University of Glasgow

Secure Communications

Networked Systems (H) – Lecture 3

2019-2020

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow | This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/4.0/ or send a letter to Creative Commons, PO Box 1866,

The Need for Secure Communications

- Protecting Privacy
- Protecting Message Integrity
- Preventing Protocol Ossification

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Protecting Privacy

- Numerous organisation monitor Internet traffic:
  - Governments, intelligence agencies, and law enforcement
    - Good reasons to monitor *some* traffic – but Edward Snowden showed that pervasive monitoring of *all* traffic is widespread
  - Businesses
    - Regulatory requirements to record some traffic – e.g., banking
    - “Your call may be monitored for quality and training purposes”
    - To profile customers for advertising
  - Network operators
    - To support network operations and trouble-shooting
    - To profile customers for advertising
  - Criminals and malicious users
    - Steal data and user credentials; identity theft; active attacks
- Mechanisms that protect privacy against malicious attackers will also prevent benign monitoring
  - No known way to stop criminals and malicious attackers from accessing private data that doesn’t also stop law enforcement



Edward Snowden

K. Moriarty and A. Morton, “Effect of Pervasive Encryption on Operators”, IETF, July 2018, RFC 8404 https://tools.ietf.org/html/rfc8404

G. Fairhurst and C. Perkins, “Considerations around Transport Header Confidentiality, Network Operations, and the Evolution of Internet Transport Protocols”, IETFJanuary 2020, Work in progress, https://datatracker.ietf.org/doc/draft-ietf-tsvwg-transport-encrypt/

S. Farrell and H. Tschofenig, “Pervasive Monitoring is an Attack”, IETF, May 2014, RFC 7258 https://tools.ietf.org/html/rfc7258

CC BY ND

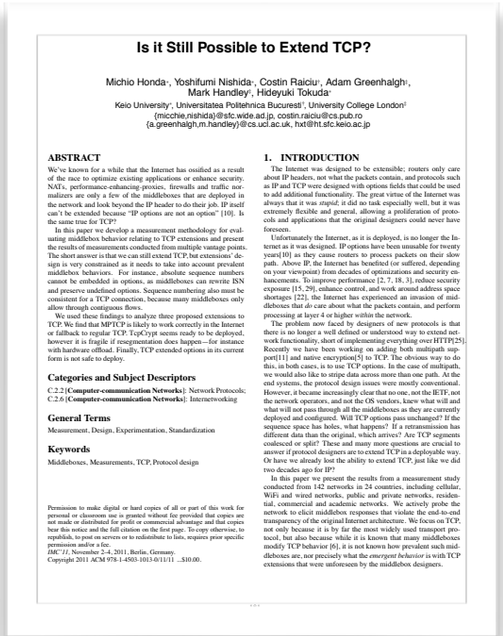
Protecting Message Integrity

- Numerous organisations may want to tamper with message integrity
  - Governments, intelligence agencies, and law enforcement
    - Many governments require ISPs to censor or modify DNS responses
    - Many governments require ISPs to block messages containing certain content
  - Businesses and network operators
    - To enforce legal restrictions on content
    - To enforce terms of service
    - To prevent copyright infringement
  - Criminals and malicious users
    - Phishing scams and identity theft
- Mechanisms that protect message integrity from malicious attackers will also prevent benign actors
  - e.g., web browsers deployed DNS-over-HTTPS (DoH) to ensure integrity of DNS responses and protect users against phishing attacks that modify DNS replies – but this prevented ISPs from modifying DNS replies to block access to domains hosting illegal material
  - No technical way to distinguish beneficial integrity violations from malicious attacks

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Preventing Protocol Ossification

- Network operators deploy **middleboxes** to monitor and/or modify network traffic:
  - NATs, firewalls, monitors, traffic shaping, filtering, etc.
  - Some are necessary and beneficial, others perhaps less so
- Middleboxes must understand protocols used by the network traffic:
  - e.g., NAT has to know where to find IP addresses and ports in packets, if it is to translate them
  - e.g., traffic shapers, that limit TCP throughput for users exceeding their monthly bandwidth use cap, need to parse TCP packets and know how to change them to influence congestion control
- This can lead to **protocol ossification** – cannot change end-to-end protocols, because doing so interacts poorly with middleboxes that don't understand the changes
  - e.g., TCP SACK, window scaling, multipath, …
- The more of a protocol that is encrypted, the easier it is to change – since middleboxes cannot understand or modify the data – but the harder it is for middleboxes to provide useful services



Is it Still Possible to Extend TCP?

Michio Honda[•], Yoshifumi Nishida[•], Costin Raiciu[◦], Adam Greenhalgh[◦], Mark Handley[◦], Hideyuki Tokuda[•]
Keio University[•], Universitatea Politehnica Bucuresti[◦], University College London[◦]
{michie,nishida}@sfc.wide.ad.jp, costin.raiciu@cs.pub.ro
{a.greenhalgh,m.handley}@cs.ucl.ac.uk, hxt@ht.sfc.keio.ac.jp

ABSTRACT

We've known for a while that the Internet has ossified as a result of the race to optimize existing applications or enhance security. NATs, performance-enhancing proxies, firewalls and traffic normalizers are only a few of the middleboxes that are deployed in the network and look beyond the IP header to do their job. IP itself can't be extended because "IP options are not an option" [10]. Is the same true for TCP?

In this paper we develop a measurement methodology for evaluating middlebox behavior relating to TCP extensions and present the results of measurements conducted from multiple vantage points. The short answer is that we can still extend TCP, but extensions' design is very constrained as it needs to take into account prevalent middlebox behaviors. For instance, absolute sequence numbers cannot be embedded in options, as middleboxes can rewrite ISN and preserve undefined options. Sequence numbering also must be consistent for a TCP connection, because many middleboxes only allow through contiguous flows.

We used these findings to analyze three proposed extensions to TCP. We find that MPTCP is likely to work correctly in the Internet or fallback to regular TCP. TcpCrypt seems ready to be deployed, however it is fragile if resegmentation does happen—for instance with hardware offload. Finally, TCP extended options in its current form is not safe to deploy.

Categories and Subject Descriptors

C.2.2 [**Computer-communication Networks**]: Network Protocols; C.2.6 [**Computer-communication Networks**]: Internetworking

General Terms

Measurement, Design, Experimentation, Standardization

Keywords

Middleboxes, Measurements, TCP, Protocol design

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IMC'11, November 2–4, 2011, Berlin, Germany.
Copyright 2011 ACM 978-1-4503-1013-0/11/11 ...$10.00.

1. INTRODUCTION

The Internet was designed to be extensible; routers only care about IP headers, not what the packets contain, and protocols such as IP and TCP were designed with options fields that could be used to add additional functionality. The great virtue of the Internet was always that it was *stupid*; it did no task especially well, but it was extremely flexible and general, allowing a proliferation of protocols and applications that the original designers could never have foreseen.

Unfortunately the Internet, as it is deployed, is no longer the Internet as it was designed. IP options have been unusable for twenty years[10] as they cause routers to process packets on their slow path. Above IP, the Internet has benefited (or suffered, depending on your viewpoint) from decades of optimizations and security enhancements. To improve performance [2, 7, 18, 3], enhance security exposure [15, 29], enhance control, and work around address space shortages [22], the Internet has experienced an invasion of middleboxes that *do* care about what the packets contain, and perform processing at layer 4 or higher within the network.

The problem now faced by designers of new protocols is that there is no longer a well defined or understood way to extend network functionality, short of implementing everything over HTTP[25]. Recently we have been working on adding both multipath support[11] and native encryption[5] to TCP. The obvious way to do this, in both cases, is to use TCP options. In the case of multipath, we would also like to stripe data across more than one path. At the end systems, the protocol design issues were mostly conventional. However, it became increasingly clear that no one, not the IETF, not the network operators, and not the OS vendors, knew what will and what will not pass through all the middleboxes as they are currently deployed and configured. Will TCP options pass unchanged? If the sequence space has holes, what happens? If a retransmission has different data than the original, which arrives? Are TCP segments coalesced or split? These and many more questions are crucial to answer if protocol designers are to extend TCP in a deployable way. Or have we already lost the ability to extend TCP, just like we did two decades ago for IP?

In this paper we present the results from a measurement study conducted from 142 networks in 24 countries, including cellular, WiFi and wired networks, public and private networks, residential, commercial and academic networks. We actively probe the network to elicit middlebox responses that violate the end-to-end transparency of the original Internet architecture. We focus on TCP, not only because it is by far the most widely used transport protocol, but also because while it is known that many middleboxes modify TCP behavior [6], it is not known how prevalent such middleboxes are, nor precisely what the *emergent behavior* is with TCP extensions that were unforeseen by the middlebox designers.

M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?", Proceedings of the ACM Internet Measurement Conference, Berlin, Nov. 2011. https://dx.doi.org/10.1145/2068816.2068834

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Security and Policy

- Strong reasons to protect privacy, to ensure integrity of messages and prevent modification in transit, and to prevent protocol ossification
- The consequences of this protection are sometimes problematic
- Dialogue between engineers, protocol designers, network operators, policy makers, and law enforcement is crucial – to understand constraints and concerns



Keys Under Doormats:

MANDATING INSECURITY BY REQUIRING GOVERNMENT ACCESS TO ALL DATA AND COMMUNICATIONS

Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matthew Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Peter G. Neumann, Susan Landau, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael Specter, Daniel J. Weitzner

Abstract

Twenty years ago, law enforcement organizations lobbied to require data and communication services to engineer their products to guarantee law enforcement access to all data. After lengthy debate and vigorous predictions of enforcement channels "going dark," these attempts to regulate the emerging Internet were abandoned. In the intervening years, innovation on the Internet flourished, and law enforcement agencies found new and more effective means of accessing vastly larger quantities of data. Today we are again hearing calls for regulation to mandate the provision of exceptional access mechanisms. In this report, a group of computer scientists and security experts, many of whom participated in a 1997 study of these same topics, has convened to explore the likely effects of imposing extraordinary access mandates.

We have found that the damage that could be caused by law enforcement exceptional access requirements would be even greater today than it would have been 20 years ago. In the wake of the growing economic and social cost of the fundamental insecurity of today's Internet environment, any proposals that alter the security dynamics online should be approached with caution. Exceptional access would force Internet system developers to reverse "forward secrecy" design practices that seek to minimize the impact on user privacy when systems are breached. The complexity of today's Internet environment, with millions of apps and globally connected services, means that new law enforcement requirements are likely to introduce unanticipated, hard to detect security flaws. Beyond these and other technical vulnerabilities, the prospect of globally deployed exceptional access systems raises difficult problems about how such an environment would be governed and how to ensure that such systems would respect human rights and the rule of law.

H. Abelson, *et al.*, "Keys under doormats: Mandating insecurity by requiring government access to all data and communications", MIT Computer Science and Artificial Intelligence Lab, technical report MIT-CSAIL-TR-2015-026, July 2015. http://dspace.mit.edu/handle/1721.1/97690

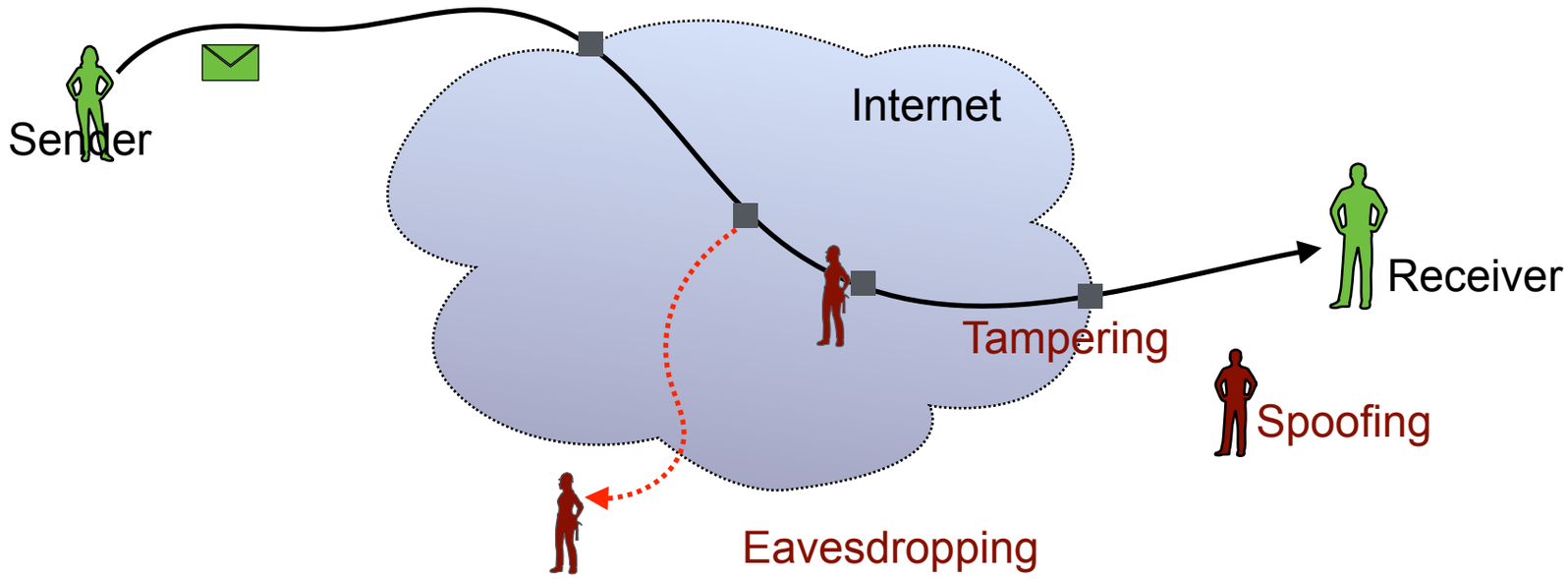
Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Principles of Secure Communication

- Ensuring Confidentiality
- Authenticating Messaging
- Validating Identity

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

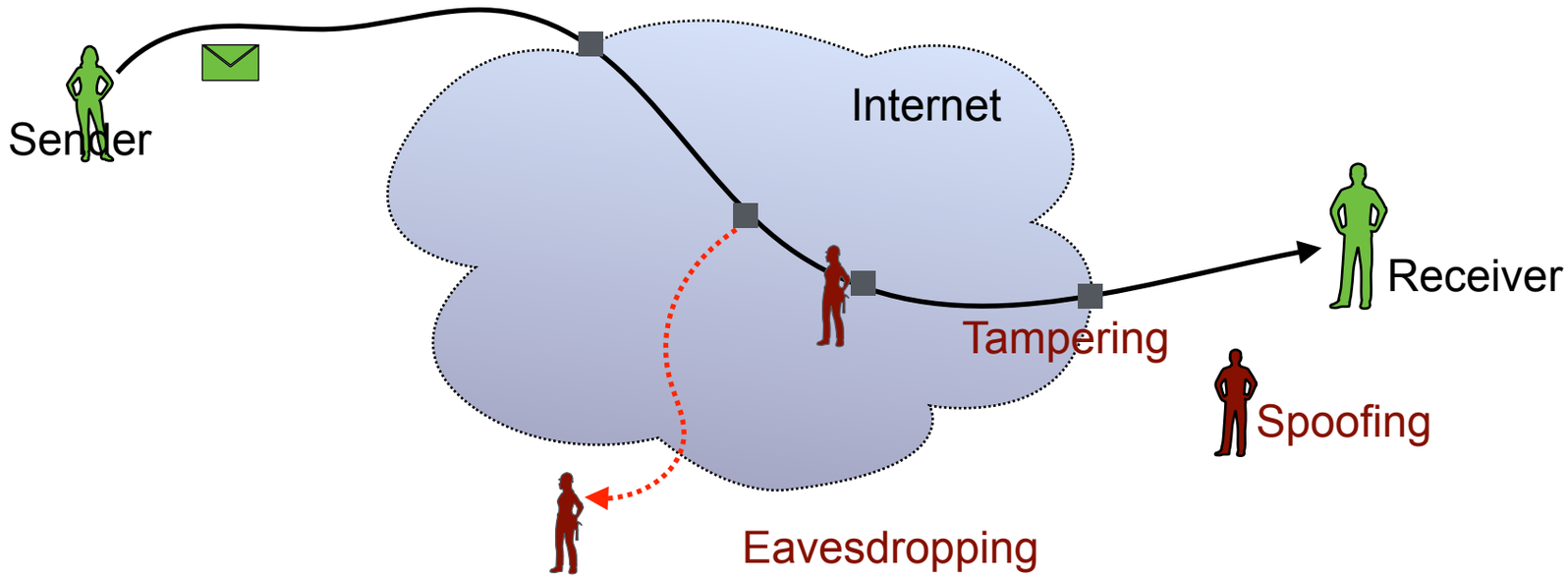
Principles of Secure Communication



- Goal: deliver message from sender to receiver
  - Avoid eavesdropping – confidentiality
  - Avoid tampering – integrity protection
  - Avoid spoofing – trust

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Confidentiality (1/2)



- Data traversing the network can be read by any device on the path
  - Can eavesdrop on packets as they traverse a link
  - Can break into a switch or router, snoop on data as it's forwarded between links
- The network operator can *always* do this; if their network has been compromised, maybe so can others

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

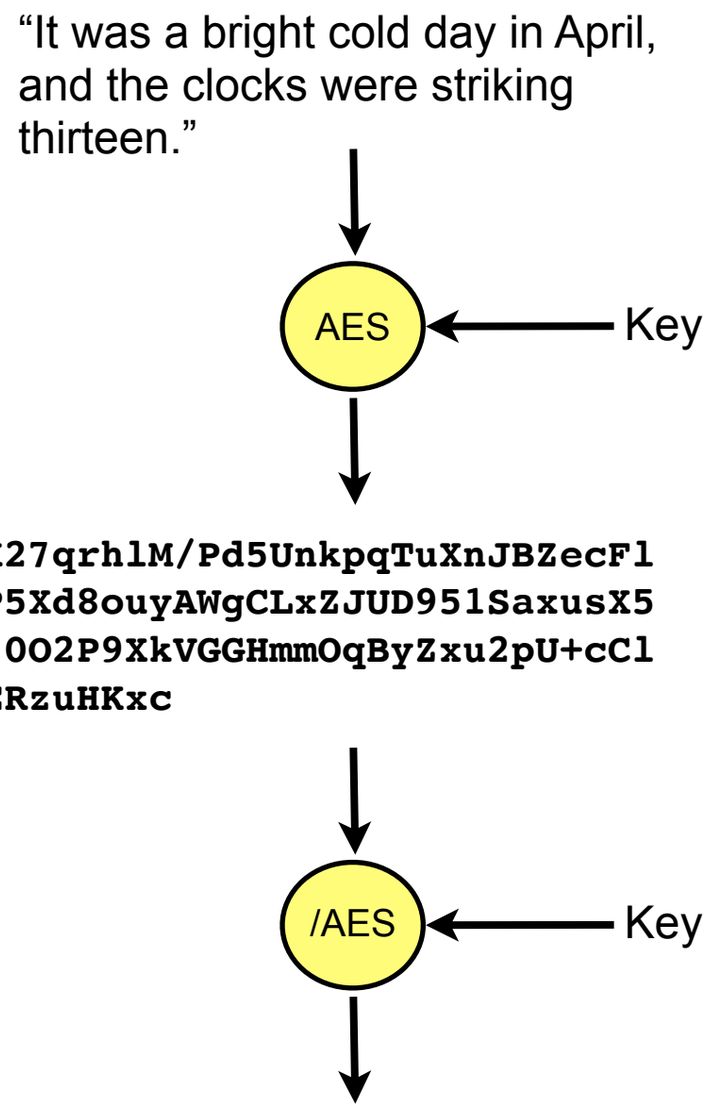
Confidentiality (2/2)

- Data can always be read; encrypt it to make it useless if intercepted
- Two basic approaches
  - Symmetric cryptography
    - Advanced Encryption Standard (AES)
  - Public key cryptography
    - The Diffie-Hellman algorithm
    - The Rivest-Shamir-Adleman (RSA) algorithm
    - Elliptic curve-based algorithms
- Complex mathematics – will not attempt to describe

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Symmetric Cryptography

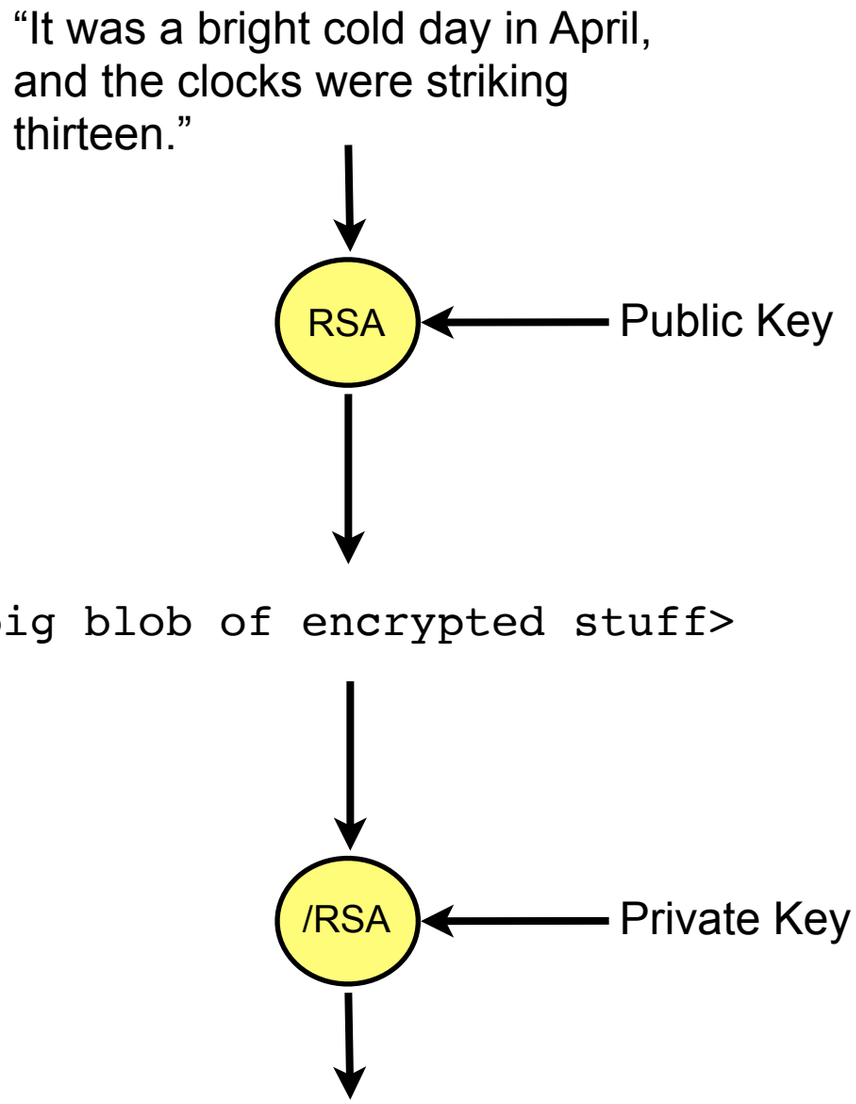
- Symmetric encryption converts plain text into cipher-text
  - Encryption and decryption algorithms are public
    - US Advanced Encryption Standard (AES) is widely used – based on Rijndael algorithm, developed by Vincent Rijmen and Joan Daemen
      - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
  - A secret **key** control the encryption and decryption process
    - Same key used to encrypt as is used to decrypt
    - Three variants of AES; keys of 128, 192, or 256 bits
    - Provided the key is secret and only known to sender and receiver, the conversation is secure – problem: **how to securely distribute the key?**
  - Very fast – suitable for bulk encryption



Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Public Key Cryptography

- Public key encryption also converts plain text into cipher-text
  - Again, the algorithms are public:
    - Diffie–Hellman
    - Rivest–Shamir–Adleman (RSA)
    - Ephemeral Elliptic Curve Diffie-Hellman
  - Public key algorithms use two related keys:
    - The **public key** for a user is widely distributed
    - **The corresponding private key must be kept secret**
    - If one key is used to encrypt, the other key is needed to decrypt the message
- Solves key distribution problem
  - Look-up the public key of the receiver in a directory
  - Sender uses the public key to encrypt the message → can only be decrypted by the private key
  - If receiver is trusted to keep private key secret, only it can decrypt the message
- Problem: very slow to encrypt and decrypt



Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Hybrid Cryptography

- Modern communications use a combination of **both** public-key and symmetric cryptography for security and speed
  - Sender chooses a random value, K_s, that can be used as key for the symmetric encryption algorithm
  - Sender looks up the receiver's public key, K_{pub}, uses it to encrypt K_s, and sends the result to the receiver; receiver uses the corresponding private key, K_{priv}, to decrypt the message and retrieve K_s
    - Securely transfers K_s from sender to receiver
    - Public key encryption is very slow, but the key K_s is small, so this doesn't matter
  - Sender encrypts future messages using symmetric cryptography with key K_s, receiver also has k_s, which it uses to decrypt the messages
    - Symmetric cryptography is fast, but requires the key to be exchanged securely
    - The public key algorithm has been used to securely exchange the key
- Ensures confidentiality of communication with good performance

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Authentication

- Encryption can ensure confidentiality – but also need to verify identify of sender and ensure messages have not been modified in transit
  - Generate a **digital signature** to authenticate the message
  - Relies on public key cryptographic and a cryptographic hash

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

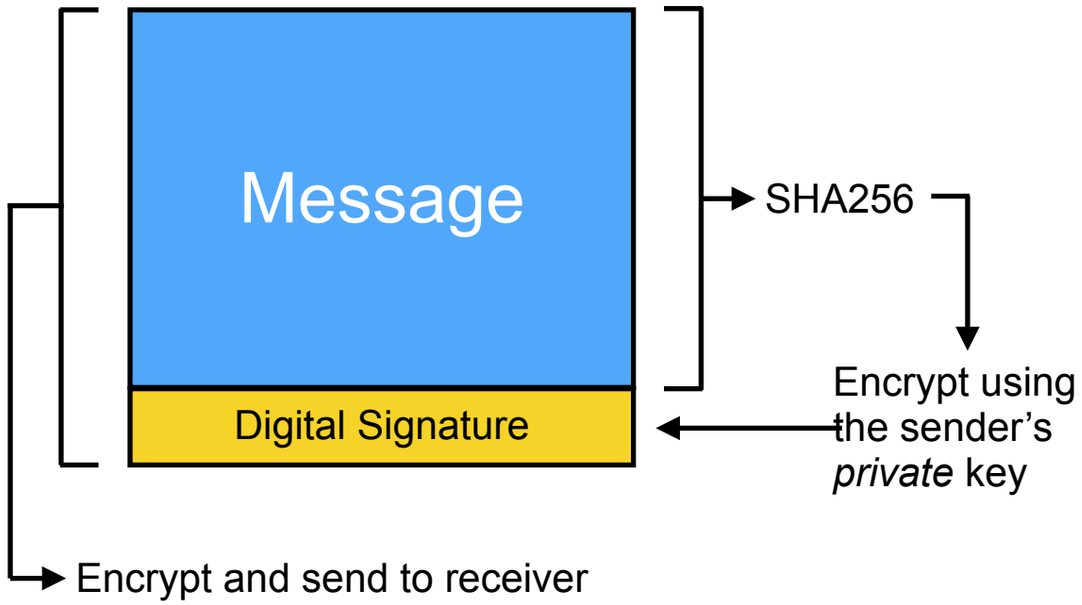
Cryptographic Hash Functions

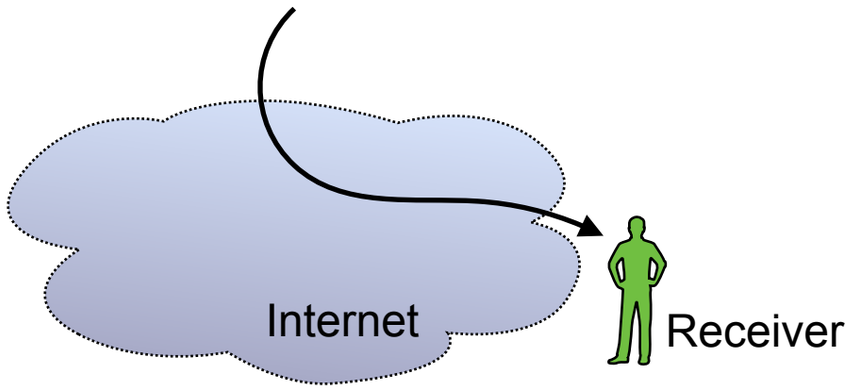


- A **cryptographic hash** function takes arbitrary input and produces a fixed length output hash of the input:
  - The output hash is always the same size (e.g., 256 bits), irrespective of the size of the input
  - It is extremely easy to calculate the output hash for any given input
  - Any change to the input will generate a different output
  - It is computationally infeasible to determine what was the input, given only the output
  - It is computationally infeasible to find two inputs that give the same output
- The algorithm used to calculate the cryptographic hash should be public:
  - SHA256 algorithm https://tools.ietf.org/html/rfc6234
  - Older algorithms, e.g., MD5 and SHA1, have known security flaws

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Digital Signatures (1/2)

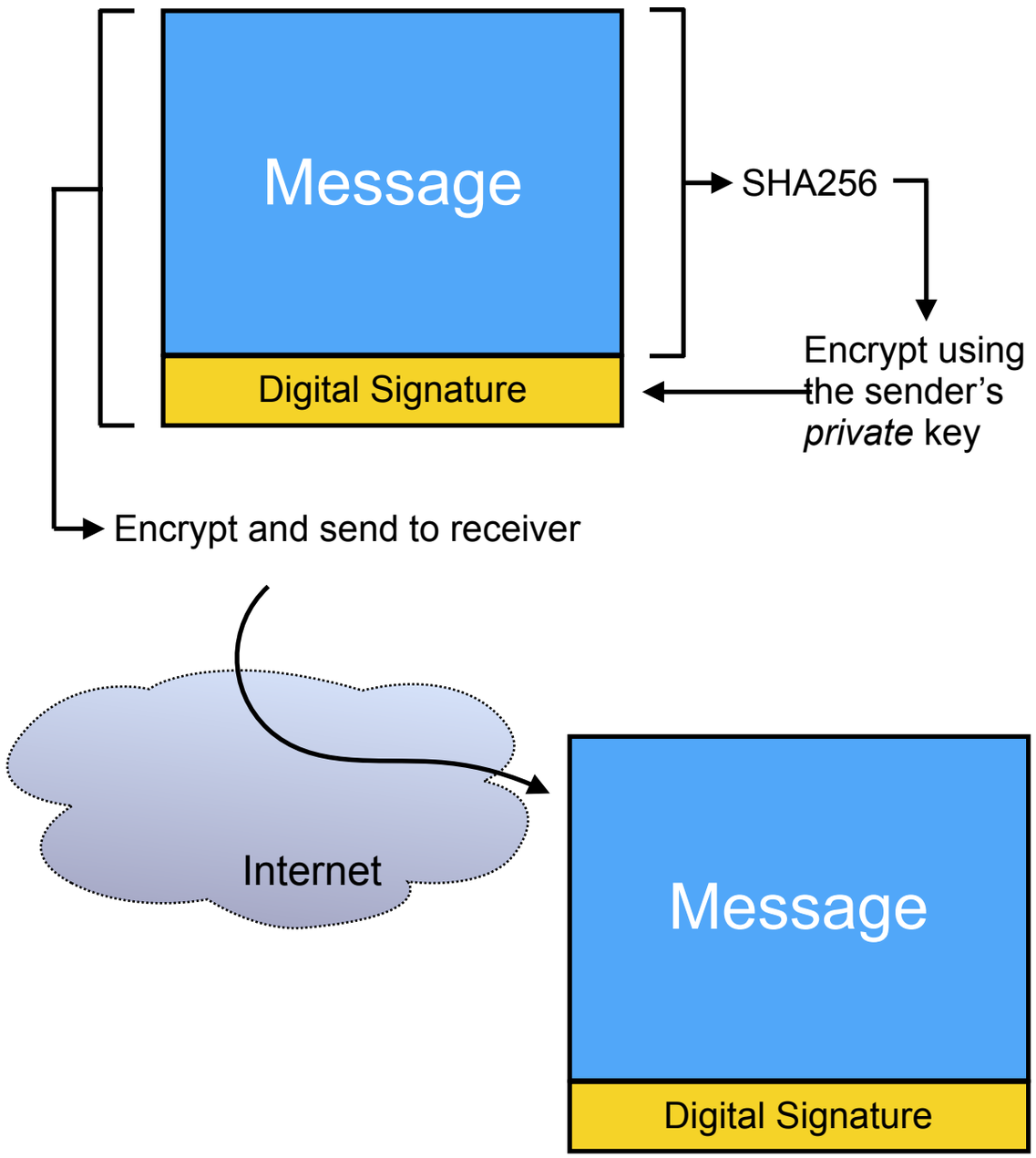




- Sender generates a digital signature
  - Sender calculates the cryptographic hash of the message
  - Sender encrypts the hash with their own *private* key
    - Anyone can use the sender's public key to decrypt this, but only the sender can have encrypted it (if trusted to keep their private key secret)
  - Attaches encrypted hash to the message
- Message and its digital signature are encrypted and sent to receiver using hybrid encryption

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Digital Signatures (2/2)

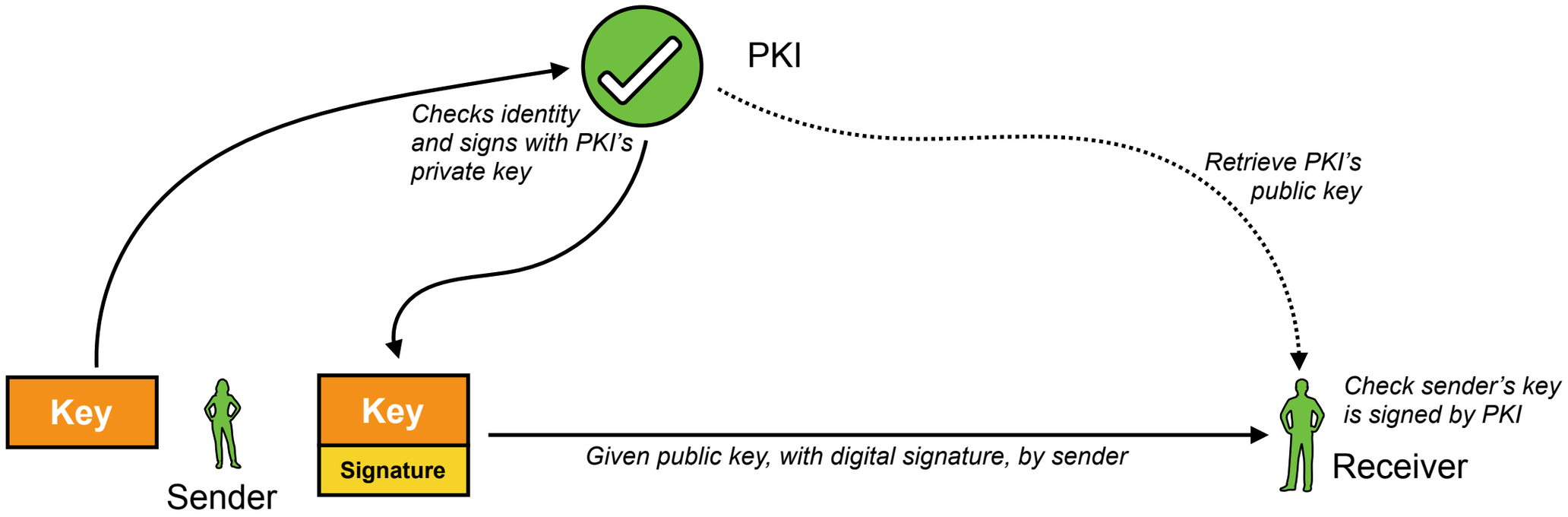


- Signature verification process:
  - Receiver decrypts the message
  - Receiver calculate cryptographic hash of the message
  - Receiver decrypts digital signature using sender's public key, to find the hash that the sender calculates
  - If hash decrypted hash and the hash calculated by the receiver match, the message is authentic and unmodified – provided the sender kept its private key secret

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Trust and Public Key Infrastructure

- How to know what public key corresponds to a particular receiver?
  - The receiver gave you their key in person
  - The receiver sent you their key, authenticated by someone you trust
  - Someone you trust gave you the receiver's key
- A **public key infrastructure** can authenticate keys
  - PKI verifies sender's identity, then adds their digital signature to sender's public key
  - If receiver trusts PKI, can verify the digital signature to confirm identity of sender



Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Transport Layer Security (TLS) v1.3

- What is TLS?
- TLS handshake protocol
- TLS record protocol
- TLS 0-RTT mode
- Limitations

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Transport Layer Security (TLS) v1.3

- TCP is not secure – neither the TCP/IP headers nor the data are encrypted or authenticated
- The **Transport Layer Security** (TLS) v1.3 protocol can be used to encrypt and authenticate data carried within a TCP connection
  - Official specification:
    - https://tools.ietf.org/html/rfc8446 – The Transport Layer Security (TLS) Protocol Version 1.3
  - Useful blog posts:
    - https://ietf.org/blog/tls13/ – TLS 1.3
    - https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/ – A Detailed Look at TLS 1.3
    - https://davidwong.fr/tls13/ – A Readable Specification of TLS 1.3
    - https://tls13.ulfheim.net – The New Illustrated TLS Connection

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Goals

The primary goal of TLS is to provide a secure channel between two communicating peers; the only requirement from the underlying transport is a reliable, in-order data stream. Specifically, the secure channel should provide the following properties:

- **Authentication: The server side of the channel is always authenticated; the client side is optionally authenticated. Authentication can happen via asymmetric cryptography (e.g., RSA [RSA], the Elliptic Curve Digital Signature Algorithm (ECDSA) [ECDSA], or the Edwards-Curve Digital Signature Algorithm (EdDSA) [RFC8032]) or a symmetric pre-shared key (PSK).**

- **Confidentiality: Data sent over the channel after establishment is only visible to the endpoints. TLS does not hide the length of the data it transmits, though endpoints are able to pad TLS records in order to obscure lengths and improve protection against traffic analysis techniques.**

- **Integrity: Data sent over the channel after establishment cannot be modified by attackers without detection.**

E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, IETF, August 2018, RFC 8446, https://tools.ietf.org/html/rfc8446

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Overview

TLS consists of two primary components:

- **A handshake protocol (Section 4) that authenticates the communicating parties, negotiates cryptographic modes and parameters, and establishes shared keying material. The handshake protocol is designed to resist tampering; an active attacker should not be able to force the peers to negotiate different parameters than they would if the connection were not under attack.**

- **A record protocol (Section 5) that uses the parameters established by the handshake protocol to protect traffic between the communicating peers. The record protocol divides traffic up into a series of records, each of which is independently protected using the traffic keys.**

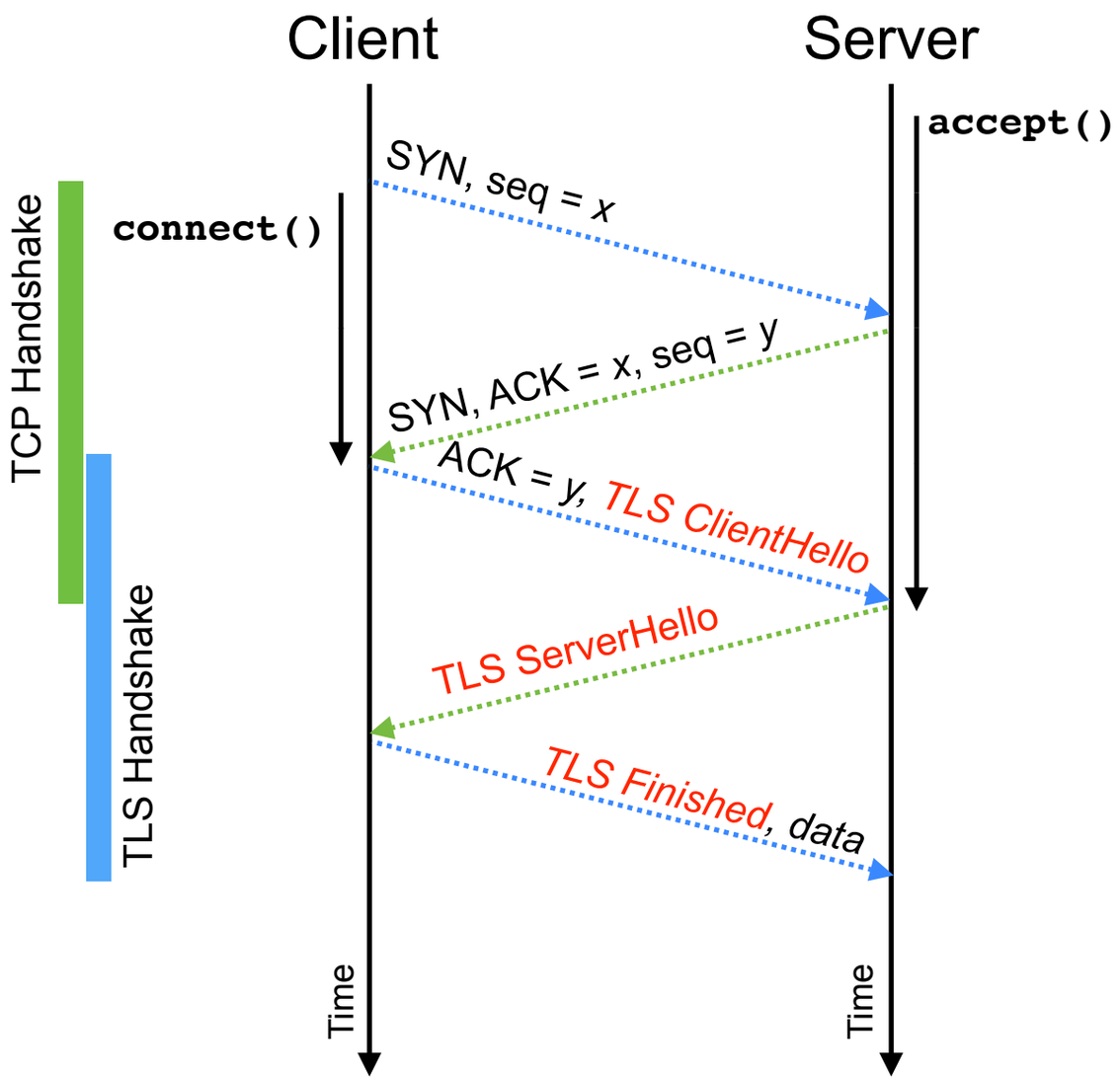
E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, IETF, August 2018, RFC 8446, https://tools.ietf.org/html/rfc8446

- An (insecure) TCP connection is established as usual
- The TLS handshake protocol runs within that TCP connection, to authenticate the endpoints and establish security keys
- Then, the TLS record protocol runs, allowing sender and receiver to exchange encrypted and authenticated blocks of data
  - TLS turns the TCP byte stream into a series of records → it provides framing

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Handshake Protocol

- TCP connection established as usual
  - **`SYN`** → **`SYN+ACK`** → **`ACK`**
- TLS handshake protocol immediately follows
  - TLS **`ClientHello`** sent with the **`ACK`**
  - TLS **`ServerHello`** sent in response
  - TLS **`Finished`** message concludes, and carries initial secure data record
- Adds 1-RTT to connection establishment



Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 **`ClientHello`**

> - **Key Exchange: Establish shared keying material and select the cryptographic parameters. Everything after this phase is encrypted.**

E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, IETF, August 2018, RFC 8446, https://tools.ietf.org/html/rfc8446

- The **`ClientHello`** message:
  - Indicates that TLS v1.3 is to be used
    - It signals TLS v1.2 in the main **`ClientHello`** message, and includes an extension header to say “I’m really TLS v1.3” because too many middleboxes break if the TLS version changes – *protocol ossification*
  - Provides the cryptographic algorithms the client supports
  - Provides the name of the server to which the client is connecting
    - If connecting to a web hosting server, it’s likely that more than one site is hosted on the same server, so need to specify which is intended
  - Does not contain any data

CC BY ND

TLS v1.3 **`ServerHello`**

> - **Server Parameters: Establish other handshake parameters (whether the client is authenticated, application-layer protocol support, etc.).**
>
> …
>
> **The server processes the ClientHello and determines the appropriate cryptographic parameters for the connection. It then responds with its own ServerHello (Section 4.1.3), which indicates the negotiated connection parameters. The combination of the ClientHello and the ServerHello determines the shared keys**

E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, IETF, August 2018, RFC 8446, https://tools.ietf.org/html/rfc8446

- The **`ServerHello`** message:
  - Indicates that TLS v1.3 is to be used
    - It signals TLS v1.2 in the main **`ServerHello`** message, and includes an extension header to say “I’m really TLS v1.3” because too many middleboxes break if the TLS version changes – *protocol ossification*
  - Provides the cryptographic algorithms selected by the server, from the set the client suggested
  - Provides the server’s public key
  - Provides the server’s certificate – i.e., the digital signature used to verify its identity
  - Does not contain any data

CC BY ND

TLS v1.3 **`Finished`**

> - **Authentication: Authenticate the server (and, optionally, the client) and provide key confirmation and handshake integrity.**

E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, IETF, August 2018, RFC 8446, https://tools.ietf.org/html/rfc8446

- The **`Finished`** message:
  - Provides the client’s public key
  - Optionally, provides the certificate needed to authenticate the client to the server
  - May contain data sent from client to server

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Cryptographic Algorithms

- Client and server use the Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) key exchange algorithm
  - Client sends its public key to server in **`ClientHello`**
  - Server sends its public key to client in **`ServerHello`**
  - Client and server both combine the two public keys to derive the key used for the symmetric cryptography
    - Complex mathematics – will not describe
  - Server provides a certificate (digital signature) to allow the client to verify its identify in the **`ServerHello`** – client can optionally provide this in **`Finished`**
- Client proposes symmetric encryption algorithms in **`ClientHello`**, server picks from them and replies in **`ServerHello`**
  - Either AES or ChaCha20 symmetric encryption

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Record Protocol

- TLS v1.3 splits the data into **records**, each containing ≤ 2^{14} bytes
  - Each record is both encrypted and authenticated; it has a sequence number
  - Can renegotiate encryption keys between records
  - TCP does not preserve record boundaries; TLS adds **framing** so that it does – reading from a TLS connection will block until a complete record is received

- Client and server exchange records, then close the connection

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 0-RTT Mode

- If a client and server have previously communicated, they can re-use the same key as before
  - The **`ClientHello`** message can include data to be delivered to the server; the **`ServerHello`** can contain a reply
  - This 0-RTT data may be delivered more than once, if the **`ClientHello`** or **`ServerHello`** messages are duplicated – TLS doesn't stop this
  - Protection against duplicate messages is provided by sequence numbers in the record layer
  - Handshake messages do not have sequence numbers, so duplicated messages can deliver data more than once
  - **Be careful writing applications that use 0-RTT mode**

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Limitations of TLS

- TLS v1.3 **is secure**, but:
  - Does not protect server name
    - Exposes hostname of server to which connection is being made
    - Encrypted SNI extension in development – difficult to deploy
  - Operates within a TCP/IP connection
    - IP addresses and TCP port numbers are not protected – exposes information about who is communicating and what application is being used
  - Relies on a PKI to validate public keys
    - Significant concerns about trustworthiness of PKI
  - May deliver 0-RTT data more than once

https://datatracker.ietf.org/doc/draft-ietf-tls-esni/

https://blog.cloudflare.com/encrypted-sni/

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Discussion

- End-to-end security
- Robustness principle
- Validating input data
- Writing secure code

CC BY ND

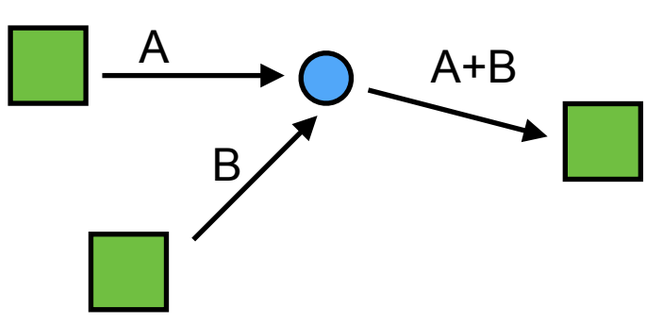
End-to-end Security? (1/2)

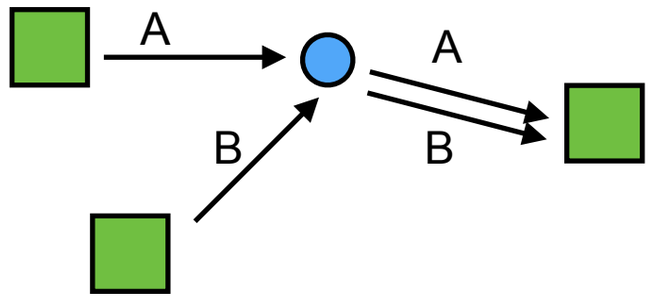
- For communication to be secure, it must be **end-to-end**
  - The two endpoints are the sender and the final recipient
  - With a data centre or CDN, what is the final recipient?
    - Is it the load balancer at the entrance to the data centre, or the server within the data centre that processes the request?
    - If the request is directed to a content distribution network (CDN), is the end the local cache that serves the request? If so, how are the encryption keys shared?
  - If the data is moving between two users, is it encrypted between the two users or between each user and the data centre?
    - i.e., can the data centre see user-to-user data flows?
    - (This is normal if you run TLS from user-to-data centre then from data centre-to-user)

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

End-to-end Security? (2/2)

- Is there in-network processing? How much data is revealed to the in-network server?
  - e.g., video conference with privacy protection vs. without
  - Does the central server decrypt the speech data, mix into one stream and send to the receiver, or does it forward all active streams in encrypted form





- Trades-off security vs bandwidth
  - For audio the bandwidth is small enough this doesn't matter
  - For video conferencing, the combined bandwidth may be significant

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

The Robustness Principle (Postel's Law)

> At every layer of the protocols, there is a general rule whose application can lead to enormous benefits in robustness and interoperability:
>
> "Be liberal in what you accept, and conservative in what you send"
>
> Software should be written to deal with every conceivable error, no matter how unlikely; sooner or later a packet will come in with that particular combination of errors and attributes, and unless the software is prepared, chaos can ensue. In general, it is best to assume that the network is filled with malevolent entities that will send in packets designed to have the worst possible effect. This assumption will lead to suitable protective design, although the most serious problems in the Internet have been caused by un-envisaged mechanisms triggered by low-probability events; mere human malice would never have taken so devious a course!
>
> RFC1122

- Balance interoperability with security – don't be *too* liberal in what you accept; a clear specification of how and when you will fail might be more appropriate

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

The Robustness Principle (Postel's Law)

"Be liberal in what you accept, and conservative in what you send"

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

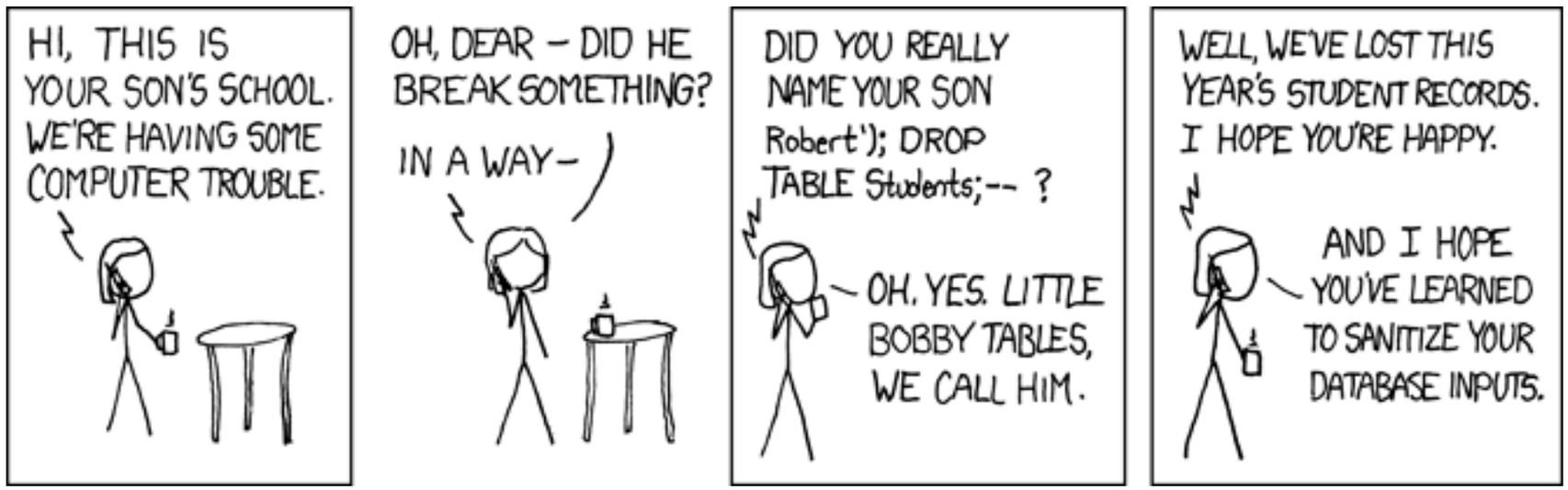
The Robustness Principle (Postel's Law)

"Postel lived on a network with all his friends.
We live on a network with all our enemies.
Postel was wrong for todays internet."
— *Poul-Henning Kamp*

See also: https://datatracker.ietf.org/doc/draft-iab-protocol-maintenance/

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Validating Input Data



http://xkcd.com/327/

- Networked applications fundamentally dealing with data supplied by un-trusted third parties
- Data read from the network may not conform to the protocol specification
- Due to ignorance and/or bugs
- Due to malice, and a desire to disrupt services
- Must carefully validate all data before use

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Writing Secure Code

- The network is hostile: any networked application is security critical
  - Must carefully specify behaviour with both correct and incorrect inputs
  - Must carefully validate inputs and handle errors
  - Must take additional care if using type- and memory-unsafe languages, such as C and C++, since these have additional failure modes
- The best encryption doesn't help if the endpoints can be compromised

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Secure Communications

- The need for secure communication
- Principles of secure communication
- TLS v1.3
- Discussion

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow