

CS 458 / 658: Computer Security and Privacy

Module 5 - Security and Privacy of Internet Applications

Part 2 - Cryptography use cases

Meng Xu (*University of Waterloo*)

Winter 2023

Outline

- 1 Overview of Security Controls
- 2 Link-layer Security (WEP, WPA)
- 3 Network-layer Security (IPSec)
- 4 Transport-layer Security (TLS)
- 5 Application-layer Security (SSH)
- 6 Application-layer Security (PGP)
- 7 Application-layer Security (OTR)

Security controls using cryptography

We use cryptography as security control in situations where **trust** cannot be assumed.

Security controls using cryptography

We use cryptography as security control in situations where trust cannot be assumed.

- We will closely examine case studies on network security
 - link layer
 - network layer
 - transport layer
 - application layer
- But first, we will see some simple use cases to warm-up.

Use cases in program and OS security

- Apps can be installed only if digitally signed by the vendor (BlackBerry) or upgraded only if signed by the original developer (Android)
- OS allows execution of programs only if signed (iOS)
- OS allows loading of certified device drivers only (Windows)
- Secure boot: OS components booted only if correctly signed

Encrypted code

- There is research into processors that executes encrypted code only
- The processor will decrypt instructions before executing them
 - The decryption key is unique to each processor
 - Malware won't be able to spread without knowing a processor's encryption key

Encrypted code

There is research into processors that executes encrypted code only

- The processor will decrypt instructions before executing them
- The decryption key is unique to each processor
- Malware won't be able to spread without knowing a processor's encryption key

Q: What is the downside?

Encrypted code

There is research into processors that executes encrypted code only

- The processor will decrypt instructions before executing them
- The decryption key is unique to each processor
- Malware won't be able to spread without knowing a processor's encryption key

Q: What is the downside?

A: Scalability of (legitimate) software deployment

Encrypted data

A common technique that aims to protect data in the storage media when the laptop gets lost/stolen, which can be performed either on hardware or by software.

However, it does not protect data against

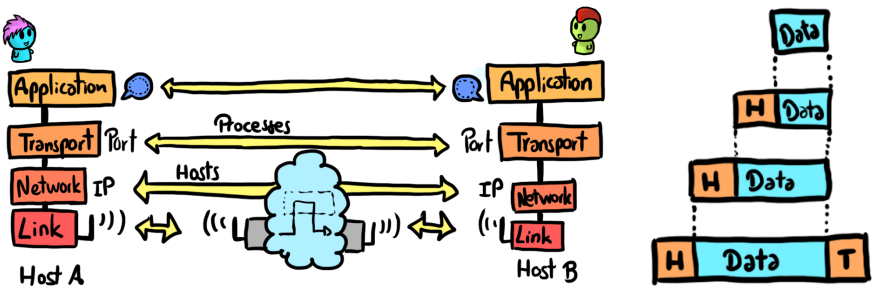
- Other users who legitimately use laptop
- Somebody installing malware on laptop
- Somebody (maybe physically) extracting the decryption key from the laptop's memory

Network security and privacy

Entities you can only communicate with over a network are inherently less trustworthy (e.g., they may not be who they claim to be). This makes networking a primary scenario for cryptography.

This is a separation of concern, and in particular, “*separating the security of the medium from the security of the message*”

Recall the Network Stack



Q: Where do we need to apply crypto?
 (i.e., one or more of confidentiality, integrity, authentication)

- A. Link layer is enough
- B. Application layer is enough
- C. We need it in all layers
- D. Who needs crypto?

Network security and privacy

Cryptography is used at every layer of the network stack for both security and privacy applications:

- Link
 - WEP, WPA, WPA2
- Network
 - VPN, IPsec
- Transport
 - TLS / SSL, Tor
- Application
 - ssh, PGP, OTR, Signal, Mixminion

Outline

- 1 Overview of Security Controls
- 2 Link-layer Security (WEP, WPA)
- 3 Network-layer Security (IPSec)
- 4 Transport-layer Security (TLS)
- 5 Application-layer Security (SSH)
- 6 Application-layer Security (PGP)
- 7 Application-layer Security (OTR)

Wired Equivalent Privacy (WEP)

The Wired Equivalent Privacy (WEP) protocol is a link-layer security protocol that aims to *make wireless communication links just as secure as wired links.*

Wired Equivalent Privacy (WEP)

The Wired Equivalent Privacy (WEP) protocol is a link-layer security protocol that aims to *make wireless communication links just as secure as wired links*.

In particular, WEP was intended to enforce three security goals

- Data Confidentiality
 - Prevent an adversary from learning the contents of the wireless traffic
- Data Integrity
 - Prevent an adversary from modifying the wireless traffic or fabricating traffic that looks legitimate
- Access Control
 - Prevent an adversary from using your wireless infrastructure

Wired Equivalent Privacy (WEP)

The Wired Equivalent Privacy (WEP) protocol is a link-layer security protocol that aims to *make wireless communication links just as secure as wired links*.

In particular, WEP was intended to enforce three security goals

- Data Confidentiality
 - Prevent an adversary from learning the contents of the wireless traffic
- Data Integrity
 - Prevent an adversary from modifying the wireless traffic or fabricating traffic that looks legitimate
- Access Control
 - Prevent an adversary from using your wireless infrastructure

Unfortunately, **none** of these is actually enforced!

WEP description

The sender and receiver share a secret s (either 40 or 104 bits)

WEP description

The sender and receiver share a secret s (either 40 or 104 bits)

In order to transmit a message M :

- Compute a checksum $c(M)$, *which does not depend on s*
- Pick an IV v and generate a keystream $K = RC4(v, s)$
- Ciphertext $C = K \oplus \langle M \parallel c(M) \rangle$
- Transmit v and C over the wireless link

WEP description

The sender and receiver share a secret s (either 40 or 104 bits)

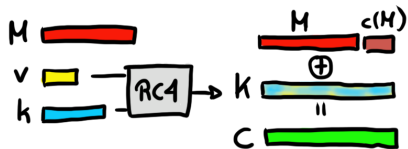
In order to transmit a message M :

- Compute a checksum $c(M)$, which does not depend on s
- Pick an IV v and generate a keystream $K = RC4(v, s)$
- Ciphertext $C = K \oplus \langle M \parallel c(M) \rangle$
- Transmit v and C over the wireless link

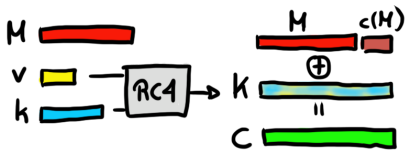
Upon receipt of v and C :

- Use the received v and the shared s for $K = RC4(v, s)$
- Decrypt as $K \oplus C = K \oplus K \oplus \langle M' \parallel c' \rangle = M' \parallel c'$
- Check to see if $c' = c(M')$
- If it is, accept M' as the message transmitted

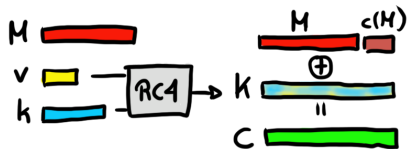
WEP illustration



WEP illustration

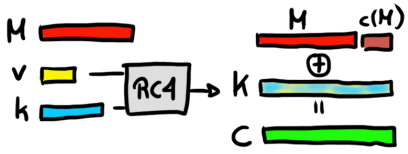


WEP illustration



Q: What kind of cipher is this?

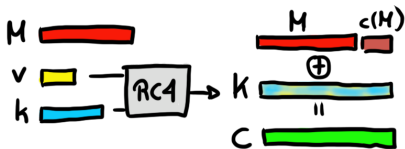
WEP illustration



Q: What kind of cipher is this?

A: It's a stream cipher (w/ symmetric key)

WEP illustration

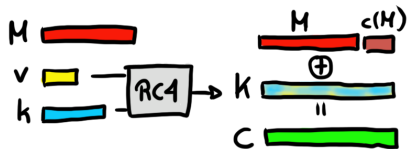


Q: What kind of cipher is this?

A: It's a stream cipher (w/ symmetric key)

Q: What does the receiver do with v and C ?

WEP illustration



Q: What kind of cipher is this?

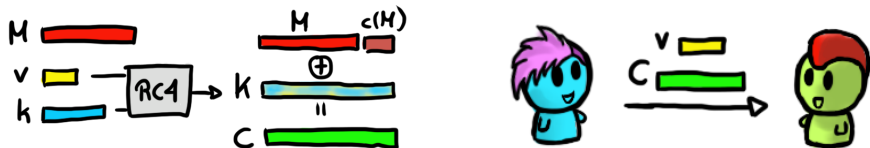
A: It's a stream cipher (w/ symmetric key)

Q: What does the receiver do with v and C ?

A: Upon receipt of v and C :

- Use the received v and the shared k for $K = RC4(v, k)$
- Decrypt as $K \oplus C = K \oplus K \oplus \langle M' \parallel c' \rangle = M' \parallel c'$
- Check to see if $c' = c(M')$
- If it is, accept M' as the message transmitted

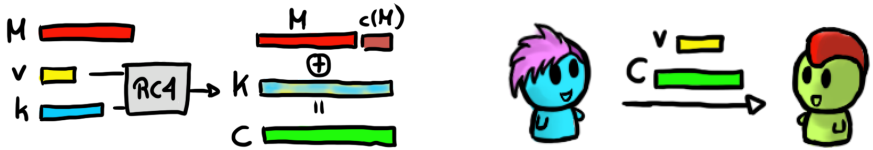
Problem 1: key reuse



Keystream is derived as: $K = RC4(v, s)$

- IV (v) is too short: only 3 bytes = 24 bits.
- Secret (s) is rarely changed!

Problem 1: key reuse

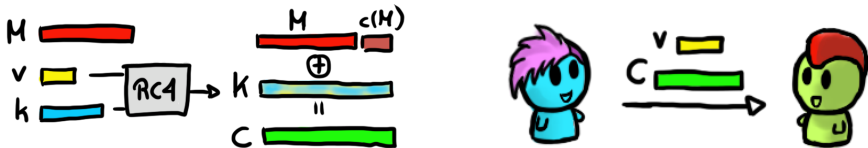


Keystream is derived as: $K = RC4(v, s)$

- IV (v) is too short: only 3 bytes = 24 bits.
- Secret (s) is rarely changed!

Q: What is the problem with this?

Problem 1: key reuse



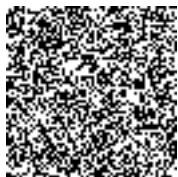
Keystream is derived as: $K = RC4(v, s)$

- IV (v) is too short: only 3 bytes = 24 bits.
- Secret (s) is rarely changed!

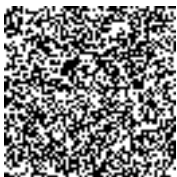
Q: What is the problem with this?

A: Key-stream gets re-used after 2^{24} iterations \rightarrow two-time pad.

Recall: two-time pad



C_1



C_2



$C_1 \oplus C_2$

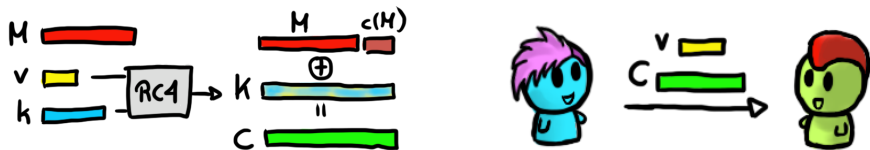


M_2



M_1

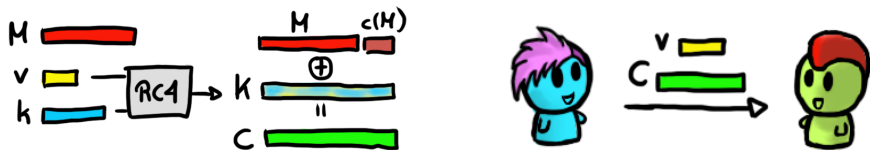
WEP checksum calculation



The checksum algorithm in WEP is CRC32, which has two important (and undesirable) properties:

- It is independent of k and v
- It is **linear**: $c(M \oplus D) = c(M) \oplus c(D)$

WEP checksum calculation

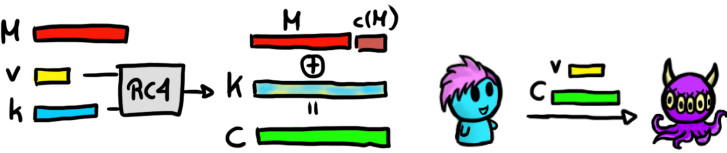


The checksum algorithm in WEP is CRC32, which has two important (and undesirable) properties:

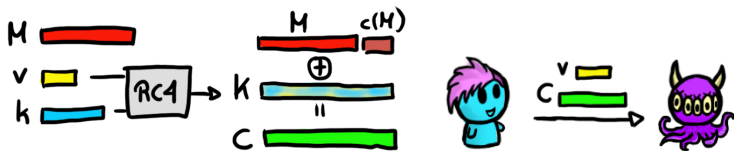
- It is independent of k and v
- It is **linear**: $c(M \oplus D) = c(M) \oplus c(D)$

Q: Why is linearity a pessimal property for your integrity mechanism to have, especially when used in conjunction with a stream cipher? (Hint: chosen ciphertext attack on Textbook RSA)

Problem 2: integrity breach



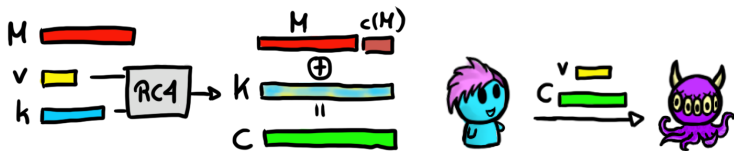
Problem 2: integrity breach



Mallory knows C and v in

$$C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$$

Problem 2: integrity breach



Mallory knows C and v in

$$C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$$

If she wants to modify the plaintext M into $M' = M \oplus \delta$,

Problem 2: integrity breach



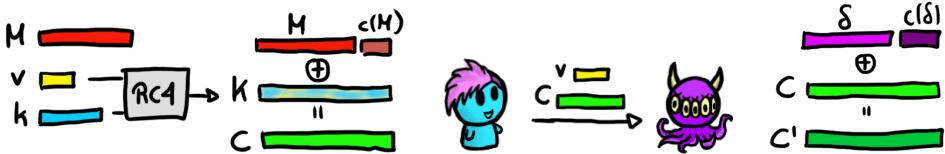
Mallory knows C and v in

$$C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$$

If she wants to modify the plaintext M into $M' = M \oplus \delta$, all she needs to do is

- Calculate $C' = C \oplus \langle \delta \parallel c(\delta) \rangle$
- Send (C', v) instead of (C, v)

Problem 2: integrity breach

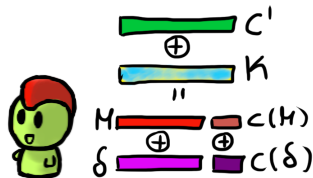


Mallory knows C and v in

$$C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$$

If she wants to modify the plaintext M into $M' = M \oplus \delta$, all she needs to do is

- Calculate $C' = C \oplus \langle \delta \parallel c(\delta) \rangle$
- Send (C', v) instead of (C, v)



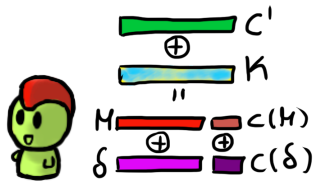
This passes the integrity check of the recipient (e.g., Bob in this case)!

Problem 2: integrity breach

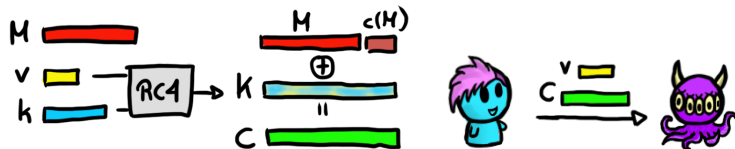


Here is the math workout:

- $C' = C \oplus \langle \delta \parallel c(\delta) \rangle$
- $C' = RC4(v, s) \oplus \langle M \parallel c(M) \rangle \oplus \langle \delta \parallel c(\delta) \rangle$
- $C' = RC4(v, s) \oplus \langle (M \oplus \delta) \parallel (c(M) \oplus c(\delta)) \rangle$
- $C' = RC4(v, s) \oplus \langle (M \oplus \delta) \parallel c(M \oplus \delta) \rangle$
- $C' = RC4(v, s) \oplus \langle M' \parallel c(M') \rangle$

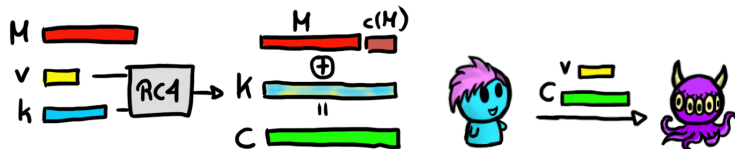


Problem 3: packet injection



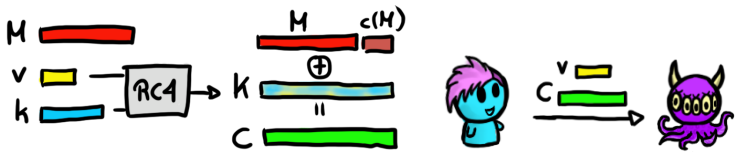
If Mallory knows C , M , and v in $C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$

Problem 3: packet injection



If Mallory knows C , M , and v in $C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$
 ... and she wants to inject message M' to the network,

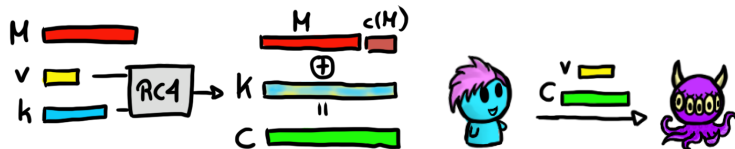
Problem 3: packet injection



If Mallory knows C , M , and v in $C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$
 ... and she wants to inject message M' to the network,
 ... then, all she needs to do is:

- Derive the keystream: $RC4(v, s) = C \oplus \langle M \parallel c(M) \rangle$
- Calculate $C' = RC4(v, s) \oplus \langle M' \parallel c(M') \rangle$
- Send (C', v) to the network

Problem 3: packet injection

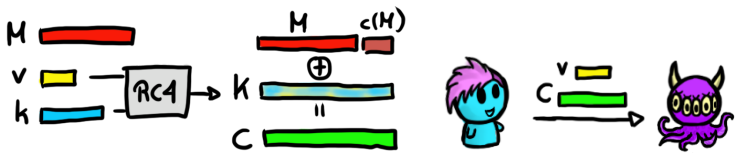


If Mallory knows C , M , and v in $C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$
 ... and she wants to inject message M' to the network,
 ... then, all she needs to do is:

- Derive the keystream: $RC4(v, s) = C \oplus \langle M \parallel c(M) \rangle$
- Calculate $C' = RC4(v, s) \oplus \langle M' \parallel c(M') \rangle$
- Send (C', v) to the network

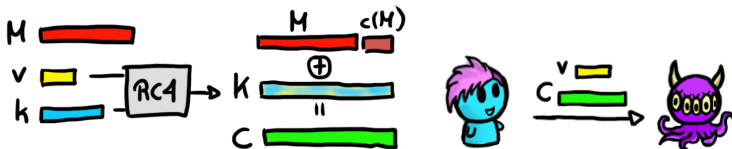
This passes the integrity check of the recipient!

Problem 3: packet injection



If Mallory knows C , M , and v in $C = RC4(v, s) \oplus \langle M || c(M) \rangle$

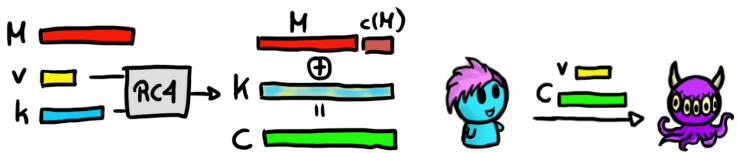
Problem 3: packet injection



If Mallory knows C , M , and v in $C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$

Q: How does Eve get a plaintext / ciphertext pair?

Problem 3: packet injection



If Mallory knows C , M , and v in $C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$

Q: How does Eve get a plaintext / ciphertext pair?

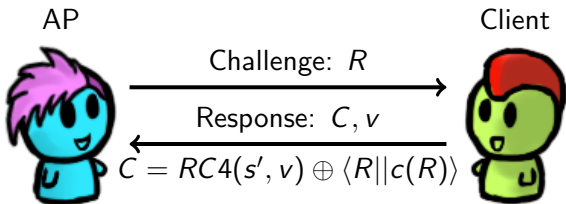
A: It turns out the authentication protocol of WEP gives it to the adversary **for free!**

WEP authentication protocol

The goal of the authentication protocol is to prove that a certain client knows the shared secret s

WEP authentication protocol

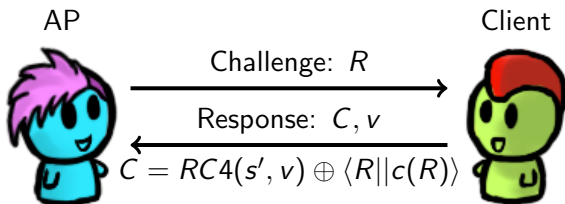
The goal of the authentication protocol is to prove that a certain client knows the shared secret s



- The access point (AP) sends a challenge (a 128-bit random number R) to the client, in plaintext.
- The client picks an IV v and sends back $C = RC4(s', v) \oplus \langle R || c(R) \rangle$ together with v
- The AP decrypts C to get R' and $R' = R \implies s' = s$

WEP authentication protocol

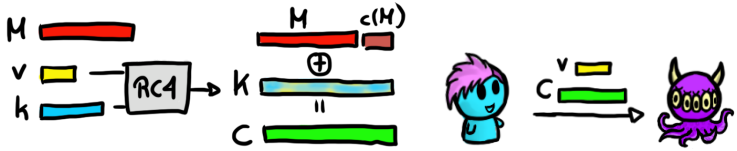
The goal of the authentication protocol is to prove that a certain client knows the shared secret s



- The access point (AP) sends a challenge (a 128-bit random number R) to the client, in plaintext.
- The client picks an IV v and sends back $C = RC4(s', v) \oplus \langle R || c(R) \rangle$ together with v
- The AP decrypts C to get R' and $R' = R \implies s' = s$

The whole process can be observed by the adversary!
 \implies getting R , C , and v for free!

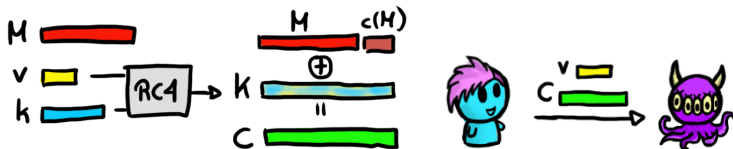
Problem 4: packet injection \implies authentication



If Mallory knows C , M , and v in $C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$
 ... and she wants to inject message M' to the network,
 ... then, all she needs to do is:

- Derive the keystream: $RC4(v, s) = C \oplus \langle M \parallel c(M) \rangle$
- Calculate $C' = RC4(v, s) \oplus \langle M' \parallel c(M') \rangle$
- Send (C', v) to the network

Problem 4: packet injection \implies authentication



If Mallory knows C , M , and v in $C = RC4(v, s) \oplus \langle M \parallel c(M) \rangle$
 ... and she wants to inject message $M' = R$ to the network,
 ... then, all she needs to do is:

- Derive the keystream: $RC4(v, s) = C \oplus \langle M \parallel c(M) \rangle$
- Calculate $C' = RC4(v, s) \oplus \langle M' \parallel c(M') \rangle$
- Send (C', v) to the network

In this way, Mallory authenticates to the AP **even without knowing s**

More problems with WEP

- Somewhat surprisingly, the ability to modify and inject packets leads to ways in which Eve can trick the AP to **decrypt** packets! Check [Prof. Goldberg's talk](#) for more details.
- Note that none of the attacks so far use the fact that the stream cipher was RC4. it turns out that when RC4 is used with similar keys, the output keystream has a subtle weakness, which lead the recovery of either a 104-bit or 40-bit WEP key in **under 60 seconds**, most of the time. Check [this paper](#) for more details.

Replacing WEP

Wi-fi Protected Access (WPA) was rolled out as a short-term patch to WEP while formal standards for a replacement protocol (IEEE 802.11i, later called WPA2) were being developed

- Replaces CRC-32 with a real MAC
- IV is 48 bits
- Key is changed frequently (TKIP)
- Ability to use a 802.1x authentication server
 - But maintains a less-secure PSK (Pre-Shared Key) mode for home users
- Ability to run on most older WEP hardware

Replacing WEP

The 802.11i standard was finalized in 2004, and the result (called WPA2) has been required for products calling themselves “Wi-fi” since 2006

- Replaces the RC4 and MAC algorithms in WPA with the CCM authenticated encryption mode (using AES)
- Considered strong, except in PSK mode
 - Dictionary attacks still possible (avoided in WPA3 (2018))

WEP Recap

- Q:** What have we learned from WEP?
- Respect to randomness? (provided by IVs?)
 - Respect to checksums?

WEP Recap

Q: What have we learned from WEP?

- Respect to randomness? (provided by IVs?)
- Respect to checksums?

A: Some reflections:

- Use sufficiently long IVs, don't share a key with many people, don't reuse short-term secret keys and IVs.
- Do not use checksums for integrity. Use keyed MACs instead!

WEP Recap

Q: What have we learned from WEP?

- Respect to randomness? (provided by IVs?)
- Respect to checksums?

A: Some reflections:

- Use sufficiently long IVs, don't share a key with many people, don't reuse short-term secret keys and IVs.
- Do not use checksums for integrity. Use keyed MACs instead!

You need to understand what was wrong with WEP, how to fix these issues, and you need to be able to identify these issues in other protocols.

Outline

- 1 Overview of Security Controls
- 2 Link-layer Security (WEP, WPA)
- 3 Network-layer Security (IPSec)**
- 4 Transport-layer Security (TLS)
- 5 Application-layer Security (SSH)
- 6 Application-layer Security (PGP)
- 7 Application-layer Security (OTR)

Network layer security: purpose

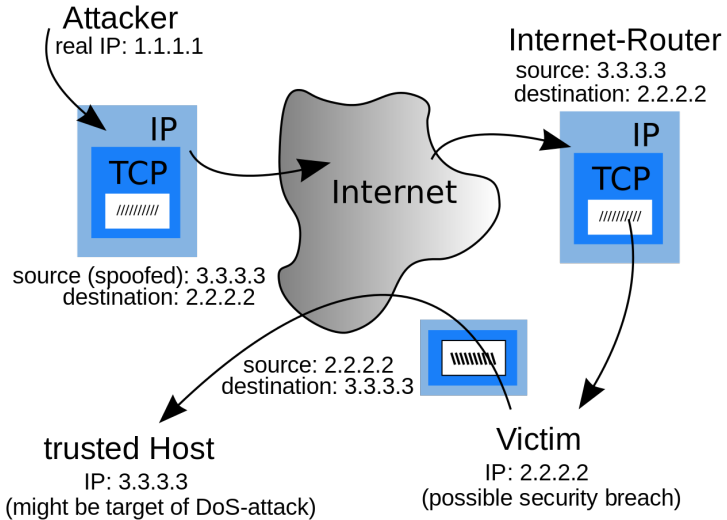
Q: Suppose every link in our network had strong link-layer security. Would this be enough?

Network layer security: purpose

Q: Suppose every link in our network had strong link-layer security. Would this be enough?

A: Source, destination IPs may not share the same link. Network layer threats such as IP spoofing still exist.

Recap: IP spoofing



Network layer security: purpose

We need **end-to-end** security **across** networks, i.e., securing network layer packets from one host to another so that routers or other hosts in the middle cannot modify or read the packet payload (they still need to read packet metadata for routing)

Network layer security: purpose

We need **end-to-end** security **across** networks, i.e., securing network layer packets from one host to another so that routers or other hosts in the middle cannot modify or read the packet payload (they still need to read packet metadata for routing)

The IP Security suite (IPSec) extends the Internet Protocol (IP) to provide confidentiality and integrity of packets transmitted across the network. IPSec enables various architectures of Virtual Private Networks (VPNs) which is the foundation in network-layer security.

When trust cannot be assumed...

We need end-to-end security across networks, i.e., securing network layer packets from one host to another so that routers or other hosts in the middle cannot modify or read the packet payload (they still need to read packet metadata for routing)

The IP Security suite (IPSec) extends the Internet Protocol (IP) to provide confidentiality and integrity of packets transmitted across the network. IPSec enables various architectures of Virtual Private Networks (VPNs) which is the foundation in network-layer security.

When trust cannot be assumed...

We need end-to-end security across networks, i.e., securing network layer packets from one host to another so that routers or other hosts in the middle cannot modify or read the packet payload (they still need to read packet metadata for routing)

The IP Security suite (IPSec) extends the Internet Protocol (IP) to provide confidentiality and integrity of packets transmitted across the network. IPSec enables various architectures of Virtual Private Networks (VPNs) which is the foundation in network-layer security.

Q: What tool do we have when trust cannot be assumed?

When trust cannot be assumed...

We need end-to-end security across networks, i.e., securing network layer packets from one host to another so that routers or other hosts in the middle cannot modify or read the packet payload (they still need to read packet metadata for routing)

The IP Security suite (IPSec) extends the Internet Protocol (IP) to provide confidentiality and integrity of packets transmitted across the network. IPSec enables various architectures of Virtual Private Networks (VPNs) which is the foundation in network-layer security.

Q: What tool do we have when trust cannot be assumed?

Q: How to get the key?

Internet Key Exchange (IKE)

The source and destination IP addresses agree on a shared symmetric key via the IKE process, which internally uses the [Diffie-Hellman](#) protocol:

Internet Key Exchange (IKE)

The source and destination IP addresses agree on a shared symmetric key via the IKE process, which internally uses the Diffie-Hellman protocol:

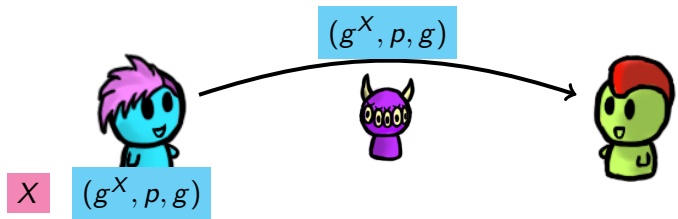
- Alice chooses prime p at random and finds a generator g (i.e., a primitive root modulo p)
- Alice chooses $X \leftarrow_R \{0, 1, \dots, p - 2\}$ and sends $A = g^X \pmod{p}$ to Bob, together with p and g
- Bob chooses $Y \leftarrow_R \{0, 1, \dots, p - 2\}$ and sends $B = g^Y \pmod{p}$ to Alice
- Alice and Bob both compute $s = g^{XY} \pmod{p}$
 - Alice does that by computing $B^X \pmod{p}$
 - Bob does that by computing $A^Y \pmod{p}$
- Now they share a common secret s which can be used to derive a symmetric key

Diffie-Hellman protocol



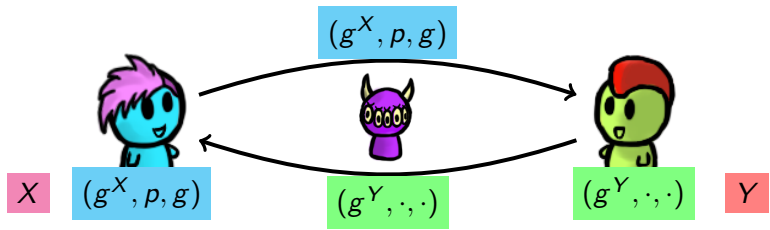
(Background: computing discrete log, e.g., $\log_g Z \pmod{p}$ is hard!)

Diffie-Hellman protocol



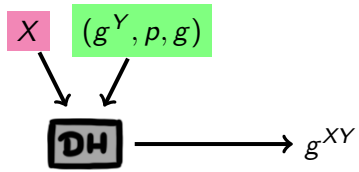
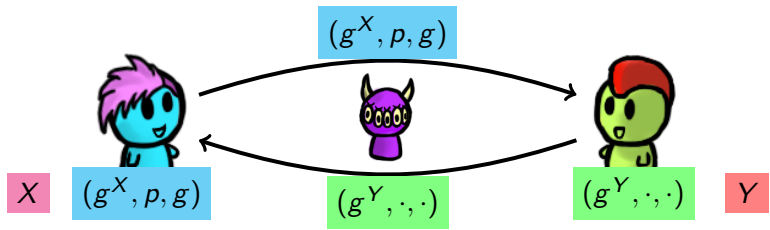
(Background: computing discrete log, e.g., $\log_g Z \pmod p$ is hard!)

Diffie-Hellman protocol



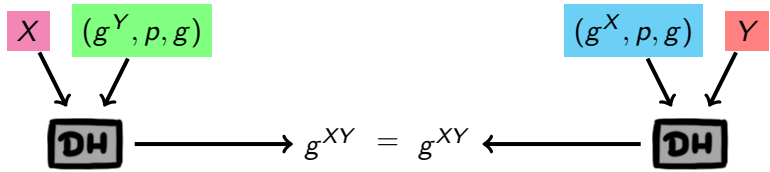
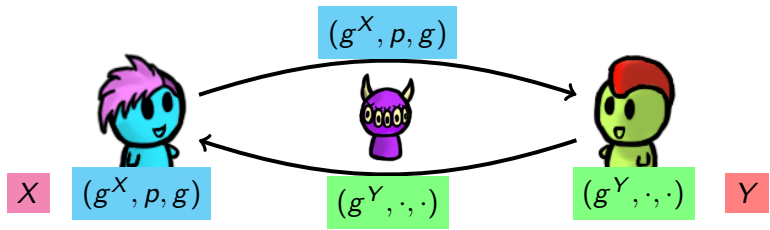
(Background: computing discrete log, e.g., $\log_g Z \pmod{p}$ is hard!)

Diffie-Hellman protocol

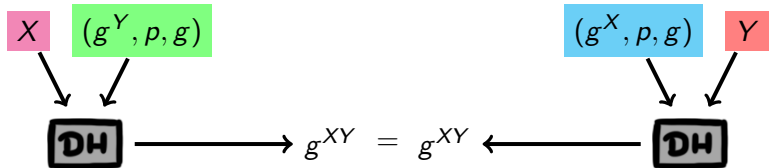
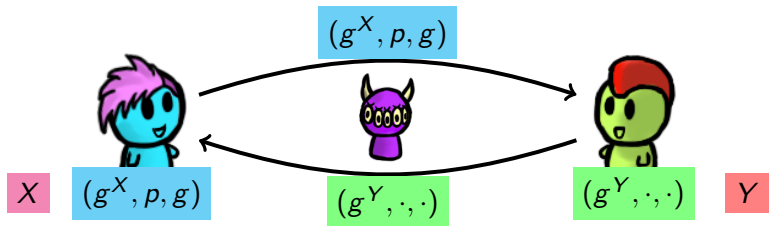


(Background: computing discrete log, e.g., $\log_g Z \pmod p$ is hard!)

Diffie-Hellman protocol



Diffie-Hellman protocol



(Background: computing discrete log, e.g., $\log_g Z \pmod p$ is hard!)

IPSec headers

Now we have a secret shared by both parties, how to use it?

IPSec headers

Now we have a secret shared by both parties, how to use it?

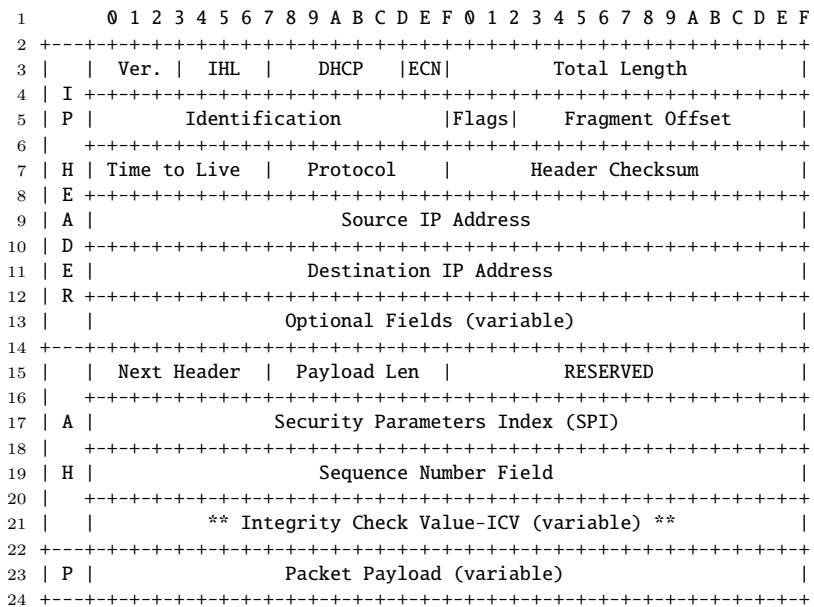
Authentication Header (AH) – RFC4302

- Offers **integrity** and data **source authentication**
 - Authenticates payload and parts of IP header that do not get modified during transfer, e.g., source IP address
- Offers protection against replay attacks
 - Via extended sequence numbers

Encapsulated Security Payload (ESP) – RFC4303

- Offers **confidentiality**
 - IP data is encrypted during transmission
- Offers **authentication** functionality similar to AH
 - But authenticity checks only focus on the IP payload
- Applies padding and generates dummy traffic
 - Makes traffic analysis harder

Authentication Header (AH): RCF4302



Mode of operation

A regular IP packet in the form of $\langle H \parallel P \rangle$ can be transformed into an IPSec packet depending on the mode of operation:

	AH	ESP
Transport		
Tunnel		

Mode of operation

A regular IP packet in the form of $\langle H \parallel P \rangle$ can be transformed into an IPSec packet depending on the mode of operation:

	AH	ESP
Transport	$H \parallel AH \parallel P$ \hookrightarrow Int. of H (partial) and P	
Tunnel		

Mode of operation

A regular IP packet in the form of $\langle H \parallel P \rangle$ can be transformed into an IPSec packet depending on the mode of operation:

	AH	ESP
Transport	$H \parallel AH \parallel P$ \hookrightarrow Int. of H (partial) and P	
Tunnel	$H' \parallel AH \parallel \langle H \parallel P \rangle$ \hookrightarrow Int. of H (whole) and P	

Mode of operation

A regular IP packet in the form of $\langle H \parallel P \rangle$ can be transformed into an IPSec packet depending on the mode of operation:

	AH	ESP
Transport	$H \parallel AH \parallel P$ \hookrightarrow Int. of H (partial) and P	$H \parallel ESP-H \parallel \langle P \rangle_k \parallel ESP-T$ \hookrightarrow Int. and Conf. of P only
Tunnel	$H' \parallel AH \parallel \langle H \parallel P \rangle$ \hookrightarrow Int. of H (whole) and P	

Mode of operation

A regular IP packet in the form of $\langle H \parallel P \rangle$ can be transformed into an IPSec packet depending on the mode of operation:

	AH	ESP
Transport	$H \parallel AH \parallel P$ \hookrightarrow Int. of H (partial) and P	$H \parallel ESP-H \parallel \langle P \rangle_k \parallel ESP-T$ \hookrightarrow Int. and Conf. of P only
Tunnel	$H' \parallel AH \parallel \langle H \parallel P \rangle$ \hookrightarrow Int. of H (whole) and P	$H' \parallel ESP-H \parallel \langle H \parallel P \rangle_k \parallel ESP-T$ \hookrightarrow Int. and Conf. of H and P

Mode of operation

A regular IP packet in the form of $\langle H \parallel P \rangle$ can be transformed into an IPSec packet depending on the mode of operation:

	AH	ESP
Transport	$H \parallel AH \parallel P$ \leftrightarrow Int. of H (partial) and P	$H \parallel ESP-H \parallel \langle P \rangle_k \parallel ESP-T$ \leftrightarrow Int. and Conf. of P only
Tunnel	$H' \parallel AH \parallel \langle H \parallel P \rangle$ \leftrightarrow Int. of H (whole) and P	$H' \parallel ESP-H \parallel \langle H \parallel P \rangle_k \parallel ESP-T$ \leftrightarrow Int. and Conf. of H and P

The Tunnel-ESP combination (also known as an IP-in-IP tunneling) is often used to implement Virtual Private Networks (VPNs)

IPSec deployment challenges

- Needs to be included in the kernel's network stack implementation.
- May use inline hardware that implements IPSec.
- There may be legitimate reasons to modify some IP header fields; IPSec breaks networking functionalities that require such changes.

Outline

- ① Overview of Security Controls
- ② Link-layer Security (WEP, WPA)
- ③ Network-layer Security (IPSec)
- ④ Transport-layer Security (TLS)**
- ⑤ Application-layer Security (SSH)
- ⑥ Application-layer Security (PGP)
- ⑦ Application-layer Security (OTR)

Transport-layer security and privacy

- Network-layer security mechanisms arrange to send *individual* IP packets securely from one network to another
- Transport-layer security mechanisms transform arbitrary TCP connections to add security and privacy

Transport-layer security and privacy

- Network-layer security mechanisms arrange to send *individual* IP packets securely from one network to another
- Transport-layer security mechanisms transform arbitrary TCP connections to add security and privacy

- The main transport-layer security mechanism:
 - TLS (formerly known as SSL)
- The main transport-layer privacy mechanism:
 - Tor — will be covered in the lecture on PETs.

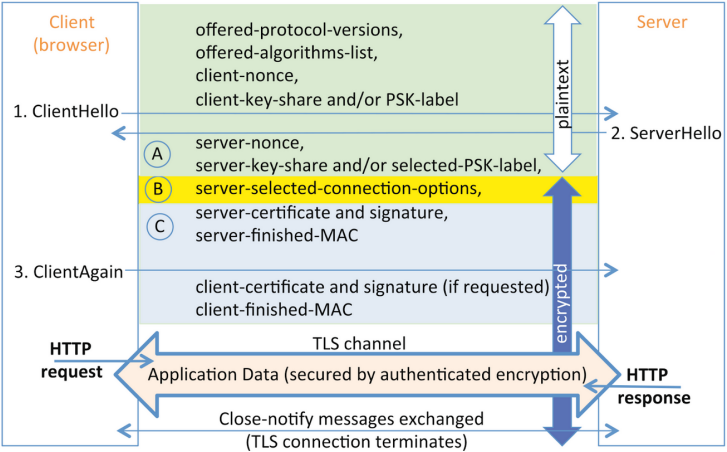
TLS / SSL

- In the mid-1990s, Netscape invented a protocol called Secure Sockets Layer (SSL) for protecting HTTP (web) connections
 - HTTP + SSL = **HTTPS**
 - The protocol, however, was general, and could be used to protect **any** TCP-based connection
- Historical note: there was a competing protocol called S-HTTP. But Netscape and Microsoft both chose HTTPS, so that's the protocol everyone else followed
- SSL went through a few revisions, and was eventually standardized into the protocol known as **TLS** (Transport Layer Security, imaginatively enough)

TLS at a high level: RFC8446

- Client connects to server, indicates it wants to speak TLS, with
 - Client key-share under ECDHE
 - The list of ciphersuites it knows
- Server sends its certificate to client, which contains:
 - Server key-share under ECDHE
 - Its host name
 - Its verification key
 - Some other administrative information
 - Server signature and certificate
- Both client and server derives tbe same session key K (which is hard for Eve to derive) based on the two key shares
- Server also chooses which ciphersuite to use
- All remaining traffic will be encrypted and authenticated under K

TLS connection establishment



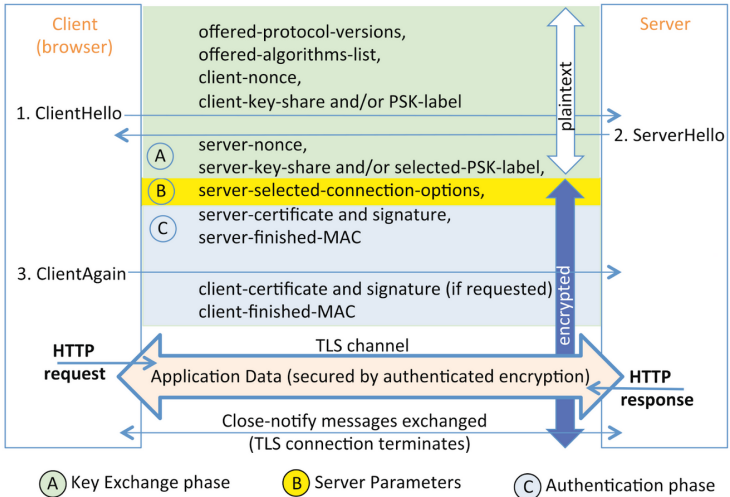
- (A) Key Exchange phase
- (B) Server Parameters
- (C) Authentication phase

Security properties provided by TLS

- Server authentication
- Message integrity
- Message confidentiality
- Client authentication (optional)

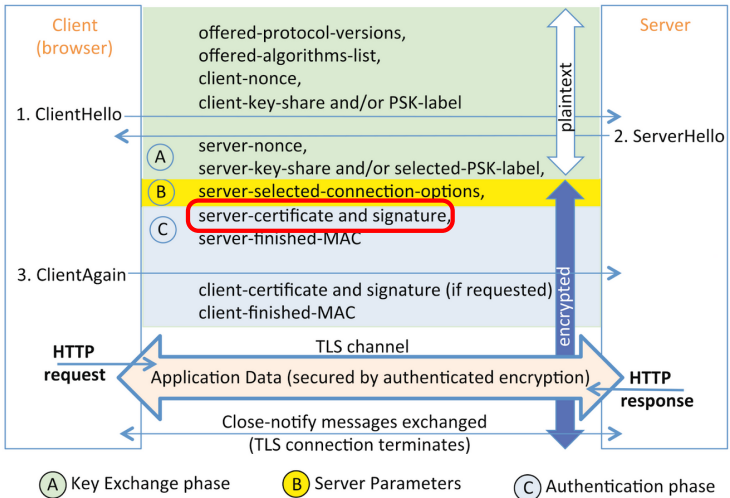
Authenticity

So we established confidentiality, but how does the client authenticate the server?



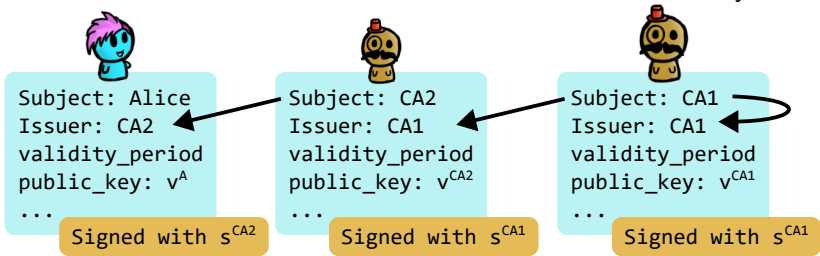
Authenticity


So we established confidentiality, but how does the client authenticate the server?



Recap: Certificate Authority (CA)

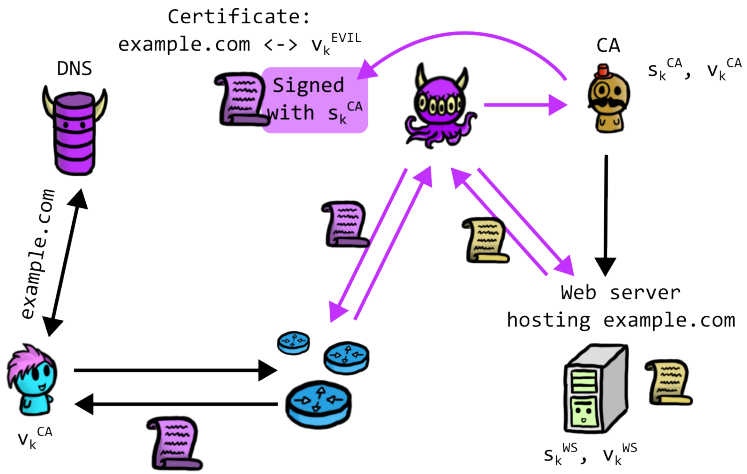
Alice sends Bob the following certificate to prove her identity. Bob can follow the chain of certificates to validate Alice's identity.



 Bob has v^{CA1}

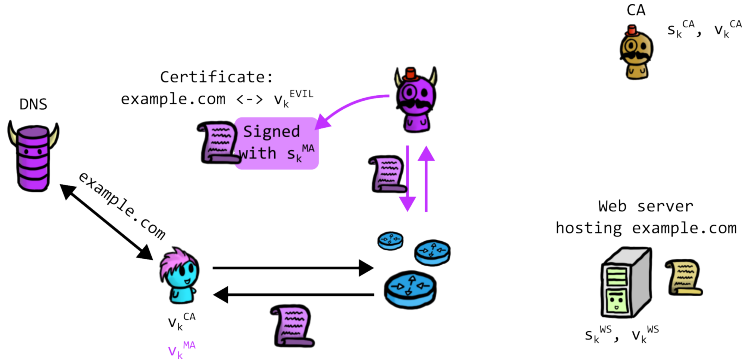
What can go wrong with these CAs

An adversary can compromise a CA to plant fake certificates (e.g., DigiNotar's fake *.google.com certificates used by an ISP in Iran)



What can go wrong with these CAs

An adversary can install a custom CA on users' devices, allowing them to sign certificates that clients will accept for any site (e.g., in 2019, Kazakhstan's ISPs mandated the installation of a root certificate issued by the government)



What's next?

TLS can provide for confidentiality, integrity, and authenticity at the TCP socket level. This is already an “end-to-end” protection in the sense of a network connection (i.e., host-to-host).

Q: Is this good enough?

What's next?

TLS can provide for confidentiality, integrity, and authenticity at the TCP socket level. This is already an “end-to-end” protection in the sense of a network connection (i.e., host-to-host).

Q: Is this good enough?

A: An adversary can still decrypt TLS traffic if the adversary has an access to the **pre-master secret** (or more directly, the session key).

- PolarProxy is primarily designed to intercept and decrypt SSL or TLS encrypted traffic from malware.
- curl and browsers such as Chrome and Firefox can generate these secrets when the connection is set up and dump them in a file pointed to by an environment variable `SSLKEYLOGFILE`.

Outline

- 1 Overview of Security Controls
- 2 Link-layer Security (WEP, WPA)
- 3 Network-layer Security (IPSec)
- 4 Transport-layer Security (TLS)
- 5 Application-layer Security (SSH)**
- 6 Application-layer Security (PGP)
- 7 Application-layer Security (OTR)

Secure remote login (ssh)

- You're already familiar with this tool for securely logging in to a remote computer (the ugster machines)

Secure remote login (ssh)

- You're already familiar with this tool for securely logging in to a remote computer (the ugster machines)
- Usual usage (simplified):
 - Client connects to server
 - Server sends its verification key
 - The client **should** verify that this is the correct key

Secure remote login (ssh)

- You're already familiar with this tool for securely logging in to a remote computer (the ugster machines)
- Usual usage (simplified):
 - Client connects to server
 - Server sends its verification key
 - The client **should** verify that this is the correct key

Q: How to verify the key is the correct key?

Secure remote login (ssh)

- You're already familiar with this tool for securely logging in to a remote computer (the ugster machines)
- Usual usage (simplified):
 - Client connects to server
 - Server sends its verification key
 - The client **should** verify that this is the correct key

Q: How to verify the key is the correct key?

A:

```
$ ssh soyadiez@ugster504.student.cs.uwaterloo.ca
The authenticity of host 'ugster504.student.cs.uwaterloo.ca (129.97.173.82)' can't be established.
ED25519 key fingerprint is SHA256:hKzMoU1+2mKMSmx2bMRHyItifiKU6POHaYLSH39jq1M.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ugster504.student.cs.uwaterloo.ca' (ED25519) to the list of known hosts.
```

Secure remote login (ssh)

- You're already familiar with this tool for securely logging in to a remote computer (the ugster machines)
- Usual usage (simplified):
 - Client connects to server
 - Server sends its verification key
 - The client **should** verify that this is the correct key
 - Client and server run a key agreement protocol to establish session keys, server signs its messages
 - All communication from here on in is encrypted and MAC-ed with the session keys
 - *Client authenticates to server*
 - Server accepts authentication, login proceeds

User authentication with ssh

There are two main ways to authenticate with ssh:

- Send a password over the encrypted channel
 - The server needs to know (a hash of) your password
- Sign a random challenge with your private signature key
 - The server needs to know your public verification key

User authentication with ssh

There are two main ways to authenticate with ssh:

- Send a password over the encrypted channel
 - The server needs to know (a hash of) your password
- Sign a random challenge with your private signature key
 - The server needs to know your public verification key

Q: Advantages/disadvantages of each?

User authentication with ssh

There are two main ways to authenticate with ssh:

- Send a password over the encrypted channel
 - The server needs to know (a hash of) your password
- Sign a random challenge with your private signature key
 - The server needs to know your public verification key

Q: Advantages/disadvantages of each?

A: People create weak passwords or have poor password management practices (e.g., writing passwords in post-it notes, etc.)

A: People usually don't protect private keys with passphrases

Outline

- 1 Overview of Security Controls
- 2 Link-layer Security (WEP, WPA)
- 3 Network-layer Security (IPSec)
- 4 Transport-layer Security (TLS)
- 5 Application-layer Security (SSH)
- 6 Application-layer Security (PGP)**
- 7 Application-layer Security (OTR)

Pretty Good Privacy

- The first popular implementation of public-key cryptography.
- Originally made by Phil Zimmermann in 1991
 - He got in a lot of trouble for it, since cryptography was highly controlled at the time.
 - But that's a whole 'nother story. :-)
- Today, there are many (more-or-less) compatible programs
 - GNU Privacy Guard (gpg), Hushmail, etc.

Pretty Good Privacy

- What does it do?
 - Its primary use is to protect the contents of email messages
 - Provides confidentiality, integrity, authentication, and **non-repudiation**

Pretty Good Privacy

- What does it do?
 - Its primary use is to protect the contents of email messages
 - Provides confidentiality, integrity, authentication, and **non-repudiation**
- How does it work?
 - Uses public-key cryptography to provide:
 - Encryption of email messages (using hybrid encryption)
 - Digital signatures on email messages (hash-then-sign)

Recall: Public-key Cryptography



- encryption/decryption:
 (e_A, d_A)
- signature/verification:
 (s_A, v_A)

- encryption/decryption:
 (e_B, d_B)
- signature/verification:
 (s_B, v_B)



Recall: Public-key Cryptography



- encryption/decryption: (e_A, d_A)
- signature/verification: (s_A, v_A)

- encryption/decryption: (e_B, d_B)
- signature/verification: (s_B, v_B)



To send a message m to Bob, Alice will:

- 1 ???
- 2 ???

Q: What is the sequence of actions to encode m ?

Recall: Public-key Cryptography



- encryption/decryption:
(e_A, d_A)
- signature/verification:
(s_A, v_A)

- encryption/decryption:
(e_B, d_B)
- signature/verification:
(s_B, v_B)



To send a message m to Bob, Alice will:

- 1 ???
- 2 ???

Q: What is the sequence of actions to encode m ?

A: $E_{e_B}(m \parallel \text{Sign}_{s_A}(m))$

Recall: Public-key Cryptography



- encryption/decryption: (e_A, d_A)
- signature/verification: (s_A, v_A)

- encryption/decryption: (e_B, d_B)
- signature/verification: (s_B, v_B)



To send a message m to Bob, Alice will:

- 1 ???
- 2 ???

Q: What is the sequence of actions to encode m ?

A: $E_{e_B}(m \parallel \text{Sign}_{s_A}(m))$

Q: And you also know what Bob does next...

Back to PGP

PGP's main functions:

- Create these four kinds of keys
 - encryption, decryption, signature, verification
- Encrypt messages using someone else's encryption key
- Decrypt messages using your own decryption key
- Sign messages using your own signature key
- Verify signatures using someone else's verification key

Back to PGP

PGP's main functions:

- Create these four kinds of keys
 - encryption, decryption, signature, verification
- Encrypt messages using someone else's encryption key
- Decrypt messages using your own decryption key
- Sign messages using your own signature key
- Verify signatures using someone else's verification key
- **Sign other people's keys using your own signature key**

Back to PGP

PGP's main functions:

- Create these four kinds of keys
 - encryption, decryption, signature, verification
- Encrypt messages using someone else's encryption key
- Decrypt messages using your own decryption key
- Sign messages using your own signature key
- Verify signatures using someone else's verification key
- **Sign other people's keys using your own signature key**

Disclaimer: in practice there are primary keypairs, used for signing, verifying, and creating encryption sub-keypairs, but let's abstract from this...

Obtaining keys

Earlier, we said that Alice needs to get an authentic copy of Bob's public key in order to send him an encrypted message.

Q: How does Alice do this in TLS setting?

Obtaining keys

Earlier, we said that Alice needs to get an authentic copy of Bob's public key in order to send him an encrypted message.

Q: How does Alice do this in TLS setting?

A: Certificate authorities (CAs)

Obtaining keys

Earlier, we said that Alice needs to get an authentic copy of Bob's public key in order to send him an encrypted message.

Q: How does Alice do this in TLS setting?

A: Certificate authorities (CAs)

What if we don't involve CAs?

- Bob could put a copy of his public key on his webpage

Obtaining keys

Earlier, we said that Alice needs to get an authentic copy of Bob's public key in order to send him an encrypted message.

Q: How does Alice do this in TLS setting?

A: Certificate authorities (CAs)

What if we don't involve CAs?

- Bob could put a copy of his public key on his webpage

Q: Is this good enough?

Verifying public keys

If Alice knows Bob personally, she could:

- Download the key from Bob's web page
- Phone up Bob, and verify she's got the right key

Verifying public keys

If Alice knows Bob personally, she could:

- Download the key from Bob's web page
- Phone up Bob, and verify she's got the right key
- Problem: keys are big and unwieldy!

```
mQGiBDi5qEURBADitpDzvvzW+9lj/zYgK78G3D76hvvvIT6gpTIlwg6WIJNLKJat
01yNpMIYNvpwi7EUd/1SNl6t1/A022p7s7bDbE4T5Njda0IOAgWeOZ/plIJC4+o2
tD2RNuSkwDQcxzm8KUNZOJla4Lvgrkm/oUubxyeY5omus7hcfNrB0wjC1wCg4Jnt
m7s3eNfMu72Cv+6FzBgFog8EANirkNdC1Q8oSMDihWjlogiWbBz4s6HMxzAaqNf/
rCJ9qoK5SLFeoB/r5ksRWty9QKV4VdhHCiy1U2B9tSTlEPYXJHQPZ3mwCxUnJpGD
8UgFM5uKXaEq2pwpArTm367k0tTpMQgXAN2HwiZv//ahQXH4ov30kBBVL5VFxMUL
UJ+yA/4r5HLTpP2SbbqtPWdeW7uDwhe2dTqffAGuf0kuCpHwCTAhr83ivXzT/70M
```

Fingerprints

- Luckily, there's a better way!
- A **fingerprint** is a cryptographic hash of a key
- This, of course, is *much shorter*:
 - B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5
- Remember: there's no (known) way to make two different keys that have the same fingerprint, provided that we use a collision-resistant hash function

Fingerprints

So now we can try this:

- Alice downloads Bob's key from his webpage
- Alice's software calculates the fingerprint
- Alice phones up Bob, and asks him to read his key's actual fingerprint to her
- If they match, Alice knows she's got an authentic copy of Bob's key

Fingerprints

So now we can try this:

- Alice downloads Bob's key from his webpage
- Alice's software calculates the fingerprint
- Alice phones up Bob, and asks him to read his key's actual fingerprint to her
- If they match, Alice knows she's got an authentic copy of Bob's key

That's great for Alice, but what about Carol?

- Carol might not know Bob
- At least not well enough to phone him

Signing keys

- Once Alice has verified Bob's key, she uses her signature key to sign Bob's key
- This is effectively the same as Alice signing a message that says
*"I have verified that the key with fingerprint
 B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5
 really belongs to Bob (bob@bobmail.com)"*

Signing keys

- Once Alice has verified Bob's key, she uses her signature key to sign Bob's key
- This is effectively the same as Alice signing a message that says
*"I have verified that the key with fingerprint
B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5
really belongs to Bob (bob@bobmail.com)"*
- Bob can attach Alice's signature to the key on his webpage
 - If Bob wants, he can get many people to sign his key...

Signing keys

- Once Alice has verified Bob's key, she uses her signature key to sign Bob's key
- This is effectively the same as Alice signing a message that says
*"I have verified that the key with fingerprint
B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5
really belongs to Bob (bob@bobmail.com)"*
- Bob can attach Alice's signature to the key on his webpage
 - If Bob wants, he can get many people to sign his key...

Q: Can you see some potential issue with key signing?

Signing keys

- Once Alice has verified Bob's key, she uses her signature key to sign Bob's key
- This is effectively the same as Alice signing a message that says *"I have verified that the key with fingerprint B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5 really belongs to Bob (bob@bobmail.com)"*
- Bob can attach Alice's signature to the key on his webpage
 - If Bob wants, he can get many people to sign his key...

Q: Can you see some potential issue with key signing?

A: Once you sign a key... you cannot take that back...

Web of Trust

- Now Alice can act as an introducer for Bob
- If Carol doesn't know Bob, but does know Alice (and has already verified Alice's key, and trusts her to introduce other people):
 - she downloads Bob's key from his website
 - she sees Alice's signature on it
 - she is able to use Bob's key without having to check with Bob personally
- This is called the Web of Trust, and the PGP software handles it mostly automatically

So, great!

So if Alice and Bob want to have a private conversation by email:

- They each create their sets of keys
- They exchange public encryption keys and verification keys
- They send signed and encrypted messages back and forth

Pretty Good, no?

HOW TO USE PGP TO VERIFY THAT AN EMAIL IS AUTHENTIC:

LOOK FOR THIS TEXT AT THE TOP.



IF IT'S THERE, THE EMAIL IS PROBABLY FINE.

(If you want to be extra safe, check that there's a big block of jumbled characters at the bottom.)

Plot twist

- Suppose (encrypted) communications between Alice and Bob are recorded by the “bad guys”
 - criminals, competitors, etc
- Later, Bob's computer is stolen by the same bad guys
- Or just broken into
 - Virus, trojan, etc

All of Bob's key material is recovered

The bad guys can...

- Decrypt past messages
- Learn their content
- Learn that Alice sent them
- And have a mathematical **proof** they can show to anyone else!

How private is that?

What went wrong?

What went wrong?

- Bob's computer got stolen?

What went wrong?

- Bob's computer got stolen?
- How many of you have never...
 - Left your laptop unattended?
 - Not installed the latest patches?
 - Run software with a remotely exploitable bug?

What went wrong?

- Bob's computer got stolen?
- How many of you have never...
 - Left your laptop unattended?
 - Not installed the latest patches?
 - Run software with a remotely exploitable bug?
- **What about your friends?**

What really went wrong

- PGP creates lots of incriminating records:
 - Key material that decrypts data sent over the public Internet
 - Signatures with proofs of who said what

- Alice had better watch what she says!
 - Her privacy depends on Bob's actions

Outline

- 1 Overview of Security Controls
- 2 Link-layer Security (WEP, WPA)
- 3 Network-layer Security (IPSec)
- 4 Transport-layer Security (TLS)
- 5 Application-layer Security (SSH)
- 6 Application-layer Security (PGP)
- 7 Application-layer Security (OTR)**

Casual conversations

- Alice and Bob talk in a room
- No one else can hear
 - Unless being recorded
- No one else knows what they say
 - Unless Alice or Bob tells them
- No one can prove what was said
 - Not even Alice or Bob

Casual conversations

- Alice and Bob talk in a room
- No one else can hear
 - Unless being recorded
- No one else knows what they say
 - Unless Alice or Bob tells them
- No one can prove what was said
 - Not even Alice or Bob

These conversations are “off-the-record” (OTR)

We like off-the-record conversations

- Legal support for having them
 - Illegal to record other people's conversations without notification
- We can have them over the phone
 - Illegal to tap phone lines
- But what about over the Internet?

What do we want to achieve?

- (Perfect) **Forward secrecy**: a key compromise does not reveal past communication.

What do we want to achieve?

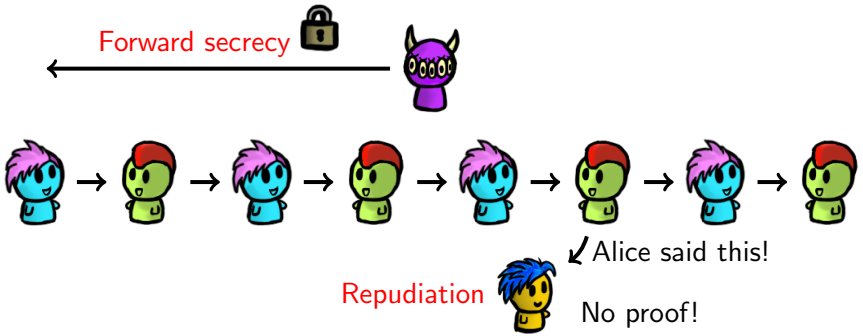
- (Perfect) **Forward secrecy**: a key compromise does not reveal past communication.
- **Repudiation (deniable authentication)**: authenticated communication, but a participant cannot prove to a third party that another participant said something.

What do we want to achieve?

- (Perfect) **Forward secrecy**: a key compromise does not reveal past communication.
- **Repudiation (deniable authentication)**: authenticated communication, but a participant cannot prove to a third party that another participant said something.

What do we want to achieve?

- (Perfect) **Forward secrecy**: a key compromise does not reveal past communication.
- **Repudiation (deniable authentication)**: authenticated communication, but a participant cannot prove to a third party that another participant said something.



Cryptographic tools

We have the cryptographic tools to do OTR, but we need to have new perspectives on how to use these tools

Perfect forward secrecy

- Future key compromises should not reveal past communication
- Use secret-key encryption with a short-lived key (a **session key**)
- The session key is created by a modified Diffie-Hellman protocol
- Discard the session key after use
 - **Securely erase it from memory (and everywhere possible)**
- Use long-term keys only to authenticate the Diffie-Hellman protocol messages only

Deniable authentication

- Do **not** want digital signatures
 - Non-repudiation is great for signing contracts, but undesirable for private conversations
- But we **do** want authentication
 - We can't maintain privacy if attackers can impersonate our friends

Deniable authentication

- Do **not** want digital signatures
 - Non-repudiation is great for signing contracts, but undesirable for private conversations
- But we **do** want authentication
 - We can't maintain privacy if attackers can impersonate our friends

Q: What should we use then?

Deniable authentication

- Do **not** want digital signatures
 - Non-repudiation is great for signing contracts, but undesirable for private conversations
- But we **do** want authentication
 - We can't maintain privacy if attackers can impersonate our friends

Q: What should we use then?

A: Message Authentication Codes (MAC)

No third-party proofs

- Shared-key authentication
 - Alice and Bob have the same K
 - K is required to compute the MAC
 - How is Bob assured that Alice sent the message?
- Bob cannot prove that Alice generated the MAC
 - He could have done it, too
 - Anyone who can verify can also forge
- This gives Alice a measure of deniability

Using these techniques

Using these techniques, we can make our online conversations more like face-to-face “off-the-record” conversations.

But there is a wrinkle:

- These techniques require the parties to communicate interactively
- This makes them unsuitable for email
- But they’re still great for instant messaging!

Off-the-Record Messaging

- Perfect Forward Secrecy
 - Shortly after Bob receives the message, it becomes unreadable to anyone, anywhere (provided the key is erased securely)
- Deniability
 - Although Bob is assured that the message came from Alice, he can't convince Carol of that fact
 - Also, Carol can create forged transcripts of conversations that are every bit as accurate as the real thing