive security mechanisms. Hence, malicious local users are potentially able to eavesdrop, to inject or modify information, or to take on fake identities.

One especially critical component is the *Address Resolution Protocol (ARP)* [20]. It performs the task of coupling the network layer with the link layer by resolving IP addresses into the corresponding MAC addresses. However, ARP lacks an authentication mechanism, making it vulnerable to different types of attacks. This constitutes a severe threat in every LAN that is accessible to not fully trustworthy users. By emitting ARP messages with wrong IP/MAC mappings—commonly referred to as *ARP spoofing*—a malicious user can *impersonate* other hosts, intercept and modify foreign IP traffic by becoming a *Man in the Middle (MiM)*, or mount a *Denial of Service (DoS)* attack against other hosts. Using freely available tools, e. g. [9,18], ARP spoofing can be easily performed even by users without deeper knowledge of the underlying protocols.

Preventing ARP attacks in the case of dynamic IP addresses requires to take also the *Dynamic Host Configuration Protocol (DHCP)* [7] into account. It is employed in almost every LAN to automatically assign IP addresses and configuration parameters. It does not provide an authentication mechanism either and thus can also become the target of various attacks. By setting up a rogue DHCP server and announcing forged IP addresses for the default gateway or the DNS server, an adversary is able to run a MiM or DoS attack against clients requesting an IP address via DHCP. Furthermore, the legitimate DHCP server is also vulnerable. In a *DHCP starvation attack* the adversary takes on many different client identities (usually MAC addresses) and requests each time a new IP address, until the server's address pool gets exhausted. Thereby the attacker can prevent new clients from acquiring a valid IP configuration.

Since modern operating systems enable the injection of raw Ethernet packets containing arbitrary MAC and IP addresses in their headers even in user mode, there exists no external barrier which would impede address fraud. The outlined attack scenarios are covered in more detail, e. g., in [1, 5, 23].

In this paper, we tackle the challenge of securing the communication in local area networks, including ARP and DHCP. We introduce a comprehensive layer 2/3 security protocol—the *Cryptographic Link Layer (CLL)*. It provides authentication and confidentiality between neighboring hosts in Ethernet LANs. Each machine gets identified by its IP/MAC address pair. Beyond safeguarding ARP and DHCP, CLL protects arbitrary layer 2 traffic, especially all encapsulated IP packets. We propose to employ CLL, e. g., in enterprise and campus networks being often accessed by frequently changing, not fully trustworthy users as well as in all kinds of publicly accessible LANs (like Internet cafés or Wi-Fi hotspots). Note that CLL does not affect the operation of higher layer security protocols.

Beginning with an ARP request, CLL applies public key cryptography to perform an initial handshake between two hosts with the aim to establish a security association. The two hosts prove their identity to each other and exchange keying material. Hereupon, secured IP data packets may be sent.

We have implemented and evaluated CLL on both Windows and Linux. In typical LANs running at 100 Mbit/s, our implementation operates at full wire-speed, thus securing the network without compromising the throughput. To ease the migration procedure, CLL-enabled machines can be configured to interoperate with ordinary, unsecured hosts. We make our CLL implementation available for free download including the sources, and complement it with a toolkit for key and certificate management [12].

The remainder of this paper is organized as follows. In the next section, we review previous approaches on securing ARP, DHCP, and the link layer. Section 3 sketches the architecture of CLL, before Section 4 justifies the underlying cryptographic design decisions. In Sections 5 and 6 we detail the operation of CLL's protocol components. Section 7 describes the implementation of CLL and evaluates its performance. Finally, we conclude the paper with a summary in Section 8.

2 Related Work

Above the link layer, there already exist well-proven security protocols which provide authentication and confidentiality by means of cryptography. *SSH* [24] and *SSL/TLS* [6] operate at the application level or directly below it. At the network layer, *IPsec* [13] can protect IP datagrams being exchanged between two end-points. However, IPsec does not authenticate the IP address of the communicating party. This enables an authorized IPsec user to impersonate the IP address of another host that is temporarily switched off or knocked out by a DoS attack. While SSH, SSL/TLS, and IPsec cannot protect from attacks on ARP and DHCP, the encryption performed by these protocols will still prevent the disclosure of sensitive data. An attacker would have to content himself with the power of rendering his victims unable to communicate.

Reviewing the attempts to cope with the insecurity of ARP, there exist two main directions. One is to detect the bulk of ARP attacks by means of a specialized Intrusion Detection System (IDS) like *Antidote* [2] or *ArpWatch* [3] and to warn the user or network administrator in time. Such tools monitor all incoming ARP messages and trigger an alarm, e. g., on observing an abnormally high number of ARP replies or a changed IP/MAC mapping. However, these ARP watchdogs cannot provide full protection against ARP attacks; in particular, they are not able to distinguish whether the MAC address in the first ARP reply is genuine or not. The other approach is to secure ARP by using cryptographic techniques. In the following, we discuss some current research taking this direction.

Gouda and Huang [10] specify a theoretical architecture with an ARP server sharing a symmetric key for message authentication with every host in the LAN. Each host periodically notifies the server about its current IP/MAC mapping and resolves the MAC addresses of its neighbors with the aid of the ARP server. However, this does not prevent an authorized machine from purposely announcing a mapping of a neighboring host's IP address to its own MAC address. In contrast, CLL authenticates all hosts based on their IP/MAC address pair. It thus also avoids ARP spoofing attempts originating from malicious, but authorized users. Furthermore, CLL does not require a central server.

In [5], Bruschi et al. introduce *Secure ARP (S-ARP)* which uses public key signatures to authenticate the ARP replies. All hosts in the LAN hold a private/public key pair and initially enroll at a central server, the *Authoritative Key Distributor (AKD)*. The AKD maintains a repository of public keys and the corresponding (static) IP addresses. Whenever a host requires a neighbor's public key to verify the signature of an ARP reply, it inquires about it from the AKD. The AKD's reply packet is digitally signed as well and the AKD's public key is preinstalled on all machines. S-ARP comes with an implementation for Linux 2.4, but it requires a kernel patch and does not support dynamically assigned IP addresses.

On the basis of S-ARP, Lootah et al. propose *Ticket-based ARP (TARP)* [16]. It foregoes a central key server and instead employs digitally signed attestations of IP/MAC mappings, so-called *tickets*. The tickets are issued by a trusted party, the *Local Ticket Agent (LTA)*. The host responding to an ARP request attaches its ticket to the ARP reply and thereby proves the validity. Since the LTA's public key is preinstalled on each host, received tickets can be verified quickly. In comparison to S-ARP, TARP requires at most one public key operation per ARP exchange and no private key operations, and thus offers better performance. However, the authors state that an attacker is able to impersonate a host that is currently offline, by replaying its previously captured ticket. TARP has been implemented on Linux 2.6 with support for DHCP-assigned IP addresses. Note, however, that both S-ARP and TARP aim to secure only ARP, while CLL provides overall layer 2 security by safeguarding DHCP and data packets as well.

RFC 3118 [8] specifies how DHCP can be extended by an authentication mechanism. In this scheme, the DHCP server shares with each client a symmetric key. It is

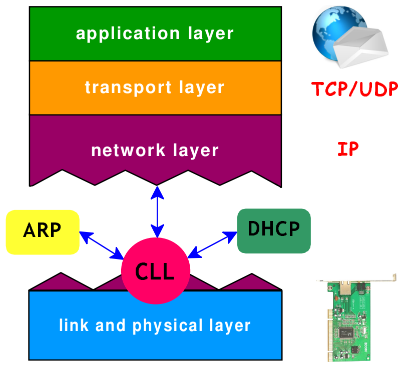

Fig. 1. CLL in the protocol stack.

used to authenticate the DHCP messages. However, DHCPDISCOVER, the first message sent by the client, remains unauthenticated. Consequently, users still might be able to perform a DHCP starvation attack. This is not the case with CLL. Another drawback is that currently no DHCP implementations with RFC 3118 support are available.

Applying cryptography at the link layer is common in Wi-Fi networks. *Wi-Fi Protected Access (WPA)* and *WPA2* provide authentication and confidentiality between wireless nodes and the access point. The IEEE working group 802.1AE [11] specifies *MACsec* as the analog of WPA/WPA2 for LANs. In contrast to CLL, WPA/WPA2 and MACsec authenticate hosts based only on their MAC address. The content of ARP and DHCP control packets encapsulated in layer 2 frames is not examined. Therefore these protocols cannot protect from ARP and DHCP attacks originating from legitimate users. Moreover, we are not aware of any MACsec implementation being available at this time.

The main contribution of this paper is a novel, comprehensive approach to layer 2 security, which provides a more complete protection of the LAN than even a combination of three existing protocols (e. g., TARP, RFC 3118, and IPsec) could achieve. That is because besides eliminating the discussed shortcomings of these protocols, CLL also authenticates broadcast traffic. The tackled security problems are all related to each other—they arise from the lack of authentication at layer 2 and the link to layer 3. Thus, a comprehensive approach to solve them seems appropriate.

3 Protocol Overview

CLL is designed as a transparent filtering service between the network adapter and the IP stack. It thus operates at the border between the link and the network layer, as displayed in Figure 1. All outgoing packets including the Ethernet header are authenticated and their payload is optionally encrypted before they are handed over to the network card for transmission. Incoming packets are passed to the IP stack only after they have been successfully authenticated and—if applicable—decrypted. CLL can be enabled or disabled without modifying the other protocol stack components. For them, CLL's services are transparent. But in fact, CLL appends its cryptographic headers to outgoing packets, and puts its own ID into the EtherType field of the Ethernet header. From successfully authenticated incoming packets CLL strips off its cryptographic headers and restores the original EtherType value before passing them up. While the operation of CLL does not require any modifications to switches, routers must either support CLL (and benefit from it) or exchange packets with the end systems in the standard, insecure manner.

CLL identifies hosts by their IP/MAC address pair. Each machine on the LAN holds a private/public key pair and a certificate issued by the local Certificate Authority (CA)—usually the network administrator—which establishes the binding between its public key, the MAC and the IP address. To verify certificates, each host requires the CA's public key. Typically it will be installed in the form of a self-signed certificate along with the host's own certificate, but a more complex Public Key Infrastructure (PKI) to support multiple LANs is also conceivable.

Basically, CLL divides all network traffic into four packet types: *ARP* and *DHCP*[1] control packets, *unicast* and *broadcast IP* data packets. Authentication is performed for all packet types and, in addition, an optional payload encryption is provided for unicast IP packets.

While ARP and broadcast IP packets are authenticated by means of public key cryptography (digital signatures in conjunction with certificates), unicast IP and DHCP packets get secured using fast symmetric key algorithms. Safeguarding unicast IP packets with a message authentication code and optionally a block cipher requires each pair of communicating hosts to share a secret key. For that purpose, CLL employs a key exchange protocol to negotiate shared keys on-demand. Since the IP traffic flow between two hosts always begins with an ARP exchange, CLL adopts it to establish a *security association (SA)* between the two peers. The two machines authenticate each other, negotiate a secret key and agree on the cryptographic algorithms to protect their IP packets. The establishment of an SA is subsequently referred to as *handshake*.

To determine the sender's (claimed) identity during the authentication of incoming packets, CLL examines the Ethernet header and, depending on the protocol, also the ARP, IP, or DHCP header. Where applicable, it performs a consistency check: the sender's MAC address can be also found in the ARP header or—in case of a DHCP client—in the DHCP header, and it must match the address specified in the Ethernet header. Such a cross-layer consistency check is not performed by other protocol layers. It is, however, crucially important to ward off ARP spoofing and DHCP starvation attacks. Layer 2 authentication alone would not suffice for this purpose.

The following listing summarizes the various LAN attacks fended off by CLL:

- *ARP spoofing:* impersonation, MiM and DoS attack
- *DHCP spoofing:* rogue DHCP server (MiM & DoS), DHCP starvation attack (DoS)
- *generic unicast attacks:* injection of spoofed packets, eavesdropping
- *generic broadcast attacks:* injection of spoofed packets, special case: smurf attack[2]

[1] Though being encapsulated in an UDP segment and an IP datagram, we handle DHCP messages as a separate layer 3 packet type due to the functional position of DHCP below the network layer.

[2] Flooding the victim via spoofed broadcast ping messages being answered by all other hosts.

4 Cryptographic Design Decisions

The security philosophy of CLL is to provide the user with a suite of up-to-date cryptographic algorithms and corresponding parameters, letting her choose between them on her own responsibility. Such a design has the advantage of considering individual security perceptions, allowing to trade off between highest-level security and best performance, and supporting the prompt exchange of an algorithm being reported as broken. With our implementation, we nevertheless provide a reasonable default configuration to assist users without deeper understanding of cryptography. The general level of protection provided by CLL may be also selected. Either CLL just authenticates all types of packets or it additionally also encrypts the payload of unicast IP packets (including the IP header). Skipping the encryption step will result in a better performance and should be done whenever a higher layer security protocol like IPsec already ensures confidentiality. CLL allows to use different ciphers and hash functions in each direction of an SA. With regard to system complexity, we however prescribe the algorithms used for key exchange, key derivation, and DHCP packet authentication. Table 1 summarizes the algorithms proposed for CLL and supported by our implementation.

During the handshake CLL applies the Diffie-Hellman key agreement protocol to exchange a symmetric *master key* with perfect forward secrecy between the two peers. Since handshake packets are digitally signed, there exists no susceptibility to man-in-the-middle attacks. To the negotiated master key we apply a deterministic key derivation function to generate for each direction two properly sized keys—one for the message authentication code and one for the optional cipher.

CLL guarantees the authenticity of unicast IP and DHCP packets by means of a *Hashed Message Authentication Code (HMAC)* [4] attached to the end of each packet. In addition to authentication, CLL offers to protect unicast IP packets from eavesdropping by optionally encrypting them with a block cipher in *Cipher Block Chaining (CBC)* mode. When establishing an SA, we generate a random *Initialization Vector (IV)* and use afterwards the last encrypted block of the preceding packet as the next packet's IV. Since transmissions on the link layer are unreliable, the sender also prepends the current IV to each packet. If the payload size is not a multiple of the block size, random padding bytes are appended. We first encrypt the plaintext and then compute the HMAC for the

Table 1. Algorithms and parameters in CLL.

message auth. codes	• HMAC with MD5, SHA-160/256, RIPEMD-160 or HAS-160 • 128–256 bit key length
encryption	• optionally with a block cipher in CBC mode, 128–256 bit key length • available ciphers: Twofish, AES, RC6, RC5, Blowfish, MARS, Serpent, CAST-128/256, SEED, GOST
key exchange	Diffie-Hellman, 2048-bit group No. 14 from the IPsec specification
key derivation	IEEE 1363a Key Derivation Function 2 (KDF2) using RIPEMD-160
key rollover	periodically on demand, e. g., every 30 min
digital signatures	• RSA with variable key length (typically 1024–2048 bits) • RSASSA-PSS signature scheme with SHA-160/256 or RIPEMD-160
certificates	X.509 v3 with RSA signature, ASN.1 BER/DER encoding

ciphertext, since this is the only order that is generally considered secure [14]. It also enables to detect a forged packet without the need to decrypt it.

To sign handshake and broadcast IP packets, CLL applies the well-known RSA algorithm in conjunction with certificates. RSA offers the great advantage of supporting public key signatures and encryption with a single key pair. And though CLL's security architecture does not require any public key encryption, in practice the local CA can make use of RSA encryption to securely deploy the DHCP HMAC keys to the users.

5 Operation of CLL in Detail

5.1 Basic Packet Format

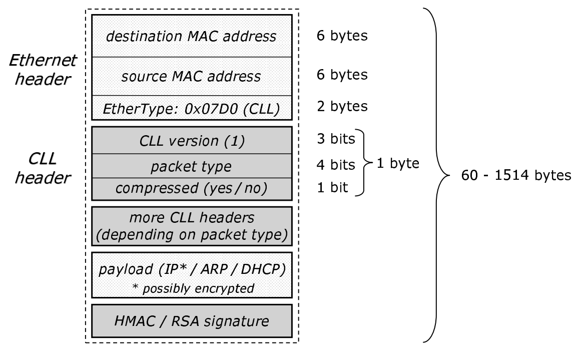

Fig. 2. An Ethernet frame in CLL.

When securing Ethernet frames, CLL inserts its own headers and replaces the EtherType value in the Ethernet header with its own identifier (0x07D0, otherwise unassigned by IEEE). Figure 2 depicts the generic layout of an Ethernet frame safeguarded by CLL. The *CLL header* is placed behind the Ethernet header. It has been designed as a compact bit field to save overhead. It consists of a version number (currently 1) like in IP, a field specifying the encapsulated packet type (*unicast* or *broadcast IP packet*, *ARP handshake packet*, *DHCP client* or *server packet*, internal *certificate packet*), and a Boolean flag stating whether the payload has been optionally compressed by CLL. This main CLL header is followed by one or more inner headers depending on the encapsulated packet's type. Therein we store, among cryptographic parameters, the original EtherType number. Behind the inner headers resides the payload, i. e., an ARP, IP, or DHCP packet. Finally, each secured Ethernet frame terminates with an authentication field containing either an HMAC (unicast IP and DHCP packets) or an RSA signature (ARP handshake and broadcast IP packets) computed over the whole frame.

5.2 ARP Handshake and SA Setup

Overview To safeguard unicast IP packets, CLL needs to establish an SA between each pair of communicating hosts. For this, CLL takes advantage of the ARP mechanism and expands it at the same time with authentication. Figure 3 illustrates this ARP handshake between two hosts A and B.

When started, a CLL implementation should first flush the ARP cache, thus ensuring that all IP traffic to other hosts is preceded by an ARP request. Having intercepted the ARP request, CLL wraps it up into a digitally signed handshake packet. It includes the host certificate and cryptographic parameters to establish the SA. The handshake packet is broadcasted like an ordinary ARP request and every station on the LAN checks whether it holds the inquired IP address. At the destination host, CLL verifies the certificate of the requesting host and validates the packet's signature. Invalid packets are dropped. Then it must be checked whether the sender's IP/MAC address pair claimed in the ARP request (and its MAC address stated in the Ethernet header) matches the one specified in its certificate.

If the handshake packet turns out to be valid, CLL creates a new SA with the requesting host, based on the local and the received cryptographic parameters. Finally, CLL strips off everything from the handshake packet except for the ARP header, restores the ARP EtherType number in the Ethernet header and passes the resulting ordinary ARP request up the protocol stack to the ARP module. The ARP module creates then an ARP table entry for the requesting host, and responds with an ARP reply. This reply gets intercepted again and is encapsulated into a digitally signed handshake packet analogously to the ARP request, along with the local cryptographic parameters and the host certificate. CLL then unicasts this second handshake packet to the requesting host like a usual ARP reply. In the following, we refer to the first handshake packet as the *handshake request* and to the second one as the *handshake reply*. On the requesting host the handshake reply undergoes the same verification process before the SA is established and the ARP reply is passed up to the ARP module.

Creating an SA implies the computation of a joint master key from the public and private Diffie-Hellman values. From the master key, CLL then derives the four keys for the HMAC and the optional block cipher. At any time, only one SA is permitted per host pair.

Handshake Packet Details We employ a UNIX timestamp and a nonce to protect against replay attacks. CLL requires the clocks of all hosts on the LAN to be synchronized within reasonable limits decided on by the network administrator, e. g., in the

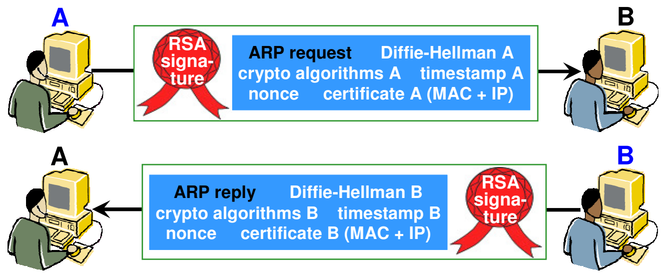

Fig. 3. ARP handshake: Diffie-Hellman key exchange in conjunction with RSA signatures.

range of 2–5 minutes. This can be easily achieved if the users manually adjust their computer's clock occasionally. An automatic clock synchronization, for instance by using NTP [17], is also possible after having established an SA to a trustworthy server.

The nonce is a random 64-bit number generated by the initiator of the handshake, which expects to find it repeated in the handshake reply. It ensures that the peer actually participates in the protocol, i. e., its handshake reply has not been replayed. Due to the nonce, it is not necessary to verify the timestamp in the handshake reply. It must, however, be stored for comparisons with timestamps of possibly future handshake requests.

The other important handshake element are the cryptographic parameters. Each host specifies the hash function configured for the HMAC and the block cipher potentially chosen to protect the payload against eavesdropping, along with the key sizes. A compression algorithm may be specified as well, if a host intends to compress its outgoing unicast IP packets. Moreover, each party states how long the SA should be valid before it is either extended or removed due to inactivity. The actual SA validity period is the minimum of the two claims. However, it may not fall below a threshold currently set to 15 minutes to prevent permanent handshakes or renegotiations.

Retransmissions and Conflicts CLL addresses the possibility of a handshake packet loss by means of retransmissions. In case of a lost (or just unanswered) handshake request the standard ARP retransmission mechanism will trigger a new ARP request. Having intercepted this ARP request, CLL retransmits the respective cached handshake request after updating its timestamp and signature. Through caching we avoid the computation-intensive generation of new Diffie-Hellman values.

The loss of a handshake reply will also result in a retransmission of the corresponding handshake request. The answering peer caches the received original handshake request as well as its own handshake reply. It is therefore able to recognize the incoming duplicate handshake request, and retransmits its handshake reply. Due to the receiver relying on the nonce, we can even omit to update timestamp and signature in this case.

Theoretically, it is conceivable that two hosts without an SA concurrently send each other a handshake request, when both of them have a pending IP datagram destined for the other one. However, only the creation of a single SA is allowed between two hosts. CLL resolves this issue by performing an integer comparison between the two 48-bit MAC addresses: the handshake request of the host with the higher MAC address "wins".

Complete and Incomplete SAs From the point of view of a host, we refer to an SA as *complete* when it is known for sure that the peer has also established the SA. Host A as the initiator of an ARP handshake can set up the SA with its peer B only after having received the handshake reply. A's SA is therefore complete right from the start. Host A can immediately send secured unicast IP packets to its peer B and be certain that B will be able to verify and decrypt them.

In contrast, host B first has an *incomplete* SA, as long as it cannot be sure whether A has received its handshake reply. Usually, the IP datagram of host A that triggered the ARP request will quickly reach host B and thereby confirm the set up SA. However, as long as this is not the case, host B may not transmit any IP packets to its peer—A might

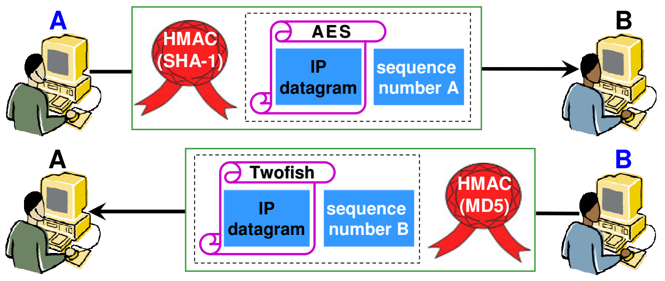

Fig. 4. Transmission of unicast IP packets safeguarded with a block cipher and a message authentication code.

not be able to authenticate them. Instead, in the unlikely case that B wants to transmit to A before A has sent the first packet, B must queue its IP datagram and send a new handshake request to A. This enforces the creation of a new SA, replacing the existing incomplete one.

Safeguarding against Replay Attacks While the initiator of the SA protects itself against a replayed handshake reply with the aid of a nonce, its peer has to rely on the timestamp check when judging the freshness of an incoming handshake request. However, a timestamp is considered valid within a period of several minutes (smaller than the minimum SA duration) to tolerate time deviations. It hence does not assure a complete protection by itself. An attacker may try to replace an existing SA by replaying a captured outdated handshake request bearing a timestamp which is still valid. CLL fends off such attacks by comparing the timestamp of a new handshake request with the timestamp of the handshake request or reply which led to the establishment of the currently existing SA. The use of timestamps avoids the necessity of a third message for a second nonce in the other direction, which would render the ARP handshake more complex.

5.3 Unicast IP Packets

Having created ARP table entries and established an SA, unicast IP packets can be transmitted between the two peers. This is illustrated in Figure 4. While host A encrypts its packets with the block cipher AES and authenticates them with an HMAC using the hash function SHA-1, its peer B employs Twofish and MD5. Taking the sender's MAC address the receiver looks up the corresponding SA to verify the packet's HMAC, sequence number, source IP address, and to decrypt the IP datagram. Only if the peer is a router, its IP address may differ from the source address stated in the IP header.

Each unicast IP packet contains a sequence number to protect against replay attacks. It is incremented by one with each packet sent to the respective destination. The receiver tolerates packet losses and only checks whether a packet's sequence number is larger than that of its predecessor. The sequence numbers are transmitted as plaintext to avoid an unnecessary decryption of replayed unicast IP packets. However, in order not to

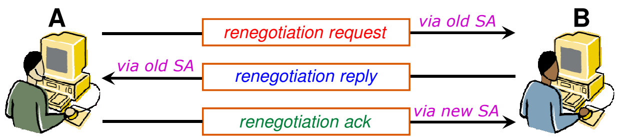

Fig. 5. Renegotiation—renewing an SA.

reveal the number of packets exchanged between two hosts so far, we generate the initial sequence numbers—one for each direction—at random.

Note that once having created an SA, CLL can also secure unicast packets carrying some other layer 3 protocol, e. g., Novell's IPX.

5.4 Periodical Key Rollover

By design, an SA has a short lifetime of typically 15–60 minutes like an ARP cache entry. But if any IP packets are transmitted during this period, it is renewed by a new Diffie-Hellman key exchange. New session keys for the HMAC and block cipher as well as sequence numbers are derived from a new master key. We call the extension of an SA *renegotiation*. Figure 5 illustrates the messages exchanged between two peers to extend their SA.

The *renegotiation request* and *reply* are the counterparts of the handshake request and reply. They are transferred through the existing SA like usual unicast packets. Each peer establishes a new SA after receiving the corresponding renegotiation packet. Just like when initially setting up an SA, host A's SA is complete from the beginning on, while host B first has an incomplete SA. But in case of a renegotiation, we cannot expect that an IP packet will be transmitted from A to B shortly and render B's SA complete as well. Therefore, host A has to explicitly acknowledge the reception of the renegotiation reply. It does so by means of a *renegotiation ack* sent through the new SA.

The renegotiation is initiated by the peer that first determines the expiration of the SA according to its clock. Concurrent renegotiation attempts are resolved in the same way as in the ARP handshake by performing a MAC address comparison.

During the renegotiation the peers re-exchange and re-validate their *current* certificates to address a possible expiration of the previous ones, especially in case of short-lived certificates issued via DHCP. While a renegotiation is in progress, pending IP packets destined for the peer can be still transferred through the old SA, i. e., there is no need to delay and queue them. We address the possibility of renegotiation packet losses by means of a retransmission mechanism.

5.5 Broadcast Packets

CLL authenticates broadcast IP packets like handshake packets by means of an RSA signature. To verify the signature, the receivers require the sender's host certificate. However, the variable payload size of a broadcast packet may well be too large to piggyback the certificate and still stay within the maximum segment size limit. Therefore, we broadcast the certificate in advance in a special *certificate packet*. CLL sends a certificate packet only when dispatching a broadcast packet and when more than a minute has passed since the previous certificate transmission, i. e., periodically on demand. All hosts on the LAN cache the received host certificates. Thus they need to validate each certificate only once and henceforth have the correct public key readily available for future signature verifications.

Like handshake packets, broadcast packets are protected against replay attacks by means of a timestamp combined with an additional counter. If a host sends more than one broadcast packet at the same UNIX time (i. e., within one second), it increments this counter with each packet by one. All receivers store for each sender the timestamp and counter from its last broadcast packet. Subsequent packets from the same sender must bear a newer timestamp or the same timestamp with a higher counter value.

When dealing with high-rate broadcast traffic, the generation of RSA signatures on a per-packet basis may become computationally infeasible in real-time. However, in this case it is conceivable to queue outgoing broadcast packets for a short time and sign the accumulated group of packets as a whole with a single private key operation before dispatching them. The receivers would reassemble this group and verify the overall signature attached to the last packet. A sophisticated but also more complex approach tolerating packet losses might be the application of a specialized broadcast authentication protocol like TESLA [19].

6 Integrating and Securing DHCP

6.1 Basic Concept

So far, we have described the case of a static IP configuration, where the local CA creates for each machine a *host certificate* incorporating its MAC and IP address. However, CLL also supports the automatic assignment of IP addresses by means of DHCP. The DHCP message exchange is safeguarded and extended. CLL protects DHCP not only from unauthorized attackers, but also from malicious behavior originating from authenticated hosts.

In case DHCP is used, the local CA issues a *base certificate* for each host, bearing only the machine's MAC address and no IP address. The DHCP server uses the base certificate as a template to generate a full-fledged host certificate, which contains the assigned IP address. Thus, it acts as a second CA. The host certificate issued by the DHCP server has the same validity period as the IP address lease. When extending the DHCP lease, the host certificate is renewed accordingly.

Securing DHCP implies the authentication of all DHCP packets and a consistency check of the DHCP header in client-originated messages. Since CLL supervises the complete DHCP traffic in a transparent way, it also takes on the automatic application for a host certificate and its issuing. Its operation does not require any modifications on the employed DHCP client or server. On the client side, CLL attaches the base certificate to the DHCPREQUEST message. On the server CLL verifies this request and strips off its own headers, before passing it up to the DHCP module. It then waits for the outgoing DHCPACK message. This message constitutes the confirmation that the

DHCP server has assigned the requested IP address. CLL extracts from it the allocated IP address along with the lease time, and issues a corresponding host certificate. Piggy-backed on the DHCPACK message, the new host certificate finally reaches the client, which can now finalize its IP configuration and is ready to establish SAs.

6.2 Authenticating the Packets

We have designed the authentication of DHCP packets in a way that allows to employ an HMAC from the beginning, without requiring an initial public key handshake. DHCP traffic occurs only between the clients and typically one single trusted server controlled by the local CA. Therefore, the number of communicating host pairs is limited and it is feasible to statically configure pre-shared HMAC keys. This task may be performed during the certificate enrollment without any additional effort. The local CA can generate a secret HMAC key for a host along with its base certificate. After encrypting the HMAC key with the host's public RSA key it can deliver these items to the user, e. g., via e-mail.

If the issued HMAC key were completely random, one would have to promptly configure it on the DHCP server as well, which involves some effort. Instead, we use a single *DHCP master key*, a concept adopted from [8]. From this master key we derive for each host the corresponding HMAC key by means of a key derivation function. The master key is known only to the local CA and the DHCP server. The pair $<$*MAC address, expiration time of the base certificate*$>$, in the following denoted as *client ID*, serves as the derivation parameter. This scheme does not require to inform the DHCP server about any newly certified hosts.

Since all hosts include their client ID into every sent DHCP packet, the DHCP server can deduce the corresponding HMAC key in an ad-hoc fashion and authenticate the packet. Conversely, when the DHCP server responds to the client, it has the right HMAC key already available. By incorporating the expiration time of the base certificate into the client ID we restrict the lifetime of the HMAC key. DHCP packets with expired client IDs are thus easily dropped without further verification. This allows, for instance, to immediately ignore DHCPDISCOVER messages sent by no longer authorized hosts.

To protect against replay attacks, we employ the same technique already introduced with broadcast packets, i. e., a UNIX timestamp in conjunction with a counter for packets bearing the same timestamp. A consistency check of the MAC and IP addresses stated in the DHCP header renders the authentication complete.

6.3 Further Security Measures

We consider the two DHCP messages DHCPDECLINE and DHCPRELEASE as a security risk. The first one allows a malicious client to spuriously tell the DHCP server that the IP address assigned to it is already in use by some other machine, thus making a DHCP starvation attack possible. The second one is utilized to release an assigned IP address to the DHCP server's address pool before the corresponding lease has expired. However, we cannot force a host to give up its certificate, and a malicious user might continue to use its certificate and with it the released IP address, while the address has also been assigned to some other machine. Therefore, we decided to simply drop these messages. Note that this does not violate the DHCP specification: these messages are transmitted in an unreliable manner without any retransmissions, i. e., they may get lost en route anyway. Moreover, no host is obliged to release its IP address ahead of time.

CLL allows to restrict the number of authenticated DHCP packets originated by the same host that the DHCP server will process during a specified time interval. Thereby the server can be secured against overload caused by malicious or misconfigured clients, attempting to renew their IP address lease extremely often. This would force the server to continuously issue new host certificates, which includes an expensive private key operation.

These security measures prevent malicious behavior originating from authenticated hosts. Without them attacks on DHCP would be still feasible and one would have to extensively analyze the server's DHCP logfiles to backtrack the identity of the attacker.

7 Implementation and Evaluation

7.1 CLL as a Cross-Platform Service

We have implemented CLL in C++ as a *user-mode service* on both Windows (2000, XP, 2003, Vista) and Linux (kernels 2.4 and 2.6) using Visual C++ 2005 and GNU GCC 4.x respectively. Our CLL implementation consists of a platform independent core, which interoperates with a tailored portability layer providing a consistent interface for OS specific functionality. The responsibilities of the portability layer include crafting and filtering raw Ethernet frames, configuring the network interface (ARP, IP, MTU), and the interfaces for threads and timers.

To set up a filter handler for Ethernet frames in user-mode, we employ the packet filtering framework *WinpkFilter* [21] on Windows. On Linux, we have implemented a link layer filtering solution on our own. We unbind the real network adapter from the IP stack, transparently replace it with a virtual one (a tap device), and set up a raw PF_PACKET socket to send and receive Ethernet frames through the unbound real network adapter. A maybe somewhat more efficient kernel-level implementation of CLL's packet processing engine would constitute a complex and error-prone task, especially when targeting multiple platforms. We therefore leave it for future work. But despite the overhead of additional context switches, our user-mode approach achieves good performance, and is able to operate at wire-speed in 100 Mbit LANs. To support the large number of cryptographic algorithms proposed for CLL, we employ the comprehensive open source crypto library *Botan* [15].

Aiming to provide a real-world solution, we address in our implementation such issues like persistent storage of SA configurations (to tolerate an OS reboot) and backward compatibility. To support non-CLL capable devices like network printers or NAS and to enable a step-by-step migration, CLL can be configured to communicate with legacy hosts in the standard, insecure fashion. This is accomplished by providing the CLL-enabled hosts with a list of the legacy IP/MAC address pairs. CLL then sets up static ARP entries and thereby provides at least an unidirectional protection against ARP spoofing.

Since the drivers of common Wi-Fi adapters exhibit an Ethernet-compatible interface to the network stack, Wi-Fi networks can be secured by CLL as well.

Table 2. Performance of the ARP handshake.

action	*duration in ms*
1st ping A → B using CLL: ARP handshake	27.4
1st ping A → B without CLL: usual ARP exchange	0.92
generating the private & public DH value (2048 bits)	host A: 26.3 host B: 44.1
deriving the master key with DH	host A: 7.2 host B: 15.7
computing an RSA-1024 signature	host A: 3.1 host B: 5.7

7.2 Performance Evaluation

We have conducted a performance evaluation with two hosts A and B, where A is a laptop equipped with an AMD64 Turion 1.8 GHz CPU running Linux 2.6.20 (32-bit) and B is a PC with an Intel Core 2 Duo E6400 2.13 GHz processor running Windows XP SP-2. The presented results are averaged over multiple runs.

The first series of measurements, shown in Table 2, is devoted to the overhead of the ARP handshake. For digital signatures both hosts use an RSA-1024 module. By pinging the neighboring host with no previously established SA we measure the time to perform the ARP handshake and the subsequent ICMP echo exchange. We compare it to the delay of the first ping in an ordinary unsecured setup, including a plain ARP message exchange.

Though it takes 30 times longer than a usual ARP exchange, the one-time delay of 27.4 ms induced by the ARP handshake with CLL is negligibly short for practical purposes. This low value is achieved due to an optimization in our implementation: we precompute the Diffie-Hellman values in a background thread, and thus have them readily available at the beginning of an ARP handshake. Otherwise the handshake would last $26.3 + 44.1 = 70.4$ ms longer. The delay of 27.4 ms can be broken down by measuring the computation time of the two dominating operations—the derivation of the master key with Diffie-Hellman and the creation of an RSA signature[3]. Deriving the master key is performed in parallel, thus taking $\max\{7.2, 15.7\} = 15.7$ ms, while signing is carried out sequentially and requires $3.1 + 5.7 = 8.8$ ms. Summing this up yields 24.5 ms. The remaining 2.9 ms are used for by the verification of the host certificates and handshake signatures, and also include the network round-trip time (RTT).

In the second series of measurements, we analyze the TCP throughput (using the tool *ttcp* [22]), the CPU load incurred at the sender and receiver, and the RTT between two hosts already sharing an SA. The results are shown in Table 3. When comparing the TCP throughput achievable with CLL to the result using a conventional, unsecured protocol stack, we observe only a very small decrease in speed of approximately 2 % without encryption and 3 % with encryption. It can be attributed quite exactly to the overhead induced by the additional CLL headers and fields. Encryption and authentication of packets with CLL apparently has virtually no effect on the achievable data rate in 100 Mbit LANs, which proves the feasibility of our approach.

[3] Though host B's CPU is faster than host A's CPU, the public key operations are slowed down by missing big integer assembler optimizations in Botan on Windows platforms.

Table 3. Performance of unicast transmissions in a 100 Mbit LAN.

action	*measured values*
TCP throughput using CLL:	
• *HMAC(MD5)*	A → B: 11 263 KB/s 55 / 26 % CPU (tx / rx)
	B → A: 11 312 KB/s 22 / 60 % CPU (tx / rx)
• *Twofish / HMAC(MD5)*	A → B: 11 113 KB/s 75 / 38 % CPU (tx / rx)
	B → A: 11 160 KB/s 31 / 76 % CPU (tx / rx)
TCP throughput without CLL	A → B: 11 522 KB/s 45 / 17 % CPU (tx / rx)
	B → A: 11 519 KB/s 10 / 44 % CPU (tx / rx)
RTT: 100 pings $A \rightarrow B$ using CLL	min: 0.287 ms ∅: 0.377 ms max: 0.501 ms σ: 0.046 ms
RTT: 100 pings $A \rightarrow B$ without CLL	min: 0.178 ms ∅: 0.198 ms max: 0.231 ms σ: 0.012 ms

By comparing the CPU utilization with and without CLL being used, we assess the induced additional CPU load. The overhead of piping the packets through the user-mode and computing the HMAC turns out to be entirely admissible. Even when enabling the block cipher, host A still has a quarter of its CPU time left for other tasks when processing packets at full wire-speed. The faster host B runs with a CPU utilization of only one third in the same situation. This machine obviously has the potential to operate CLL even in a Gigabit LAN, and to achieve a throughput of at least some hundred Mbit/s. Just like the TCP throughput, the RTT measured when running CLL in the Twofish / HMAC(MD5) configuration is very satisfactory. On average it is 0.38 ms, i. e., only twice the ordinary RTT without CLL. It should thus not represent a drawback for any typical application scenario.

8 Conclusion

In this paper, we have introduced the *Cryptographic Link Layer (CLL)*. CLL employs public key cryptography to identify all hosts in the Ethernet LAN based on their IP/MAC address pairs. It safeguards the packets transmitted between them against different spoofing attacks and eavesdropping. Pairs of hosts willing to communicate first establish security associations by an extension of the ARP handshake. In the course of this, the hosts authenticate each other, exchange cryptographic parameters, and negotiate symmetric session keys to protect their following unicast packets with a message authentication code and an optional cipher. Broadcast packets are also secured by CLL using digital signatures. When IP addresses are to be configured dynamically, CLL extends DHCP to automatically issue host certificates with the leased IP address. In the course of this, it also adds authentication to DHCP and safeguards it against various attacks.

We have implemented CLL on both Windows and Linux without modifying the existing protocol stack. Backward compatibility to ordinary, unsecured hosts can be enabled to support a step-by-step migration and retain legacy devices. The evaluation of CLL demonstrated the excellent performance of our protocol in a 100 Mbit Ethernet LAN, where it achieved wire-speed throughput and short round-trip times.

References

[1] Hayriye Altunbasak, Sven Krasser, Henry Owen, Joachim Sokol, Jochen Grimminger, and Hans-Peter Huth. Addressing the Weak Link Between Layer 2 and Layer 3 in the Internet Architecture. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 417–418, November 2004.

[2] Antidote. `http://antidote.sourceforge.net`.

[3] ArpWatch. `http://ee.lbl.gov` and `http://freequaos.host.sk/arpwatch`.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Message Authentication Using Hash Functions: the HMAC Construction. *RSA CryptoBytes*, 2(1), 1996.

[5] D. Bruschi, A. Ornaghi, and E. Rosti. S-ARP: a Secure Address Resolution Protocol. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, pages 66–74, December 2003.

[6] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, April 2006.

[7] R. Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997.

[8] R. Droms and W. Arbaugh. Authentication for DHCP Messages. RFC 3118, June 2001.

[9] Ettercap. `http://ettercap.sourceforge.net`.

[10] Mohamed G. Gouda and Chin-Tser Huang. A secure address resolution protocol. *Computer Networks*, 41(1):57–71, 2003.

[11] IEEE 802.1AE. Media Access Control (MAC) Security. `http://www.ieee802.org/1/pages/802.1ae.html`.

[12] Yves Igor Jerschow. The CLL service & toolkit for Windows and Linux. `http://www.cn.uni-duesseldorf.de/projects/CLL`.

[13] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, December 2005.

[14] Hugo Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In *CRYPTO 2001: Proceedings of the 21st Annual International Cryptology Conference*, pages 310–331, August 2001.

[15] Jack Lloyd. Botan Cryptographic Library. `http://botan.randombit.net`.

[16] Wesam Lootah, William Enck, and Patrick McDaniel. TARP: Ticket-based Address Resolution Protocol. *Computer Networks*, 51(15):4322–4337, 2007.

[17] David L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. RFC 1305, March 1992.

[18] Massimiliano Montoro. Cain & Abel. `http://www.oxid.it/cain.html`.

[19] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Song. The TESLA Broadcast Authentication Protocol. *RSA CryptoBytes*, 5(2):2–13, 2002.

[20] David C. Plummer. Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. RFC 826, November 1982.

[21] NT Kernel Resources. WinpkFilter. `http://www.ntkernel.com`.

[22] Test TCP (TTCP) - Benchmarking Tool for Measuring TCP and UDP Performance. `http://www.pcausa.com/Utilities/pcattcp.htm`.

[23] Eric Vyncke and Christopher Paggen. *LAN Switch Security*. Cisco Press, 2007.

[24] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, January 2006.