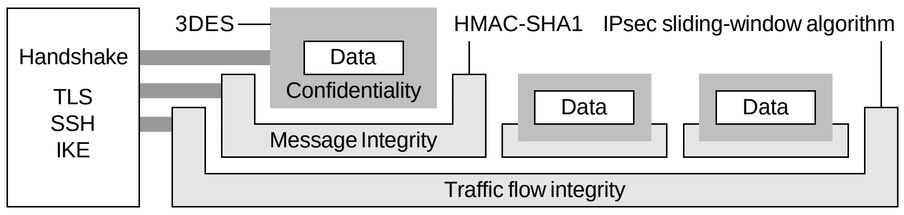
Session-Level Security

PGP, ssh, S/WAN, satan & crack: Securing the internet by any means necessary

— Don Kitchen

Session-level Security Overview

Four main functionality blocks

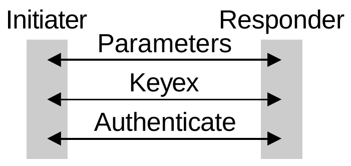


- Handshake/key setup
- Data confidentiality (encryption)
- Message integrity (MAC)
- Message flow integrity (sequence counters)

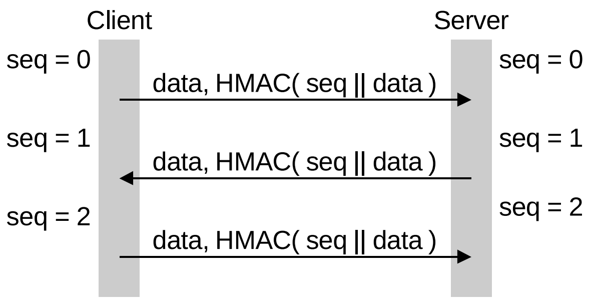
Session-level Security Overview (ctd)

Most handshake mechanisms use some variation of

1. Decide on security parameters
2. Establish a shared secret to protect further communications
3. Authenticate the previous exchange



Session-level Security Overview (ctd)


Client
Server
seq = 0
seq = 0
data, HMAC(seq || data)
seq = 1
seq = 1
data, HMAC(seq || data)
seq = 2
seq = 2
data, HMAC(seq || data)

Synchronised sequence counters provide message flow integrity protection

- If both sides can transmit simultaneously, distinct send and receiver counters are used
- Use over unreliable transport requires IPsec sliding-window algorithm (see later slides)

SSL

Secure sockets layer — TCP/IP socket encryption

- Authenticates the server via proof-of-possession of private key
  - Ability to decrypt client data ties it to the public key
- Can authenticate the client, but this is never used
  - Client-side PKI is just too painful to handle
- Confidentiality protection via encryption
- Integrity protection via MACs
- Message flow integrity protected via sequence numbers
  - TCP/IP transport is inherently reliable
- Provides end-to-end protection of communications sessions

History

SSLv1 was designed by Netscape, broken by members of the audience while it was being presented

SSLv2 shipped with Navigator 1.1

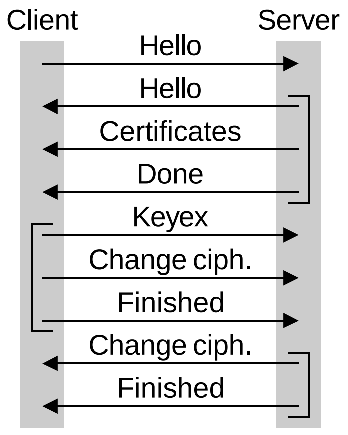
- Microsoft proposed PCT: PCT != SSL

SSLv3 was peer-reviewed, proposed for IETF standardisation

- Never finalised, still exists only as a draft

SSL Handshake

1. Negotiate the cipher suite
2. Establish a shared session key
   - Implicitly authenticates the server via its ability to decrypt the client data
3. Authenticate the client (optional)
4. Authenticate previously exchanged messages



SSL Handshake (ctd)

Client hello:

- Client nonce
- Available cipher suites, e.g. RSA + 3DES + (HMAC-)SHA-1

Server hello:

- Server nonce
- Selected cipher suite

Server adapts to client capabilities

Optional certificate exchange to authenticate the server/client

- In practice only server authentication is used

SSL Handshake (ctd)

Client key exchange:

- RSA-encrypt(premaster secret)

Both sides:

- master secret = PRF(premaster + client-nonce + server-nonce)

Client/server change cipher spec:

- Switch to selected cipher suite and key

Client/server finished

- MAC of previously exchanged parameters
  - Uses an early version of HMAC
- Authenticates all previous messages

SSL Handshake (ctd)

Can resume previous sessions from cached information

- Previous session is identified via ID in Hello message
- More useful for stateless HTTP 1.0 than streaming HTTP 1.1

Can bootstrap weak crypto from strong crypto

- Server has > 512 bit certificate
- Generates a 512-bit temporary key
- Signs the temporary key with the > 512 bit certificate
- Uses the temporary key for security

Other key establishment mechanisms possible

- DH key agreement, exchange a value signed by the server
- Not widely supported, rarely used

SSL Applications

Designed for multiple protocols (HTTP, SMTP, NNTP, POP3, FTP) but used mostly with HTTP

Also used for VPNs due to the many problems of IPsec

- Tunnelling TCP over TCP is problematic
  - Inner and outer layers of TCP flow control interact destructively
  - Outer TCP provides the appearance of a high-reliability but possibly high-latency link
  - Inner TCP tries to adapt to the traffic conditions created by the outer TCP, not of the actual link
  - Result: Meltdown
- Only occurs when the physical link characteristics are bad
  - Not noticeable on a LAN

TLS

Transport layer security

IETF-standardised evolution of SSLv3

- Non-patented technology
- Non-crippled crypto
- Updated for newer algorithms

Substantially similar to SSL

- Just different enough to be incompatible
- TLS identifies itself as SSL 3.1
- Current version is TLS 1.1 or SSL 3.2

TLS-PSK

TLS with pre-shared keys (TLS-PSK)

- Neatly sidesteps the X.509 PKI mess
- Provides mutual authentication of the client and server
  - Standard TLS only authenticates the server

Mix a pre-shared key (e.g. password) into the handshake

- master secret = hash(premaster secret + password)
- Handshake only succeeds if both the client and server know the secret value
  - Not just a global value (cert) but one specific to each user
- Proof-of-possession is done without either side revealing the password

This almost completely stops phishing attacks

TLS-PSK (ctd)

Using TLS-PSK to stop phishing requires two client changes

- Unambiguous indicator that TLS-PSK is in effect
  - Turn the URL bar light blue or green
  - Mozilla currently uses light yellow for one-sided cert use
- Unambiguous way to enter the TLS-PSK password
  - Not part of the normal web page
  - User can’t be spoofed into handing it over to a phishing site

Still quite new, not widely supported yet

SGC

Server Gated Cryptography

- Allowed strong encryption on a per-server basis
- Originally available only to “qualified financial institutions”
- Later extended slightly (hospitals, some government departments)

Required a special SGC server certificate from Verisign

- Presence of the SGC certificate turned on strong encryption when talking to the server with the certificate

Strong SSL Browsers

Netscape patched to do strong encryption

Original:

```
POLICY-BEGINS-HERE:                   Export policy
Software-Version:                     Mozilla/4.0
MAX-GEN-KEY-BITS:                     512
PKCS12-DES-EDE3:                      false
PKCS12-RC2-128:                       false
PKCS12-RC4-128:                       false
PKCS12-DES-56:                        false
PKCS12-RC2-40:                        true
PKCS12-RC4-40:                        true
…
SSL3-RSA-WITH-RC4-128-MD5:            conditional
SSL3-RSA-WITH-3DES-EDE-CBC-SHA:       conditional
…
```

Strong SSL Browsers (ctd)

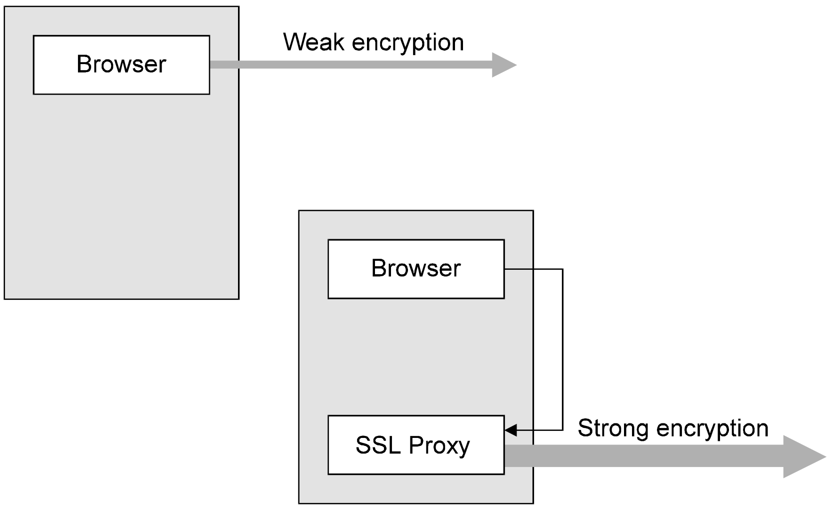
Patched version

```
POLICY-BEGINS-HERE:                  Cypherpunk policy
Software-Version:                    Mozilla/4.0
MAX-GEN-KEY-BITS:                    1024
PKCS12-DES-EDE3:                     true
PKCS12-RC2-128:                      true
PKCS12-RC4-128:                      true
PKCS12-DES-56:                       true
PKCS12-RC2-40:                       true
PKCS12-RC4-40:                       true
…
SSL3-RSA-WITH-RC4-128-MD5:           true
SSL3-RSA-WITH-3DES-EDE-CBC-SHA:      true
…
```

Today you can do this with Mozilla `about:config`

Strong SSL Proxies

Tunnel weak or no SSL over strong SSL



SSH

Originally developed in 1995 as a secure replacement for rsh, rlogin, rcp, et al (ssh = secure shell)

- Also allows port forwarding (tunneling over SSH)
- Built-in support for proxies/firewalls
- Includes zlib compression
- Can be up and running in minutes

Within a year or two of its appearance had almost completely displaced telnet, rlogin, rcp

- ssh is one of the great security success stories

Replaced by IETF-standardised SSHv2

- Several minor flaws in SSHv1 encouraged SSHv2 adoption

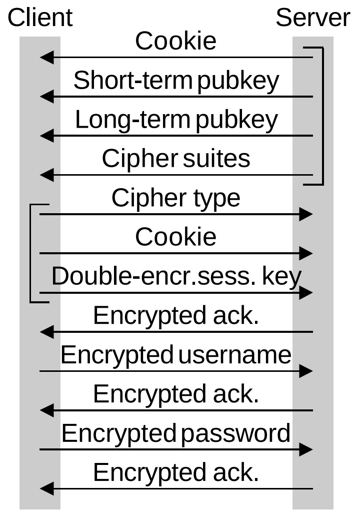
SSHv1 Protocol

Server uses two keys

- Long-term server identification key
- Short-term encryption key, changed every hour

Long-term server key binds the connection to the server

Short-term encryption key makes later recovery impossible



- Short-term keys are regenerated as a background task

SSHv1 Authentication

Multiple authentication mechanisms

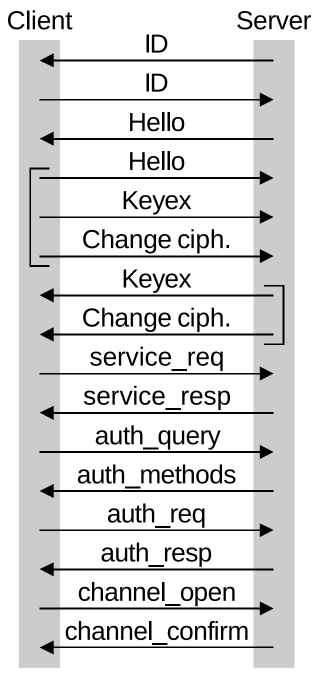
- Straight passwords (protected by SSH encryption)
- RSA-based authentication (client decrypts a challenge from the server, returns the hash to the server)
- Plug-in authentication mechanisms, e.g. SecurID

Developed outside the US, native support for strong crypto was a big deal at the time

- 1024 bit RSA long-term key
- 768 bit RSA short-term key (has to fit inside the long-term key for double encryption)
  – RSA was unpatented outside the US
- Triple DES session encryption (other ciphers available)

SSH Handshake

1. Negotiate the cipher suite
2. Establish a shared session key
   - Server signs its half of the key exchange data
3. Authenticate previously exchanged messages



SSH Handshake (ctd)

ID exchange

- Protocol version information as a text string
- Main real use is to work around implementation bugs

Server hello

- Server nonce
- Cipher suites

Client hello

- Client nonce
- Cipher suites

SSH Handshake (ctd)

Complex interlock mechanism to determine the greatest common denominator of client and server suites

- The process requires two pages of specification to describe
- Protocol allows both sides to send their hello simultaneously (!!)
- If there’s a disagreement, they both back off and try again

Key exchange

- Standard DH exchange

Both sides

- master secret = PRF(client nonce + server nonce +
  server key + DH keyex values +
  premaster secret)

SSH Handshake (ctd)

Client/server change cipher spec

- Switch to selected cipher suite and key

Long and chatty further exchange to actually set up the session

- Minimum of eight further messages sent/received

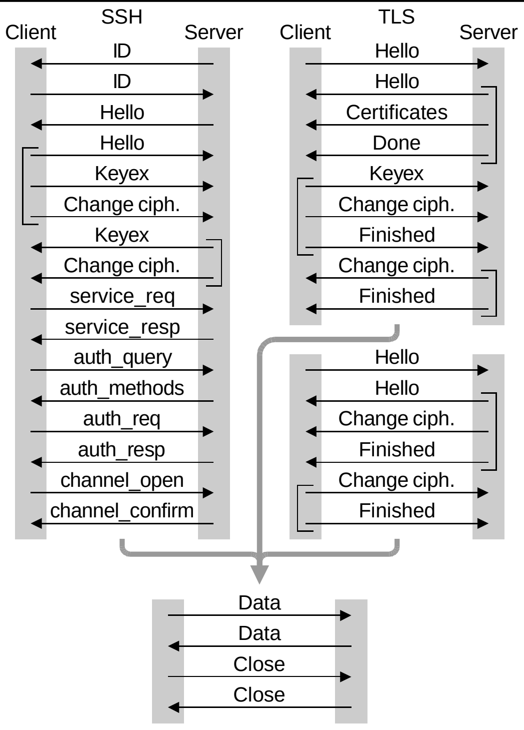
TLS vs. SSH

TLS: 2RTT

TLS resume or TLS-PSK: 1.5 RTT

SSH: 7.5 RTT

TLS authenticates the entire handshake, SSH only portions of some messages

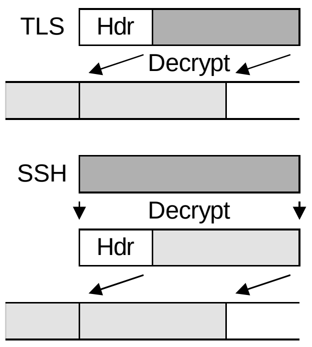


TLS vs. SSH (ctd)

TLS cipher suite selection is straightforward, SSH is awkward

SSH encrypts the packet length, requiring unnecessary transitions from the network to the crypto layer

- Interacts badly with high-performance I/O implementations that use posted reads
- SSH forces a pipeline stall in hardware cryptologic while the header is processed



IPsec

IP security — security built into the IP layer

- Provides host-to-host (or firewall-to-firewall) encryption and authentication
- Required for IPv6, optional for IPv4
  - Some IPv6 people advocated making it illegal to use with IPv4 in order to force a move to IPv6

Comprised of two parts

- IPsec proper (authentication and encryption)
- IPsec key management

IETF Politics

IPsec design was heavily influenced by IETF religious views

- End-to-end doctrine: All hosts should be visible and accessible over the Internet
  - NAT is evil and must be destroyed at all costs
  - Firewalls are bad (but not totally evil like NAT), since security should be done at the host
- Much IPsec (and IPv6 in general) design either ignores NAT/firewall considerations or is designed to deliberately break them

This explains some of IPsec's all-elbows design

IETF Politics (ctd)

Internet Architecture Board came up with a special acronym UNSAF (RFC 3424) to describe anything that worked with NAT

- Documentation can now refer to UNSAF systems, UNSAF clients, UNSAF servers, UNSAF protocols, …
- UNSAF designs must deliberately limit their own scope, include a self-criticism section explaining why they're no good, and provide an exit strategy

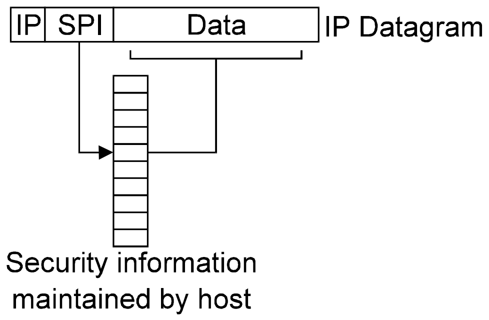
IPsec has a strange obsession with identity protection

- This greatly complicates session establishment
- You can't authenticate someone when you don't know who they are

IPsec Architecture

Key management establishes a security association (SA) for a session

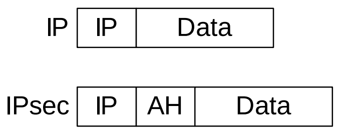
- SA is used to provide authentication/confidentiality for that session
- SA is referenced via a security parameter index (SPI) in the IP datagram header



AH

Authentication header — integrity protection only

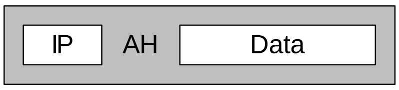
- Inserted into an IP datagram



- Designed by IPv6 fans, looks like a standard IPv6 extension header
  - Note that the MAC awkwardly precedes the data
- AH includes an integrity check value (ICV)
  - 96-bit HMAC
  - 96-bit ICVs have since ended up in other security protocols via cargo cult protocol design

AH (ctd)

Authenticates the entire datagram

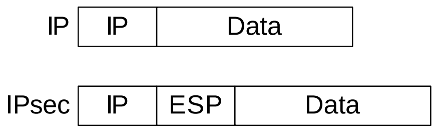


- Mutable fields (time-to-live, IP checksums) are zeroed before the AH is added
- Sequence numbers provide replay protection
- Receiver tracks packets within a 64-entry sliding window

ESP

Encapsulating security protocol — authentication (optional) and confidentiality

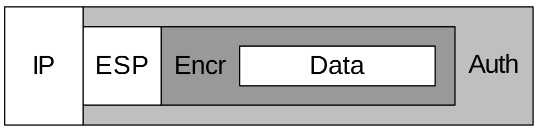
- Inserted into an IP datagram



- Designed by security people for ease of use/processing
- Contains sequence numbers and an optional ICV as for AH
  - This time the ICV follows the data

ESP (ctd)

Secures the data payload in a datagram



Encryption protects the payload

- Authentication protects the header and the encrypted payload

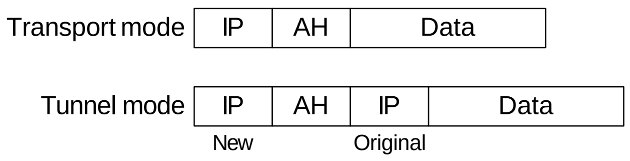
SA bundling is possible

- ESP without authentication inside AH
- Authentication covers more fields this way than just ESP with authentication

IPsec Modes

IPsec provides two modes

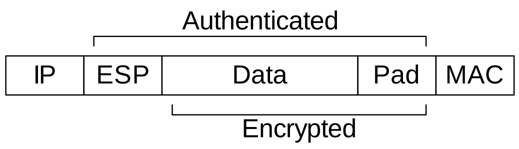
- Transport mode modifies the original packet
- Tunnel mode encapsulates the original packet unchanged



- Complex mixtures of tunnel and transport modes are possible

IPsec Packet Processing

Use the SPI to look up a security association (SA)

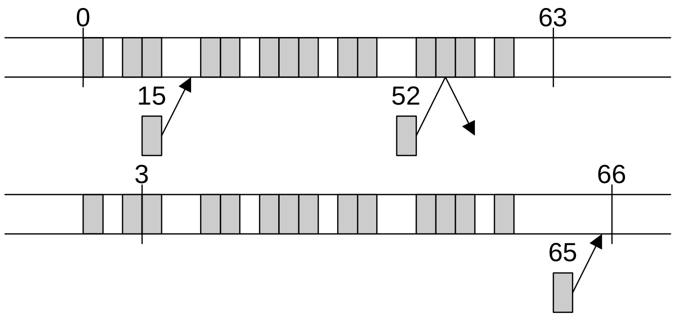


- Perform authentication check using the SA
  - Avoids the need to decrypt if the packet has been corrupted
- Perform decryption of the now-authenticated data using the SA

IPsec Sliding Window Algorithm

Provides message flow integrity protection

- IPsec itself runs atop unreliable transport



- Message sequencing is preserved
- Duplicate messages are rejected
- New messages advance the sliding window

IPsec Key Management

Need to understand its history to understand the design

ISAKMP

- Internet Security Association and Key Management Protocol

OAKLEY

- DH-based key management protocol

Photuris

- DH-based key management protocol

SKIP

- Sun’s DH-based key management protocol

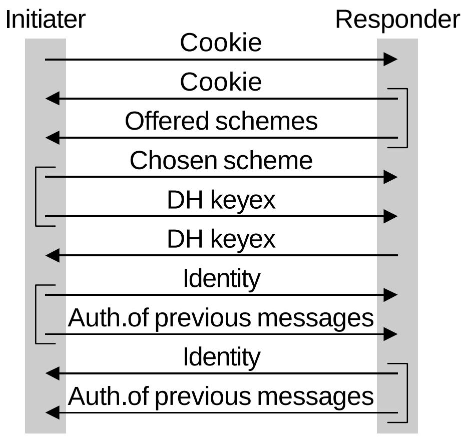
Protocols changed considerably over time, borrowing ideas from each other

Photuris

Latin for “firefly”, Firefly is the NSA’s key exchange protocol for STU-III secure phones

Three-stage protocol

1. Exchange cookies
2. Use DH to establish a shared secret
   Agree on security parameters
3. Identify other party
   Authenticate data exchanged in steps 1 and 2



n. Change session keys or update security parameters

Photuris Cookies

Attacker can request many key exchanges and bog down the host (clogging attack)

- Use a cookie to stops clogging attacks

Cookie depends on

- IP address and port
- Secret known only to the host
- Cookie = hash(source and dest.IP and port + host secret)

Host can recognise a returned cookie

- Attacker can't generate fake cookies

Later adopted by IKE, although they got it wrong

SKIP

Each machine has a public DH value authenticated via X.509 or PGP certificates

Public DH value is used as an implicit shared key calculation parameter

- Makes DH work more like RSA
- Shared key is used once to exchange an encrypted session key
- Session key is used for further encryption/authentication

Clean-room non-US version was developed by a Sun partner in Moscow

- US government forced Sun to halt further work on the non-US version

ISAKMP

NSA-designed protocol to exchange security parameters (but not establish keys)

- IETF: “This box appears to be addressed to Pandora”
  NSA: “Go ahead and open it anyway…”

Protocol to establish, modify, and delete IPsec security associations

- Provides a general framework for exchanging cookies, security parameters, and key management and identification information
- Exact details left to other protocols
  - In fact, significant portions of ISAKMP itself were underspecified

Possibly adopted as an excuse to avoid Photuris and SKIP

OAKLEY

Written to fit within the ISAKMP framework

Exchange messages containing any of

- Client/server cookies
- DH information
- Offered/chosen security parameters
- Client/server ID’s

until both sides are satisfied

OAKLEY (ctd)

Oakley is extremely open-ended, with many variations possible

Exact details of the messages exchanged depend on exchange requirements

- Speed vs. thoroughness
- Identification vs. anonymity
- New session establishment vs. re-key
- DH exchange vs. shared secrets vs. PKC-based exchange

SKEME

A more rigorous approach to designing a general session establishment framework

- Similar principles as OAKLEY, but taken from a pure crypto/mathematical point of view
- Elegant theoretical framework, but cost, complexity, and overhead aren’t considered

IKE

Internet Key Exchange: ISAKMP merged with OAKLEY and SKEME

- ISAKMP provides the protocol framework
  - Internet DOI defines the usage of the ISAKMP fields
  - Only the Internet DOI is ever used, IETF assigned values for RIP and OSPF but never defined the DOI
- OAKLEY and SKEME provide the security mechanisms
- Combined version clarifies both protocols, resolves ambiguities
  - Imagine "The Fly" done with security protocols

IKE (ctd)

Phase 1: Negotiate IKE SA to protect further exchanges

- Run infrequently

Phase 2: Negotiate further details protected by the IKE SA

- Run frequently to generate more SAs

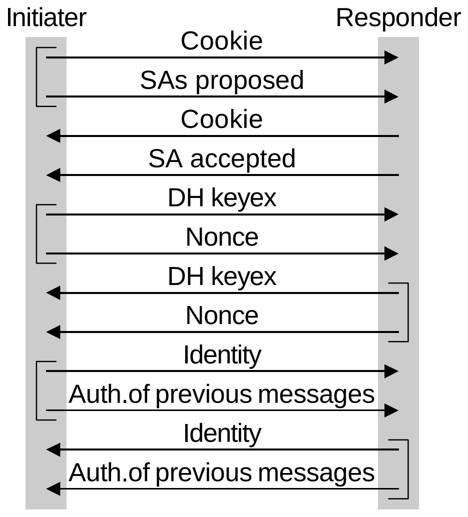
IKE Phase 1

Many, many variants are possible

- Data spread over more or less messages
- Authentication via digital signatures, public-key encryption, shared secrets, KDC

Unlike Photuris, the cookies don't allow statelessness

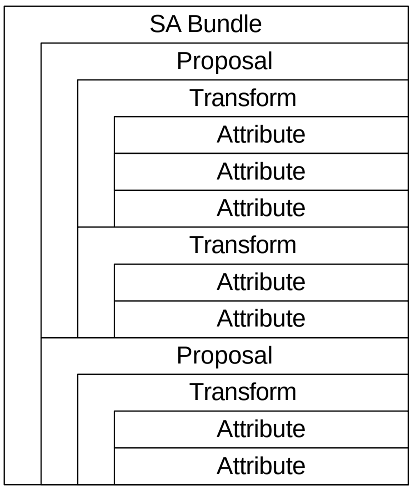
- Useless as an anti-clogging defence



IKE Phase 1 (ctd)

ISAKMP proposals consist of a multi-level hierarchy of values

- Security protocol
  - ESP
- Transforms for protocol
  - 3DES + HMAC-SHA1
- Attributes for transform
  - SA lifetime, key size, …



IKE Phase 1 (ctd)

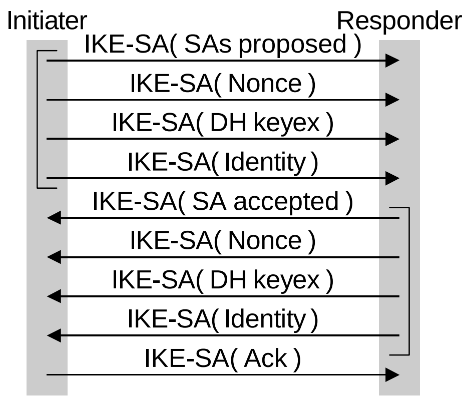
Transforms are listed in product-of-sums form

- { 3DES | DES } • { HMAC-SHA1 | HMAC-MD5 } →
  { 3DES + HMAC-SHA1 } or
  { 3DES + HMAC-MD5 } or
  { DES + HMAC-SHA } or
  { DES + HMAC-MD5 }
- Leads to a combinatorial explosion of combinations
- No control over algorithm combinations like SSL and SSH cipher suites

IKE Phase 2

Exchange is encrypted and authenticated using the previously established IKE SA

- SAs negotiated at this stage are used to protect payload data



IKE Modes

Four different exchange types

- Pre-shared secret keys (PSK)
- Digital signatures
- Public-key encryption
  - Original mechanism required four (!!) private-key operations, didn't identify the decryption key to use for the initiator, couldn't encrypt data larger than the modulus size
- Revised public-key encryption

IKE Modes (ctd)

Three different modes

- Main mode
  - Only mandatory protocol in main mode is PSK
  - Almost unworkable since it uses IP addresses as identities — Requires static IP on both sides
  - Unworkable with NAT
- Aggressive mode
- Base mode
- (Phase 2 also has Quick mode)

IKE Details

Censored

Too horrible to investigate

IKE Details (ctd)

Well OK, maybe a few representative examples

- You can’t rationally explain IKE, so let’s look at the irrational bits…

Each time IKE uses a PRF, it’s different

- Hashes, HMACs, cookies, nonces, and key data are mixed up arbitrarily
- Record is six different PRF types on one page (!!)

Main secret value is called SKEYID

- No-one knows why

IKE Details (ctd)

Each SKEYID derivation is different

- PSK SKEYID =
  PRF(PSK, initiator nonce || responder nonce)
- Signature SKEYID =
  PRF(initiator nonce || responder nonce, g^{xy})
- Public-key encryption SKEYID =
  PRF(hash(initiator nonce || responder nonce),
  initiator cookie || responder cookie))
- No-one knows why either
- Partially copied from the SKEME paper, where this usage didn't make much sense either

IKE Details (ctd)

Next level is SKEYID_d

- SKEYID_d =
  PRF(SKEYID, g^{xy} || initiator cookie || responder cookie || 0)

Next level is more SKEYIDs

- SKEYID_a (authentication) =
  PRF(SKEYID, SKEYID_d || g^{xy} || initiator cookie ||
  responder cookie || 1)
- SKEYID_e (encryption) =
  PRF(SKEYID, SKEYID_a || g^{xy} || initiator cookie ||
  responder cookie || 2)

Finally the keys

- K1 = PRF(SKEYID_e, 0)
- K*n* = PRF(SKEYID_e, *n*)

IKE Details (ctd)

IKE uses 32-bit random values for replay protection

- Not sequence counters but random values

Many other weird things happen in IKE

- Some fields are protected, some aren’t
- Various protocol fields are mixed into crypto ops for no known reason
- Crypto ops vary mysteriously depending on the exchange type and mode
- Exception cases frequently end up in undefined states
  - Get back an error, but can’t tell why
  - Negotiation halts/is dropped, and you have to start again

IKE: Design by counterexample

IPsec Problems

IPsec is excessively complex and difficult to use

- Specification is so difficult to understand that most texts just paraphrase it or reprint the RFCs verbatim
- AH is redundant, only ESP is necessary
- Transport mode is redundant, only tunnel mode is necessary
- Just these two changes produce a 4:1 complexity reduction for no noticeable loss

Unidirectional SAs double the complexity of SA management

- Unidirectional SAs give you the flexibility to use 3DES in one direction and RC4/40 in the other
- Neither SSL/TLS nor SSH feel a need for unidirectional SAs

IPsec Problems (ctd)

IPsec breaks NAT

- “It’s OK, IPsec will prove the bigger hammer”
- Uhh, yeah…
  - Some IPv6 advocates have gone out of their way to design NAT-breaking protocols
  - They like AH because it’s incompatible with NAT’s address-rewriting
- Routers need to be made IPsec-aware
- IPsec implementations need to be made NAT-aware
  - Only ESP tunnel mode is possible
  - Biggest problem is running the IKE negotiation through the NAT

IPsec Problems (ctd)

NAT support was finally (officially) kludged onto IKE in 2005

- Exchange a hash of both sides’ IP addresses and ports to detect if NAT is changing it

Some NAT devices to their own IPsec handling

- Don’t rewrite traffic to IKE source port 500, use cookies to demultiplex traffic
- Interacts negatively with IKE NAT detection

Ongoing arms race between NAT-aware IKE and IKE-aware NAT

IPsec Problems (ctd)

IKE is unworkable

- Configuring IKE is equivalent in complexity to setting up an X.25 link
  - I dislike IKE — Cisco badge handed out at trade shows
  - Experienced network engineer budgets two working days to set up an IPsec link when different vendor hardware is used
- Users rely on hand-carrying shared keys around
- Vendors kludge on “management tunnels” to bypass IKE
  - Homebrew protocols created without any security review
  - Typical management tunnel: Single DES, fixed password

OpenVPN

“The Internet views IPsec as damage and OpenVPNs around it”

Combines the best features of IPsec and TLS

- ESP in tunnel mode from IPsec
- TLS handshake from TLS

OpenVPN’s great advantage: It just works

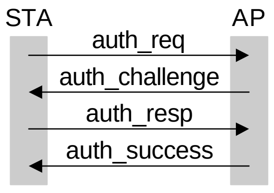
- OpenVPN is sneaking into companies like Linux did in the 90s

WEP

Wired Equivalent Privacy (well, in name anyway)

- Attempt to add some measure of security to 802.11

Provides security between a station (STA) and an access point (AP)



- After this initial exchange, no further authentication is performed
  - Subsequent messages were simply automatically trusted
- WiFi profile dropped this stage as pointless

WEP (ctd)

It’s actually worse than useless

- AP sends 128-bit plaintext
- STA responds with encrypted plaintext
- Attacker now has plaintext/ciphertext pair to try passwords on
- Algorithm used is RC4, can trivially recover 128 bits of keystream
  - Using the recovered keystream, the attacker can then authenticate themselves
- Only authenticates the STA, not the AP
  - Doesn’t protect against rogue APs
- Vulnerable to all the other standard attacks on weak authentication protocols

WEP Encryption

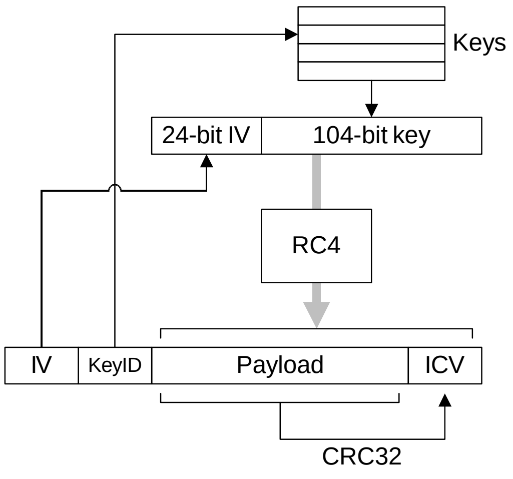
Two keying modes

- Default key (a.k.a. “shared key”, “group key”, “multicast key”, “broadcast key”)
  - Shared among all STAs using an AP
- Key mapping key (a.k.a. “individual key”, “per-station key”, “unique key”)
  - One key for each STA using an AP
  - Broadcast messages are still sent using the default key

AP stores multiple generations of keys to make key rollover seamless

WEP Encryption (ctd)

- Look up the 104-bit fixed key using the keyID
- Add the 24-bit variable IV to the key to get a 128-bit RC4 key
- Decrypt the payload + ICV
- Calculate the CRC32 on the packet



WEP Problems

Every single part of WEP is broken

Use of IVs (and RC4) is incorrect

- Many systems start IVs at the same value after a restart
- Many systems update IVs in a predictable manner
- Even if IVs are random, they roll over after as little as 6-7 hours
  - All of these lead to RC4 keystream re-use
  - If a default key is being used with devices that start at a fixed IV value, collisions will occur immediately

WEP Problems (ctd)

CRC-32 isn’t a MAC

- Can be defeated without needing to attack RC4
- Flip bits in the ICV to match bits flipped in the payload
- Flips bits in the payload to cancel out changes in the ICV

No replay protection

- Replay an old authentication message
- Replay network control messages
  - Can guess these based on their length and/or traffic patterns
  - Attacks like SYN floods are trivial

WEP Problems (ctd)

RC4 keying problems

- RC4 has weak keys
- A number of bits in the first few bytes of RC4 output are directly determined by the key
  - It takes a few cycles for the RC4 state table to become well-mixed
  - RC4 usage guidelines recommend discarding the first few dozen bytes of output
- Because of WEP's per-packet re-keying, it provides this output for *every packet sent*
- First few bytes of packet plaintext are usually easy to guess
  - 802.1 LLC SNAP header bytes 0xAA

WEP Problems (ctd)

- Prepending the changing IV to the fixed key makes it the most vulnerable
- Problems first pointed out in Fluhrer, Mantin, and Shamir (FMS) paper in 2001

Ever-changing IV guarantees that weak keys will be used

- Wait for a weak-key IV, then attack that packet
- Can guess the first key byte after approximately 60 messages
- Walk down the key getting each byte in turn

WEP Problems (ctd)

No DoS protection

- Management messages have no protection
- Send forged deauthenticate or disassociate messages
  - Can also be done from the LAN on which the AP resides
- Load an AP with bogus STA requests

Session hijacking possible

- Send deauthenticate or disassociate messages to the STA
- Continue the session with the AP while pretending to be the (now-disconnected) STA

WEP Attacks in Practice

A large number of tools exist to attack every aspect of WEP

- Airsnort finds keys using the FMS attack
  - Declining in popularity as devices are updated to minimise weak-key IVs (advertised as “WEP+”)
  - Of course, that means IVs roll over faster…
- Aircrack and WepLab handle a broader range of weak keys
  - Aren’t affected by WEP+
- Aireply harvests network management packets and replays them

WEP Attacks in Practice (ctd)

- WEPWedgie creates e.g. SYN floods using the keystream from sniffed challenge/response pairs
- WEPAttack performs a dictionary attack on sniffed packets

These are only samples to show that every possible weakness is readily exploitable, there are many more tools out there

- WEP may be breakable in some circumstances — vendor literature
- “circumstances” = “the equipment is switched on”

WPA

WiFi Protected Access, an attempt to fix the WEP mess

- WEP was an unmitigated disaster
- Vendors introduced a confusing mass of terminology to distance themselves from the term “WEP”

WEP+

- Trivial patch to fix one particular attack type

WPA (or “WPA with TKIP”)

- Band-aid fix that can generally be applied with a firmware upgrade

WPA2 (or “WPA with AES” or “AES-CCM”)

- Better long-term fix requiring more extensive re-engineering

TKIP

Temporal Key Integrity Protocol

- Attempt to patch up WEP in a backwards-compatible manner
- Had to run on existing hardware and fit in the existing WEP framework
  - This was a *severe* restriction

Fixes the existing WEP weaknesses

- Uses a message integrity check (MIC) instead of CRC32 for the ICV
  - Normally called a MAC but that acronym was already taken
- Uses the IV as a sequence counter
- Changes the entire encryption key for each packet
- Increases the IV size to avoid IV reuse

TKIP (ctd)

MIC is performed using a custom-designed MAC called Michael (Mic-hael)

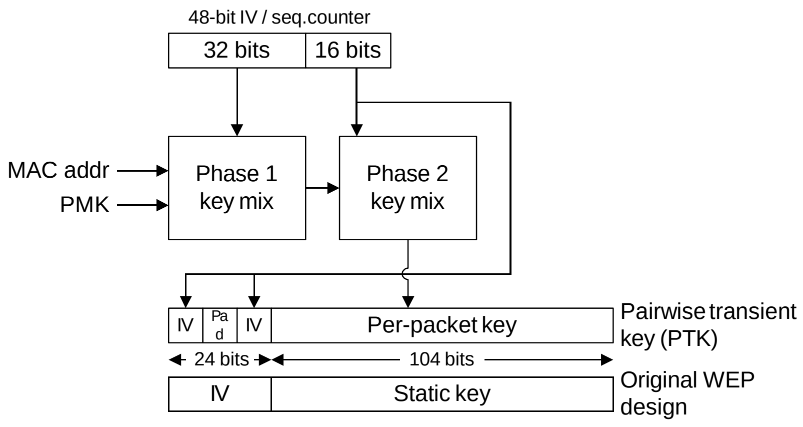
- Had to be simple enough to be implementable on low-powered CPUs in STAs/APs
- Simplicity makes it somewhat vulnerable to brute-force attacks
- Counter this by suspending communications for 60s when a MIC failure is detected
  - Balance between security and DoS protection

IV security is increased

- Expanded from 24 to 48 bits
- Specially constructed to avoid (most) types of weak keys
- Serves double duty as a sequence counter for replay protection

TKIP Key Management

Hash the user-supplied key and SSID 4096 times with PBKDF2 to get the 256-bit pairwise master key (PMK)



- Apply the PMK to a two-phase key generation process

TKIP Key Management (ctd)

IV management is extended

- 48-bit IV is used as a sequence counter
  - TKIP sequence counter, TSC
  - Split into high 32 bits (mostly static) and low 16 bits (constantly changing)
- Sequence counter is used to implement an IPsec-style 16-entry sliding window
  - Eliminates replays
- 8-bit pad in transmitted IV value is used to avoid most weak keys
  - Better would be to discard the initial output bytes, but existing hardware doesn’t support this

TKIP Key Management (ctd)

Key-mixing is CPU-intensive

- Solved with a two-phase mixing process
- Phase 1 is static and one-off
  - High 32 bits of the IV only change every 64K packets
- Phase 2 is per-packet
  - Since the counter is predictable, phase 2 can be computed in advance while waiting for the next packet(s) to arrive

TKIP is truly remarkable recovery from the WEP mess

Attacks on TKIP

As for WEP, attack tools have been created for TKIP

- WPACracker, coWPAtty perform dictionary attacks on TKIP-protected packets
  - Dictionary $\rightarrow$ PMK $\rightarrow$ PTK $\rightarrow$ trial decrypt to see if the key matches
- Solution: Use passphrases of 20 chars or more
- Alternatively, set the 256-bit PMK directly rather than generating it by applying the PBKDF2 transform to a password and the SSID

Attacks on Auth Mechanisms

Attacks on additional WEP/WPA security mechanisms are also possible

- 802.1*x* specifies the use of EAP for 802.*x* networks
- Most popular EAP mechanism is Cisco's lightweight EAP, LEAP
- LEAP uses the broken MS-CHAPv2 mechanism
  – See the section on authentication for details
- Anwrap, THC-LEAPcracker, and asleap allow password recovery using a precomputed dictionary

AES-CCM

AES counter mode + CBC-MAC

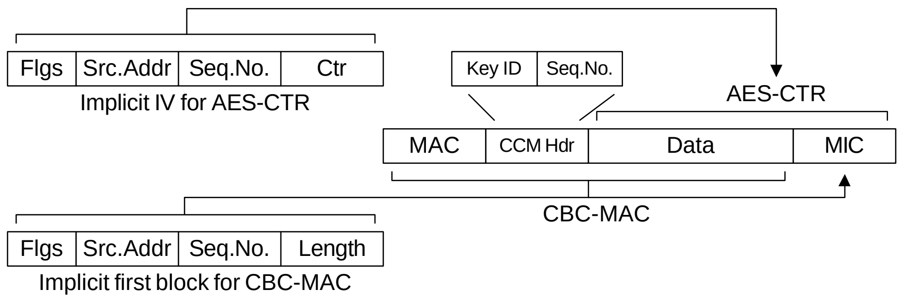
- Combined encrypt + MAC mode
- OCB was the first choice, but it was encumbered
  – Many APs run open-source OSes or use open-source toolkits
- Counter mode = like CFB but encrypts a counter rather than feeding back ciphertext
- CBC-MAC = process data in CBC mode, last output block is the MAC
  – Popular before HMAC came along (DES-MAC)
  – Used rather than the HMAC because AES was conveniently available

AES-CCM (ctd)

CCM adds extra features for 802.11 use

- Authenticates further data (MAC address, management information, additional out-of-band data) that must remain unencrypted

AES-CCM (ctd)



- Mutable fields in the MAC header are set to 0 as for IPsec AH
- MIC is truncated to 64 bits
- Nonce guarantees uniqueness in the case of keys shared by multiple users
- AES-CTR IV used for each packet is similar to the CBC-MAC first block
  - Counter field is incremented for each block in the packet

AES-CCM continued

Both CBC-MAC and AES-CTR require an exact number of AES data blocks

- Authenticated and encrypted data is padded with zeroes before processing
- Zero padding is only used for encryption/MIC computation, but isn't transmitted

DNSSEC

DNS name space is divided into zones, each zone has resource records (RR's)

```
Owner_name Type Class TTL Rdlength Rdata
```

- Owner = name of node
- Type = RR type
  - A, AAAA = Host address
  - NS = authoritative name server
  - CNAME = canonical name for alias
  - SOA = start of zone authority
  - PTR = domain name pointer
  - MX = mail exchange
- Class = IN (Internet)
- TTL = time for which RR may be cached

DNSSEC (ctd)

Name servers hold zone information

- Each zone has primary and secondary servers
- Secondaries perform zone transfers to obtain new data from primaries

Resolvers extract information from name servers

- Cached entry is returned directly
- Interative query returns referral to the appropriate server
- Recursive query queries other server and returns the result

All of these points present security vulnerabilities

DNSSEC (ctd)

DNSSEC splits the service into a name server and a zone manager

- Zone manager signs zone data
- Name server publishes signed data
  - Compromise of the name server doesn’t compromise DNSSEC

Resolvers need to store at least one top-level zone key

DNSSEC (ctd)

RR’s are extended with new types

- KEY, server public key
- SIG, signature on RR
- NXT, chains from one name in a zone to the next
  - Allows authenticated denial of the existence of a name
- RR’s have signature start and end times
  - Requires synchronised clocks on hosts

DNSSEC (ctd)

Transaction signature guarantees that the response came from a given server

- Signature covers query and response

Also used for

- Secure zone transfers
- Secure dynamic update (replaces editing the zone’s master file)
- Offline update
  - Uses authorising dynamic update key for update
  - Zone data is signed later with the zone key

DNSSEC Problems

Nasty corner cases where trivial configuration errors can cause serious problems

- DNSSEC error reporting is poor

Model for updating keys at the top levels of the heirarchy: Don’t do it

Requires absolute time synchronisation between servers rather than just relative/elapsed time like DNS

- PKI protocols also have a habit of pulling time into the TCB

DNSSEC is rarely used

- High overhead for little apparent gain

S-HTTP

Security extension for HTTP rather than the surrounding transport

- Secures each message in the session rather than the overall session
- Predates SSL and S/MIME

Document-based

- (Pre-)signed documents
- Encrypted documents

Not supported by browsers (or much else)

S-HTTP (ctd)

Example S-HTTP exchange

```
200 OK HTTP/1.0
Encryption-Identity: DN-1779, null,
  CN=Sample User, O=Sample CA, C=US;
Certificate-Info: CMS,
  MIAGCSqGSIb3DQEHAqIACAQExAG9w0BBwEAAKCAM
  …
Content-Privacy-Domain: CMS
…
```

- Header indicates
  - Sender DN
  - Sender certificate
  - Encryption type (Cryptographic Message Syntax, CMS)

S-HTTP (ctd)

Response

```
Secure * Secure-HTTP/1.4
Content-Type: message/http
Content-Privacy-Domain: CMS

MIAGCSqGSIb3DQEHA6CAMIACAQAxgDCBqQIBAD
BTME0xCzAJBgNVBAYTAlVTMSAwHgYDVQQKExdS
U0EgRGF0YSBTZWN1cml0eSwgSYFKw4DAgcEX6A
…
```

Message is encrypted using the given certificate and the CMS data format

- CMS is the format also used in S/MIME

SNMP Security

SNMP = Security Not My Problem

- Security model: Block it at the router

Authentication: hash(secret value + data)

Confidentiality: encrypt(data + hash)

- More recent versions add more sophisticated security, but they’re rarely used

Many devices are too limited to handle the security themselves

- Handled for them by an element manager
- Device talks to the element manager via a shared key

SNMP Security (ctd)

Users generally use a centralised enterprise manager to talk to element managers

- Enterprise manager is to users what the element manager is to devices