# Standard Security Protocols

*Going up the stack:*

*PKI, DNS, TLS, IPsec and all that*

# Why standardise?

- Generic reasons:
  - Multi-Vendor Interoperability
  - “Open”-ness
- Security Reasons
  - Remember: Crypto => Interop is hard!
  - Review: Many (non-security) standards decrease security
- NB: Not everything needs to be standardised!

# A Possibly New Why?

- Human Rights Protocol Considerations Research Group
  - https://irtf.org/hrpc
- “Rethinking Privacy Online and Human Rights: The Internet’s Standardisation Bodies as the Guardians of Privacy Online in the Face of Mass Surveillance”
  - Adamantia Rachovitsa, Conference Paper No.5/2016 2016 ESIL Research Forum, Istanbul, 21-22 April 2016 (see materials page for copy and link to original)

# Standards Devlopment Organisations (SDOs)

- International organisations
  - e.g.: UN/ITU, ISO
- "Open" Internet Standards Development Organisations
  - e.g.: IEEE, IETF, W3C
- Commercial Enterprise Driven SDOs
  - e.g.: OASIS, FIDO Alliance
- Company-specific pet projects
  - e.g.: FB Free basics/internet.org
- Open-source projects
  - e.g.: Apache, WHATWG, OpenSSL, ...
- Open-source commercial alliances
  - e.g.: OpenStack
- Operational entities:
  - e.g. ICANN, RIPE, ARIN,...
- Topic specific alliances
  - e.g. M3AAWG, Lora alliance, ...

# Types of standards

- All SDOs have difference categories of standard
  - RFC1149; http://www.blug.linux.no/rfc1149
  - Draft versions may be published or not, long-lived or not, rubbish or not
- Almost all interesting standards are openly available - some for a fee!
- But note: All SDOs have some business model, even the best ones:-)

# ISO/ITU-T

- National bodies are members (NIST, Enterprise-Ireland)

https://www.iso.org  https://www.itu.int

- Security stuff:
  - Child online protection (ITU?)
  - Cryptographic mechanisms (ISO)
    - Recent good news there wrt “lightweight” crypto/NSA: https://www.schneier.com/blog/archives/2017/09/iso_rejects_nsa.html
  - Even more X.509 (PKI stuff ITU-T)
  - Various ITU telephony specs
    - Some inheriting from/profiling IETF

# Internet Engineering Task Force (IETF)

- No members: a group of individuals developing the Internet

https://www.ietf.org https://www.irtf.org

- Security:
  - About 1-2 dozen of about 100 working groups usually doing interesting security  stuff
- Best of a bad lot really! (But I would say that:-)

# World-Wide-Web Consortium (W3C)

- Membership organisation ($5-50k per annum) plus strong “team” plus invited experts

https://www.w3.org

- Some interesting old security groups
  - XML Sig, Enc, XKMS
- Currently:
  - WebRTC
  - Focus more on browser APIs and not protocols

See also: WHATWG [https://whatwg.org/](https://whatwg.org/)

# IEEE Standards Association

- Individual memberships but meeting attendance counts

https://www.ieee.org/

- Security:
  - IEEE 802 – various security things, WPA etc.
  - MAC address randomisation,
  - MACsec (layer 2 crypto) now (Feb 2020) also considering MAC address privacy
    - https://1.ieee802.org/security/802-1aedk/

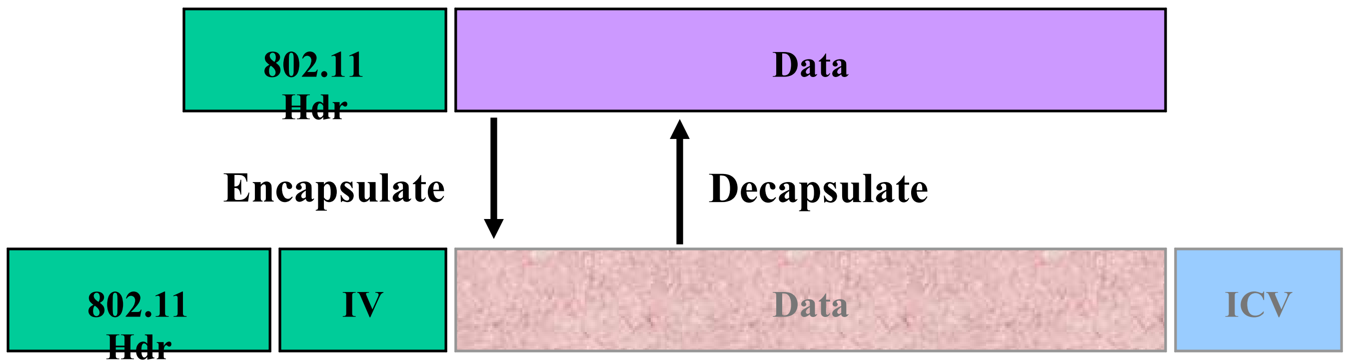
# Examples of “standards”

- Really good: RFC822/2822/5322 – Mail message format
- Middling: RFC3185 “Re-use of CMS Content Encryption Keys”
  - (Ahem!) Co-author present :-)
  - Length: 10pp (-crud=3pp)
  - Duration: ~18 months
  - Purpose: Fix a problem for RADIUS/Diameter
  - BUT: Zero implementations

# “Bad” examples

- Many to choose from
- IETF: IKEv1 (see later)
- IEEE: WEP (or is it?)

# WEP Encapsulation

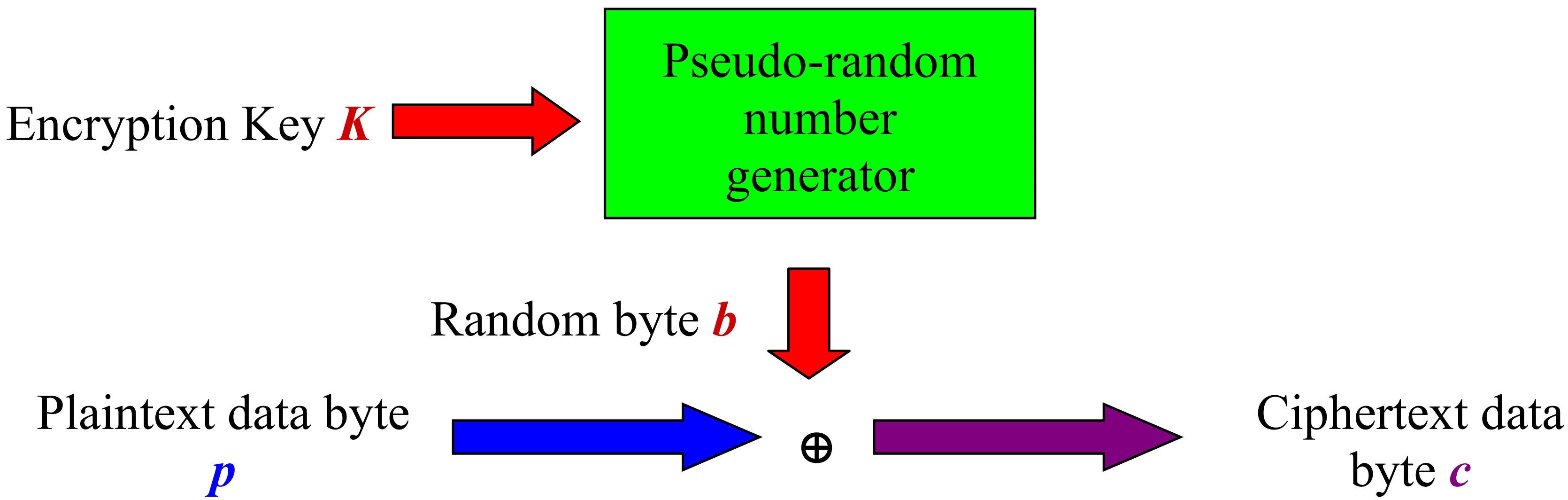



WEP Encapsulation Summary:

- Encryption Algorithm = RC4
- Per-packet encryption key = 24-bit IV concatenated to a pre-shared key
- WEP allows IV to be reused with any frame
- Data integrity provided by CRC-32 of the plaintext data (the "ICV")
- Data and ICV are encrypted under the per-packet encryption key

# Properties of Vernam Ciphers (1)

The WEP encryption algorithm RC4 is a Vernam Cipher:




Decryption works the same way: $p = c \oplus b$

# Properties of Vernam Ciphers (2)

***Thought experiment 1***: what happens when $p_1$ and $p'_2$ are encrypted under the same "random" byte $b$?

$$c'_1 = p'_1 \oplus b \qquad\qquad c_2 = p_2 \oplus b$$

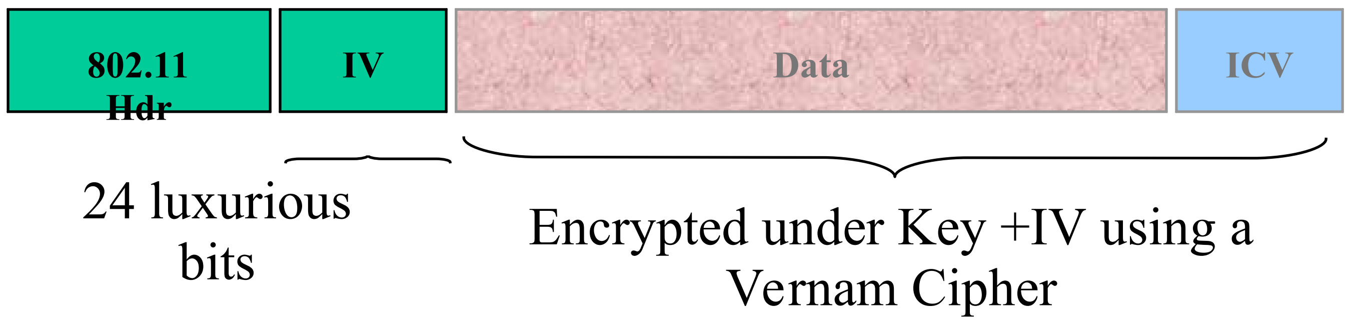
Then:

$$c'_1 \oplus c_2 = (p'_1 \oplus b) \oplus (p_2 \oplus b) = p'_1 \oplus p_2$$

***Conclusion***: it is a very bad idea to encrypt any two bytes of data using the same byte output by a Vernam Cipher PRNG.

***Ever.***

# How to Read WEP Encrypted Traffic (1)




- By the Birthday Paradox, probability $P_n$ two packets will share same IV after n packets is $P_2 = 1/2^{24}$ after two frames and $P_n = P_{n-1} + (n-1)(1-P_{n-1})/2^{24}$ for n > 2.
- 50% chance of a collision exists already after only 4823 packets!!!
- Pattern recognition can disentangle the XOR'd recovered plaintext.
- Recovered ICV can tell you when you've disentangled plaintext correctly.
- After only a few hours of observation, you can recover all $2^{24}$ key streams.

# How to Read WEP Encrypted Traffic (2)

- Ways to accelerate the process:
  - Send spam into the network: no pattern recognition required!
  - Get the victim to send e-mail to you
    - The AP creates the plaintext for you!
  - Decrypt packets from one Station to another via an Access Point
    - If you know the plaintext on one leg of the journey, you can recover the key stream immediately on the other
  - Etc., etc., etc.

# When to standardise

- When many implementer’s codebases have to talk a protocol
  - HTTP, SMTP,... (many, many examples)
- When one implementer has to use another vendor's API
  - WebRTC, PKCS#11
- When serious review is required
  - Routing (BGP) or TCP changes like RED, AQM
  - Crypto algs, e.g. AES, PQ algs

# When not to…

- When you just want your name in “lights”
- When your clever algorithm is the tenth way to do the job
- When your scheme is patented
  - Or secretly about to be patented!
- When no-one cares
- See RFC 6417 for guidance for researchers

# More on when not to…

- If you’re an open-source team and don’t have the cycles to engage with all the nuts who’ll get involved when you engage in a really open process (and they will) – e.g. Tor
- If you claim that implementation agility and speed is more important than multi-implementer interop - e.g. Signal, maybe wireguard

# Standard Security Structures and Protocols

*PKI, S/MIME, SSL, Kerberos, IPsec, SSH, Wireguard*

*Basically, things that do automated key management and secure application data transport*

# Materials

- Lots of RFCs
- https://datatracker.ietf.org/wg/
- Bleichenbacher/"Avoiding the million-message attack" RFC 3218

# Next hour(s)...

- PKI model and protocols
- SMIME formats and secure email
- **TLS protocol (TLS1.2 or TLS1.3)**
- IPsec
- Kerberos
- SSH
- Wireguard

Knowing **one** of these in detail is enough for exam purposes – you **will** need to read the source materials

# Public Key Infrastructure (PKI)

- Key management is a scaling problem
- Possible to push “trust” (bad term!) up to a certification authority (CA)
- “Trust” here is: CA is responsible for binding information about an entity (esp. Name) with a public key
  - Anyone who trusts that CA for that purpose can then check that entity's signature or encrypt to it

# PKI History

- Original public key concept involved publishing public keys in a newspaper
- Computing equivalent suggested in early 1980's
- First standard was X.509 in 1988
- IETF X.509 profile is in **RFC 5280** from 2008
- Used by TLS, S/MIME and lots of other protocols

# X.509-based PKI Problems

- Everyone sensible hates this stuff
  - It's really old, gnarly & horrible
- Every now and then someone suggests replacing it with <foo>
  - So far, no <foo> has been sufficiently better to displace X.509 based PKI (sadly)
- Maybe in 5-10 years it'll be less important, but for now we have to suffer with it.

# Certificates (1)

```
Certificate  ::=  SEQUENCE  {
 tbsCertificate       TBSCertificate,
 signatureAlgorithm  AlgorithmIdentifier,
 signatureValue       BIT STRING  }
```

- Who knows what ASN.1 is?
  - An abstract syntax notation
  - With tag, length value encoding schemes (BER, DER, PER)
    - SEQUENCE -> 0x30, INTEGER -> 0x02
  - PITA

# Certificates (2)

```
TBSCertificate  ::=  SEQUENCE  {
   version         [0]  EXPLICIT Version DEFAULT v1,
   serialNumber         CertificateSerialNumber,
   signature            AlgorithmIdentifier,
   issuer               Name,
   validity             Validity,
   subject              Name,
   subjectPublicKeyInfo SubjectPublicKeyInfo,
   issuerUniqueID [1]IMPLICIT UniqueIdentifier OPTIONAL,
   subjectUniqueID[2]IMPLICIT UniqueIdentifier OPTIONAL,
            -- If present, version MUST be v2 or v3
   extensions      [3]  EXPLICIT Extensions OPTIONAL
                 -- If present, version MUST be v3
   }
```

# Certificate Revocation Lists

- A list of "bad" certificate serial numbers (plus list- and entry-extensions)
- Periodically issued by a CA
- Revocation = putting on black-list
- Also via on-line certificate status protocol (OCSP)
- Reasons to revoke: Key compromise, Key loss, Change of function, Uninstall web server…

# To check a certificate

- Start with a (set of) trusted CAs
- Progress down certification path:
  - Is next-signature ok with previous public?
  - Is certifcate revoked?
  - Continue
- Missing lots (see rfc5280)
  - E.g. Policy mappings (yuk!)

# PKI Entities

```
+---+
| C |                       +------------+
| e | <-------------------->| End entity |
| r |       Operational     +------------+
| t |       transactions          ^
| i |      and management         |  Management
| f |       transactions          |  transactions        PKI
| i |                             |                     users
| c |                             v
| a | =======================  +--+------------+  ==============
| t |                          ^               ^
| e |                          |               |         PKI
|   |                          v               |      management
| & |                       +------+           |       entities
|   | <---------------------|  RA  |<----+     |
| C |  Publish certificate  +------+     |     |
| R |                                    |     |
| L |                                    |     |
|   |                                    v     v
| R |                                +------------+
| e | <------------------------------|     CA     |
| p |   Publish certificate          +------------+
| o |   Publish CRL                     ^     ^
| s |                                   |     |  Management
| i |                +------------+     |     |  transactions
| t | <--------------| CRL Issuer |<----+     |
| o |   Publish CRL  +------------+           v
| r |                                      +------+
| y |                                      |  CA  |
+---+                                      +------+
```

# PKI Protocols

- You need some to operate a PKI
- Registration & renewal: PKCS#10, proprietary, CMP, CMC, EST and now **ACME**
- Certificate retrieval: **in-band,** LDAP, DAP, HTTP, FTP, DPD
- Certificate status checking: **OCSP**, DPV, CRL processing, and now (maybe) application specifics like CRLite
  - https://blog.mozilla.org/security/2020/01/21/crlite-part-3-speeding-up-secure-browsing/
  - Mind you, I've heard nothing more since
- XKMS was a W3C attempt to do similar things
  - Deployment didn't happen
- Certificate Transparency (**CT**) for logging certificate issuance

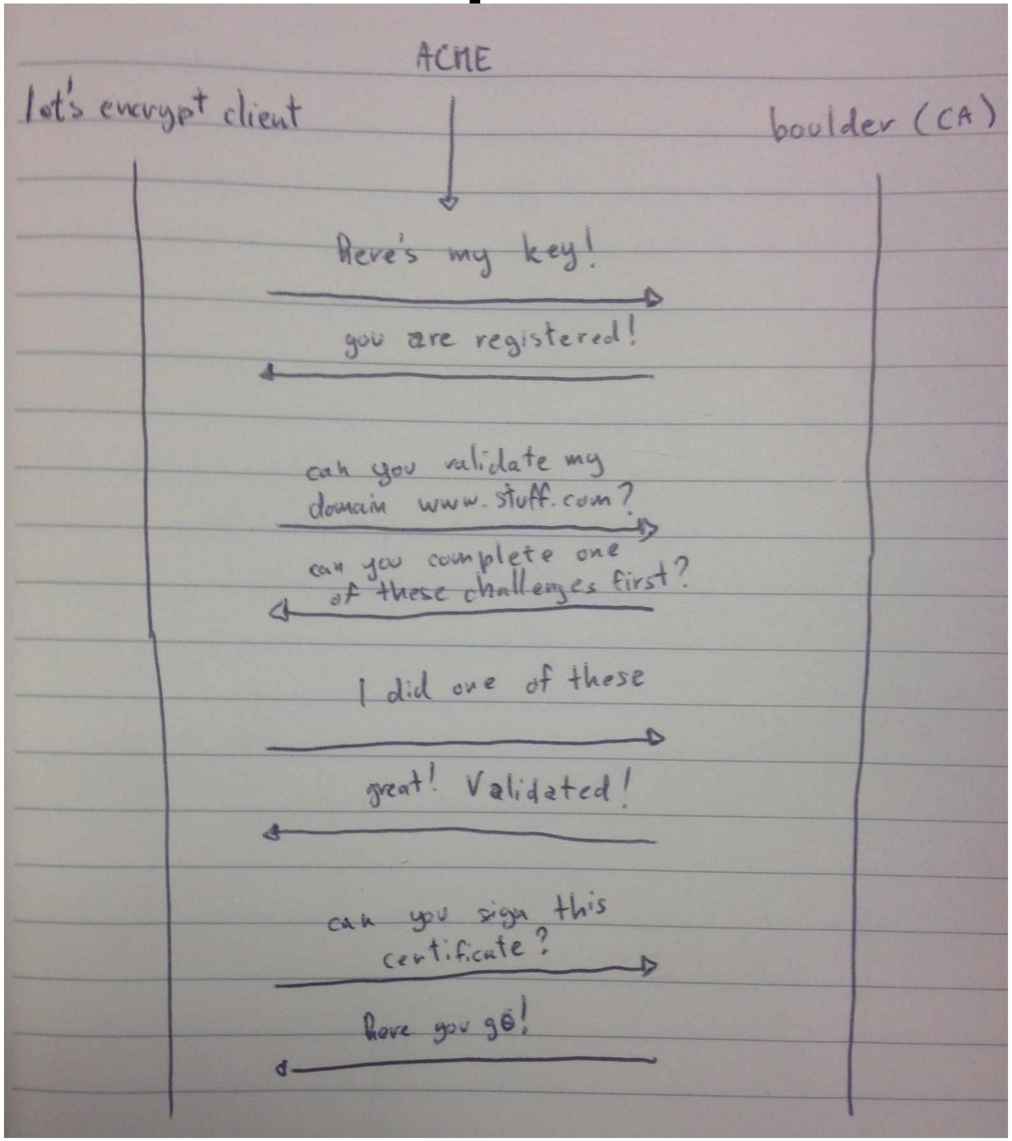
# Roots/Trust Points

- Applications using PKI need to have a set of root CA public keys they “trust”
- Browsers have those – each with O(100) CAs in the list
- In/ex-clusion is highly political
  - https://cabforum.org/ is a venue for some of that politics
- The WebPKI is a special case today – biggest and most important PKI but quite a few others do exist
- Other applications and OSes handle things similarly, typically with some overlap with WebPKI

# ACME protocol

- Automated Certificate Management Environment (ACME) is an HTTP-based protocol for managing certificates
  - Works with LE and other CAs for acquiring Domain Validated (DV) certificates
  - RFC 8555
- Quite a few client implementations:
  - Certbot, ACME.sh, ...
- Typically: command line registration then (weekly) cron job for renewal

# ACME protocol




https://cryptologie.net/article/274/lets-encrypt-overview/

# ACME protocol

- ACME challenges:
  - Evidence for domain validation
  - There’s a history of these turning out tricky in some deployment scenarios
- Challenge types:
  - http-01: random content at e.g. “/.well-known/acme-challenges/abcdef0123456”
  - dns-01: As above but in DNS
  - tls-sni-01/02: deprecated! Demonstrate ability to use random TLS SNI value

# Certificate Transparency (CT)

- There have been cases where CA’s have been hacked or misbehave (more later)
- CT improves the WebPKI
  - CT specified in (non-standard) RFC6962
  - Standardisation process has been sloooow…. https://datatracker.ietf.org/wg/trans/documents/
- Idea: append-only public logs of all certificates issued to allow detection of mis-issuance (not prevention, detection!)
- Nice (if not quite reliable) search UI:
  - https://crt.sh/

Reminder to self: pop up this URL and show stuff: https://crt.sh/?q=tcd.ie&dir=v&sort=2&group=none

# X.509-based PKI Summary

- Mix of mature and new(ish) technology
  - Plenty of open-source, products and services and significant deployment experience
- Deployment and application integration problems will always exist
  - Can be overcome but expensive
- Still technology-of-choice for scalable key management
- Improvement is possible: e.g. ACME/CT
- Might be affected by post-quantum crypto
  - Or, hopefully PQC will tip us into finally moving to something better

# E2E mail security: S/MIME

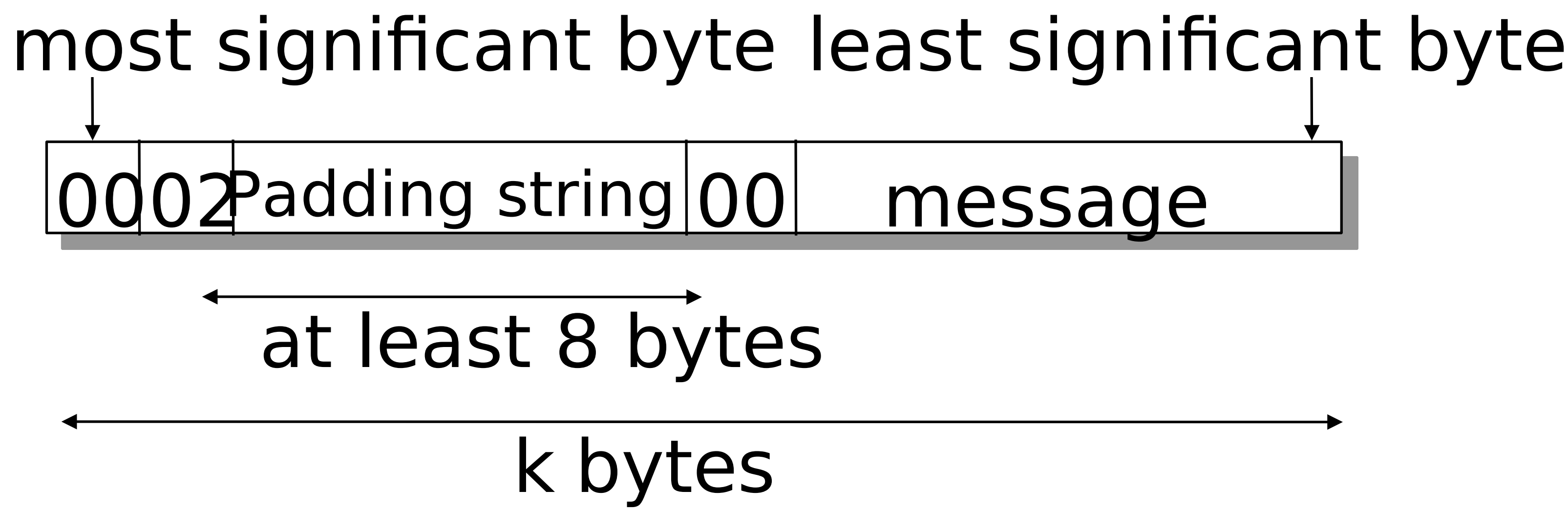
- There's a history here too!
  - PGP, PEM and MOSS
  - RSADSI's PKCS#7 based proposal
  - DKIM is a different development in this space
- Cryptographic Message Syntax (CMS) is the basis for S/MIME and various other crypto applications using ASN.1
- So S/MIME = CMS + Message-Specification + Certificate-specification

# CMS

- How to MAC, sign and/or encrypt application data in an ASN.1 oriented way
  - E.g. CMS defines: SignedData and EnvelopedData
  - XML and JSON equivalents started from here
- Algorithms and options same as X.509
- Latest CMS specification: RFC 5652
- But first, to encrypt, we need a bit of glue between arithmetic (for RSA) and ASN.1 BIT STRINGs – often uses PKCS#1 v1.5 format...

# PKCS#1 v1.5 Padding
# RFC 8017

RSA modulus: n=pq of length k bytes;
i.e. 256 ** (k-1) < n < 256 ** k



# CMS SignedData (1)

```
SignedData ::= SEQUENCE {
 version CMSVersion,
 digestAlgorithms DigestAlgorithmIdentifiers,
 encapContentInfo EncapsulatedContentInfo,
 certificates [0] IMPLICIT CertificateSet
                         OPTIONAL,
 crls [1] IMPLICIT CertificateRevocationLists
                         OPTIONAL,
 signerInfos SignerInfos }
```

# CMS SignedData (2)

```
SignerInfo ::= SEQUENCE {
 version CMSVersion,
 sid SignerIdentifier,
 digestAlgorithm DigestAlgorithmIdentifier,
 signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
 signatureAlgorithm SignatureAlgorithmIdentifier,
 signature SignatureValue,
 unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL }

SignerIdentifier ::= CHOICE {
 issuerAndSerialNumber IssuerAndSerialNumber,
 subjectKeyIdentifier [0] SubjectKeyIdentifier }

SignedAttributes ::= SET SIZE (1..MAX) OF Attribute
UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute
Attribute ::= SEQUENCE {
 attrType OBJECT IDENTIFIER,
 attrValues SET OF AttributeValue }
AttributeValue ::= ANY
SignatureValue ::= OCTET STRING
```

# CMS Message Specification

- Latest specification is RFC 8551
- Tells you how to:
  - Start with a MIME email message
  - Treat that as like plaintext the CMS way
  - Then take the resulting bytes and make them into a MIME message
- Note: LARGE messages exist
  - Have to handle BER as well as DER

# CMS Certificate Specification

- Latest specification is RFC 8550
- Tells you how to interface an s/mime mail user agent with a PKI
- Tells you how to interpret RFC 5280 for secure mail purposes
  - E.g. How to include email addresses in certificates

# Pretty Good Privacy (PGP)

- PGP can do all that S/MIME does
- PGPMime is RFC 3156
- PGP's basic formats in RFC 4880
  - Not ASN.1 based (TLVs)
- Web-of-trust model != X.509 PKI
  - But you don't have to
- Most important use-case: package signing

# PGP Key Management

- X.509-based PKIs are hierarchies
- PGP web of trust (WoT) based on user’s signing one another’s keys, with possibly many signatures per public key
- PGP Key IDs are hashes
- PGP key servers exist
- Usability: sucks:-(
- Details: lots of HOWTOs on the web

# S/MIME and PGP Deployment

- Most MUAs suport s/mime or PGP either built-in or as an option
  - There are also “plug-in” products
- And mostly then *can* work together
  - I’ve used both, PGP more usable (via Thunderbird/Enigmail)
- But secure mail is not ubiquitous
  - Why?

# e2e email security barriers

- Designs pre-date web user agent which changes trust model (where's the private key kept? Needs new infrastructure)
- Needs all major email service providers (yahoo, hotmail, gmail) to deploy the same thing which also needs to be implemented by all major user agent developers (microsoft, mozilla, apple, google)
- Public key retrieval needs to be fixed (doable if the above done, but a killer if not done), likely with some new PKI (doable but who's gonna pay?)
- Mail headers need to be protected as users don't get that S/MIME and PGP only protect body and not e.g. Subject, From (new enveloping protocol needed, can be done but kludgy)
- We need to unify S/MIME and PGP or pick one or we'll lose interop (it's ok if the other soldiers on for some niches)
- Users don't care much, so it has to be entirely transparent for them (needs significant UI work, co-ordinated across MUAs and significant web-UAs)

# e2e email current attempts

- (At least) two current projects have are trying to address general e2e mail security:
  - Autocrypt: https://autocrypt.org/
  - p≡p : https://pep.foundation/
  - Some niche service providers try make e.g. PGP easier:
  - ProtonMail: https://protonmail.com/
- All worthy, none (yet) with real MUA/mail-mega-provider traction
- Most email security today depends on TLS for mail transport security which is hop-by-hop and not end-to-end
- e2e email security doesn’t play so well with server-side anti-spam/malware/phishing techniques
- BUT, without e2e email security, it’s all postcards!
  - And there are people reading those: http://www.spiegel.de/international/europe/gchq-monitors-hotel-reservations-to-track-diplomats-a-933914.html

# Transport Layer Security (TLS)

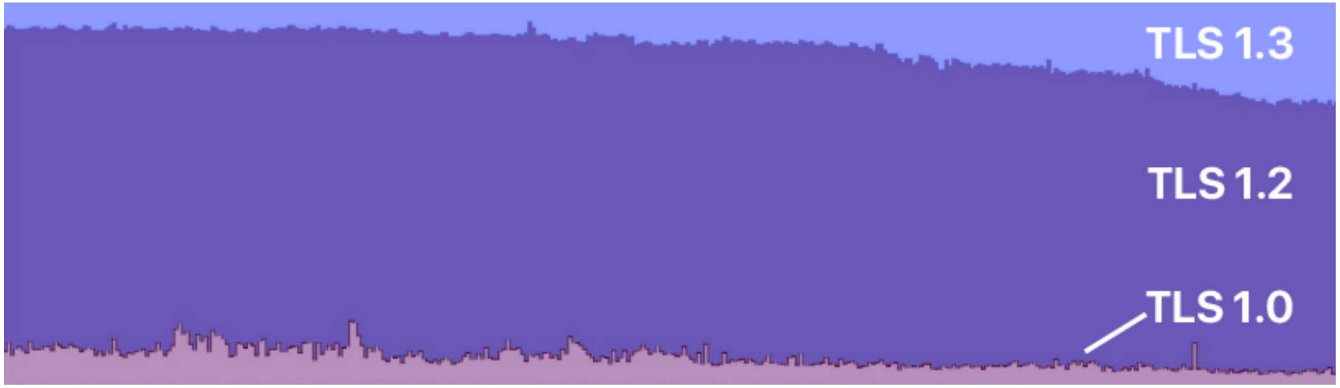
- Secure Sockets Layer (SSL): General Purpose Network Security Protocol proposed by Netscape in 1994.
- Designed to work with any application that uses sockets to communicate e.g., ftp, http, nntp, telnet...
  - Platform independent, application independent negotiations
- SSL standardised as Transport Layer Security (TLS)
  - Latest spec: TLS1.3, RFC 8446
  - Widely deployed: TLS1.2, RFC 5246
  - Oddly: RFC6101 is SSL3.0!

# SSL original services

- Server Authentication - Buyer can believe they are dealing with a bona fide merchant
  - (Optional) Client authentication
  - Digital certificates (X.509)
- Message Encryption - Buyer can send credit card details across the network without fear of interception
  - Also message integrity and freshness
- Relatively transparent to the user and application developer

# TLSv1.2 and TLSv1.3

- We'll look first at TLSv1.2, then lots more on TLSv1.3
- Note that TLSv1.3 is a major update despite the minor version number change
- Older versions: next slide and RFC8996
- Deployment snapshots:
  - https://www.ssllabs.com/ssl-pulse/ ~43% v1.3 Feb 2021
  - https://ietf.org/blog/tls13-adoption/ ~27-30% v1.3 Nov 2019




Cloudflare measurement from May 2018-May 2019

From: draft-ietf-tls-oldversions-deprecate-12 Best Current Practice
Errata exist

Internet Engineering Task Force (IETF)
Request for Comments: 8996
BCP: 195
Obsoletes: 5469, 7507
Updates: 3261, 3329, 3436, 3470, 3501, 3552,
3568, 3656, 3749, 3767, 3856, 3871,
3887, 3903, 3943, 3983, 4097, 4111,
4162, 4168, 4217, 4235, 4261, 4279,
4497, 4513, 4531, 4540, 4582, 4616,
4642, 4680, 4681, 4712, 4732, 4743,
4744, 4785, 4791, 4823, 4851, 4964,
4975, 4976, 4992, 5018, 5019, 5023,
5024, 5049, 5054, 5091, 5158, 5216,
5238, 5263, 5281, 5364, 5415, 5422,
5456, 5734, 5878, 5953, 6012, 6042,
6083, 6084, 6176, 6347, 6353, 6367,
6460, 6614, 6739, 6749, 6750, 7030,
7465, 7525, 7562, 7568, 8261, 8422
Category: Best Current Practice
ISSN: 2070-1721
March 2021

K. Moriarty
CIS
S. Farrell
Trinity College Dublin

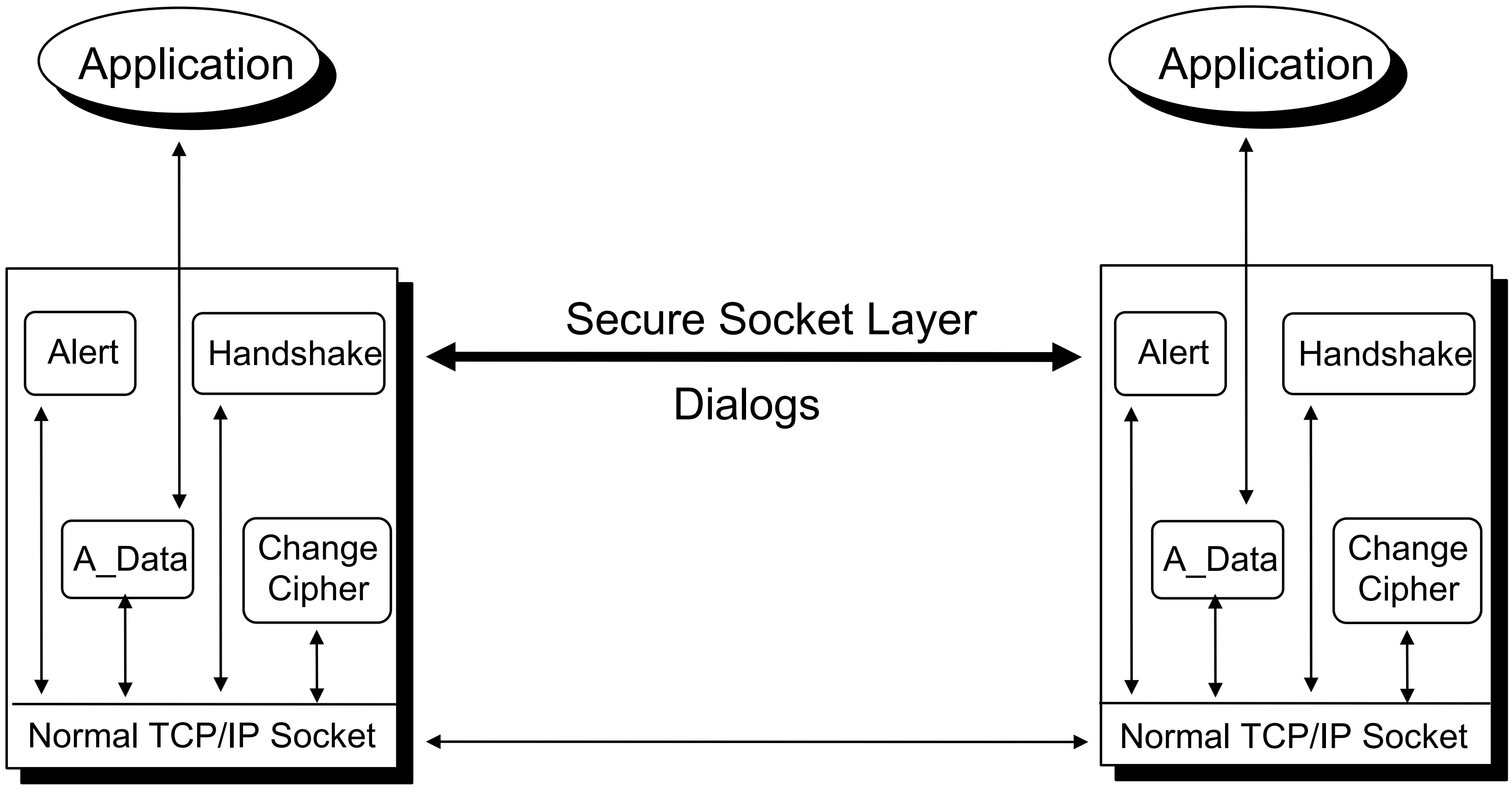
# Deprecating TLS 1.0 and TLS 1.1

## Abstract

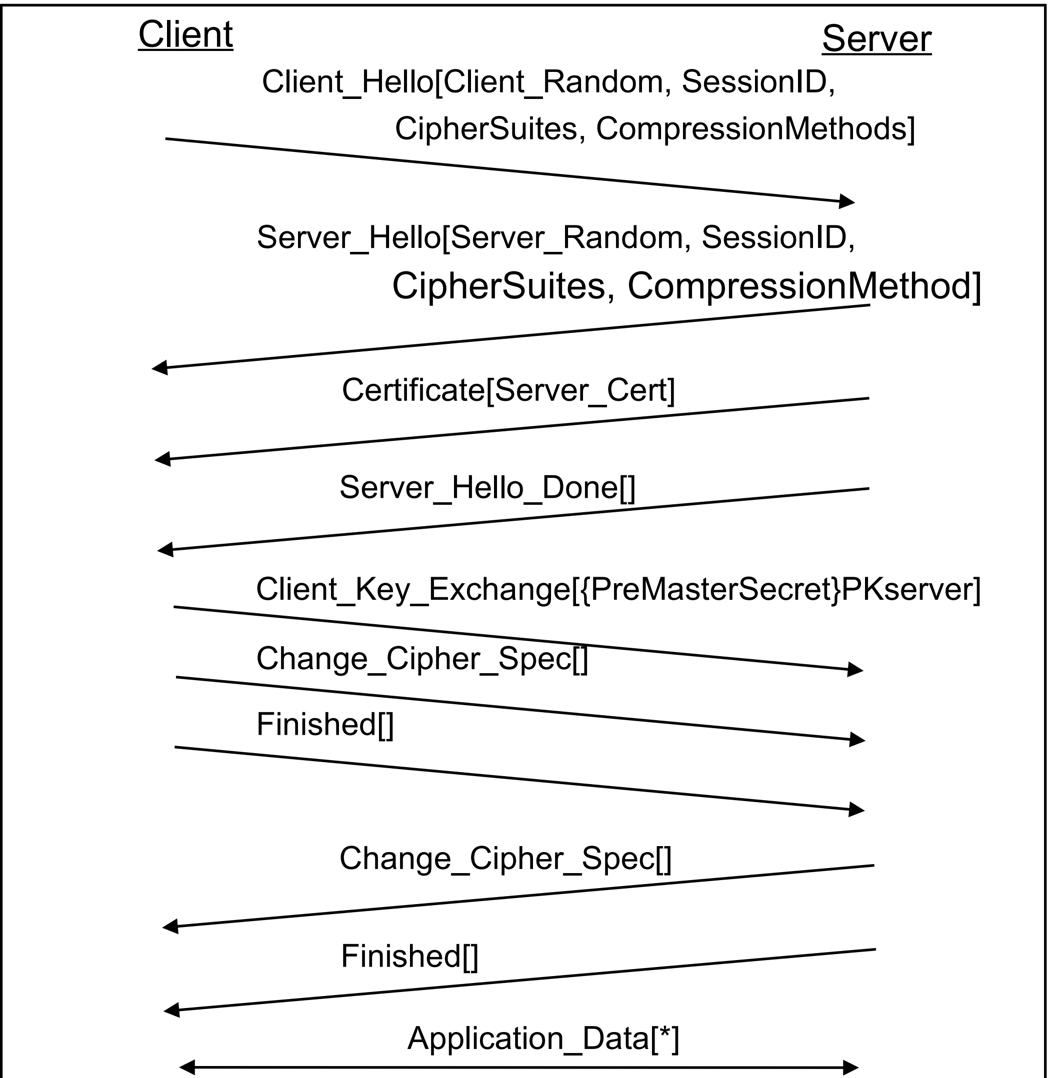
This document formally deprecates Transport Layer Security (TLS) versions 1.0 (RFC 2246) and 1.1 (RFC 4346). Accordingly, those documents have been moved to Historic status. These versions lack

# Components of the TLS Protocol

- TLS broken into 4 interrelated sub-protocols



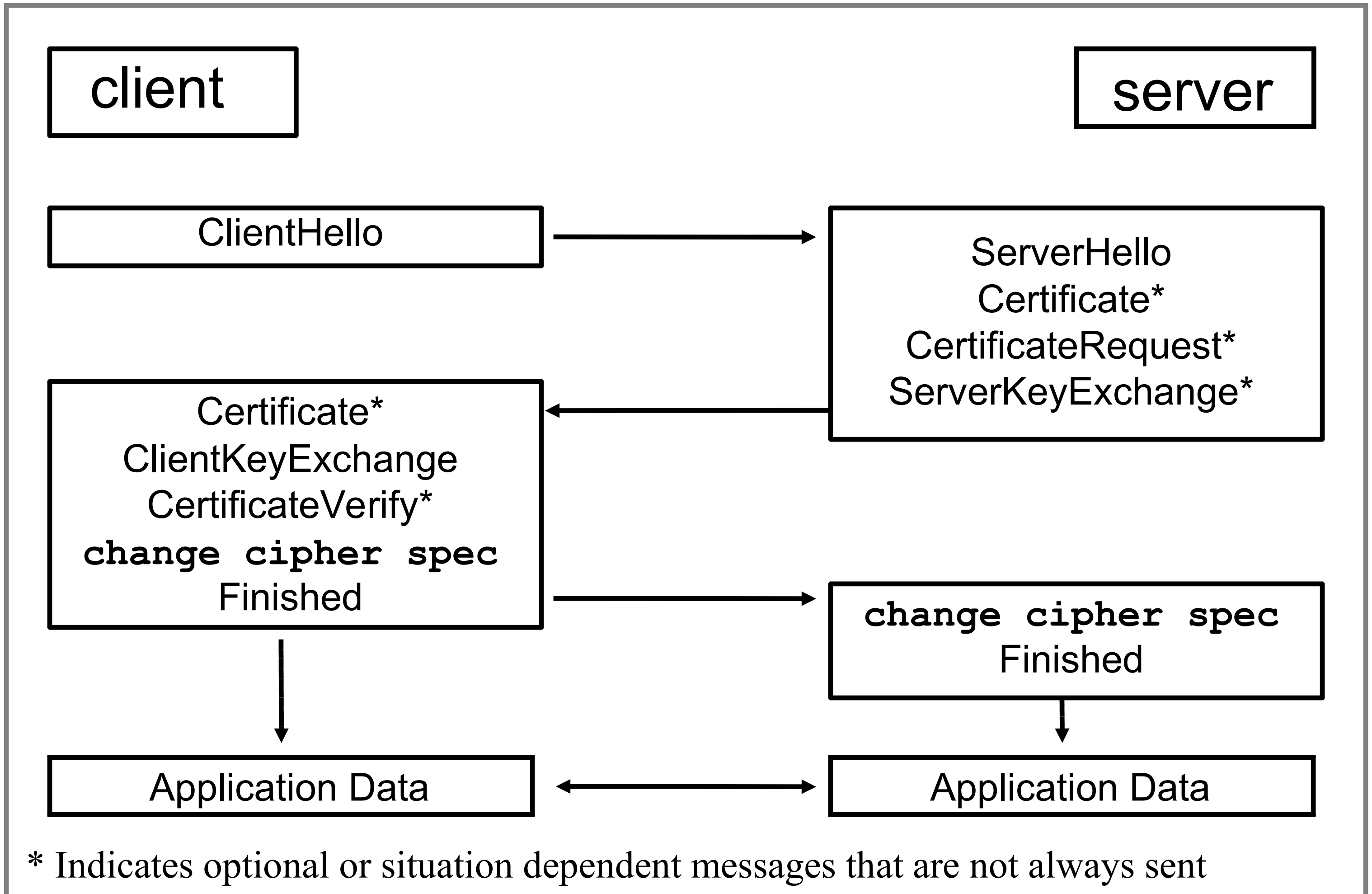
# The Handshake Protocol




- Negotiate Compression Method and Cipher Suite
- Swap random quantities
- Client obtains server certificate path
- Client invents PreMasterSecret (48 bytes) and securely sends it to the server
- Keys Calculated by both
- Finished message using new algorithms
- Data send in A_Data Units

# Handshake Protocol

## summary

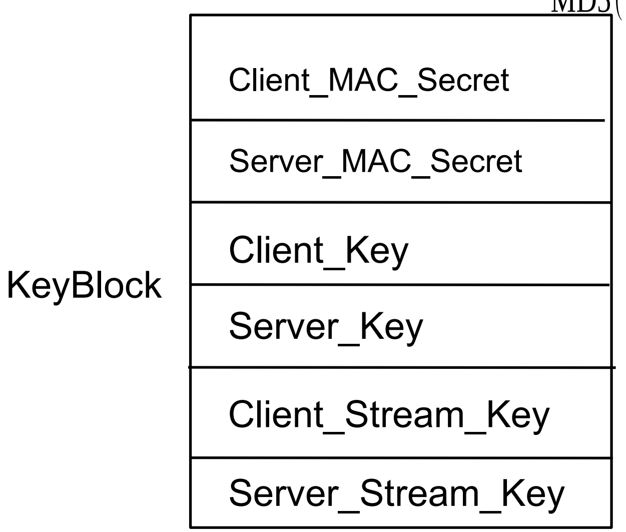


# Computing the Keys from the Pre-MasterSecret

- First compute MasterSecret
  - MasterSecret = f(PreMasterSecret, Client.Random, Server.Random)

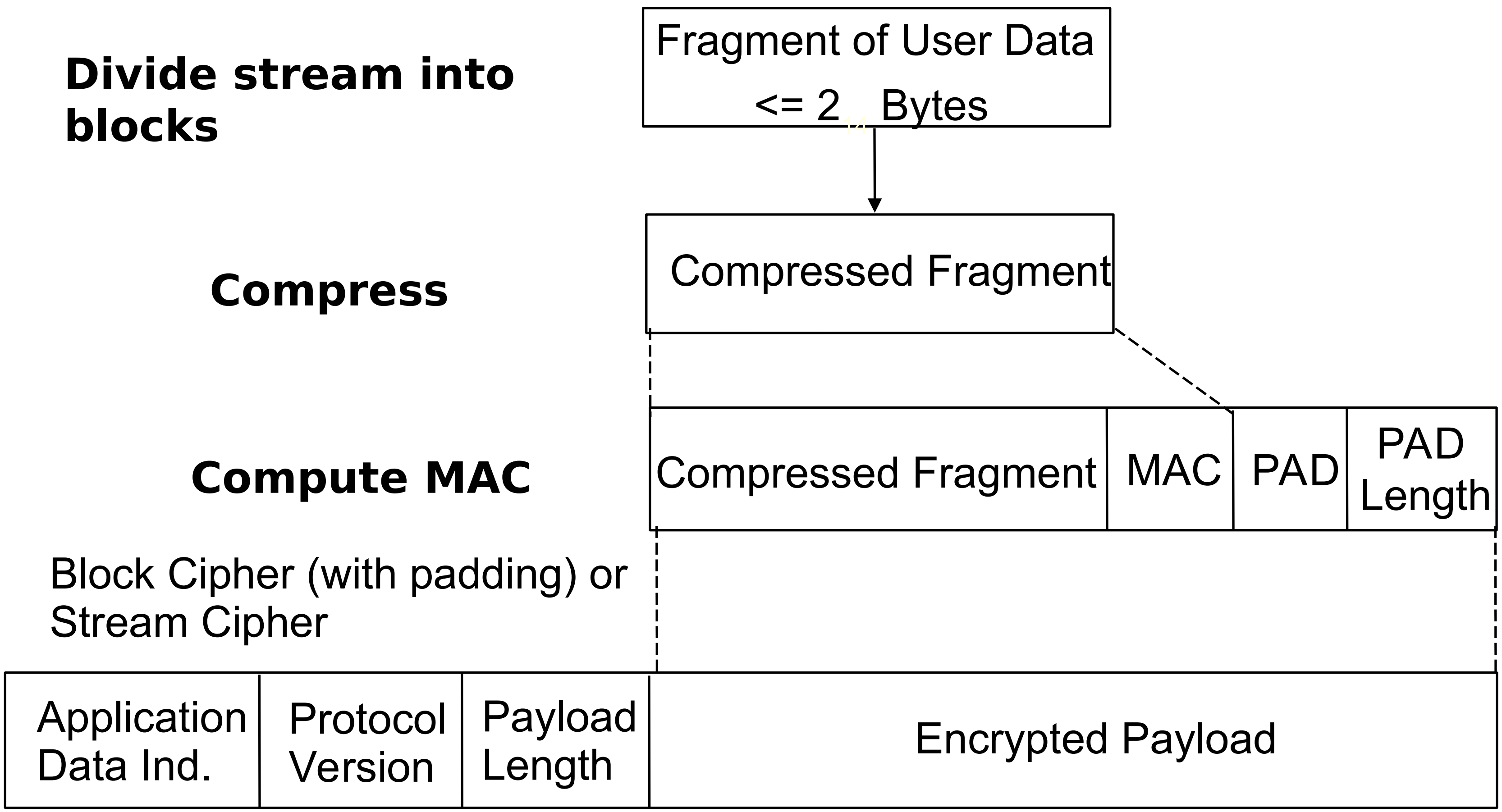
- MasterSecret Used to prime key-generator
  - KeyBlock = f(MasterSecret, Client.Random, Server.Random)

$$
\begin{aligned}
KeyBlock = {} & MD5(MasterSecret + SHA('A' + MasterSecret + ServerHello.Random + ClientHello.Random)) + \\
& MD5(MasterSecret + SHA('BB' + MasterSecret + ServerHello.Random + ClientHello.Random)) + \\
& MD5(MasterSecret + SHA('CCC' + MasterSecret + Server.Hello.Random + ClientHello.Random)) + [\ldots]
\end{aligned}
$$




Partition key-stream into individual quantities

# Applying the Keys to Application Data (Record Layer)



# Ciphersuites

- SSL/TLS supports various cryptographic options
  - Digest algorithms, key transport, …
- Design decision was to represent all choices made in a single value
  - Ciphersuite – a 16 bit number
  - TLS_RSA_WITH_3DES_EDE_CBC_SHA
- Interesting consequences…

# A TLS Handshake Visualisation

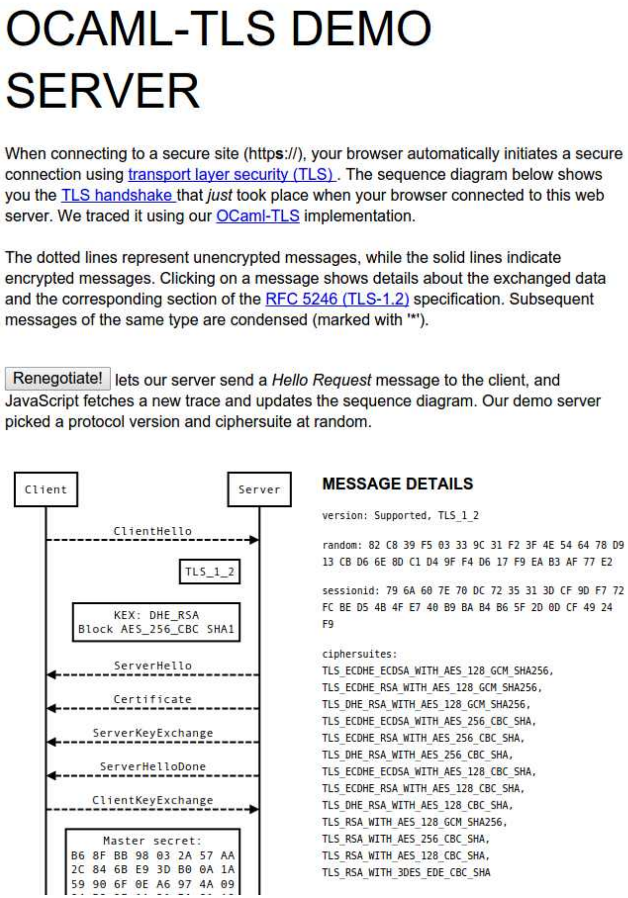
https://tls.nqsb.io/

Shows the messages you exchange with that server and references bits of text from RFC 5246.

If you allow JS, you can click the arrows.

Nice!

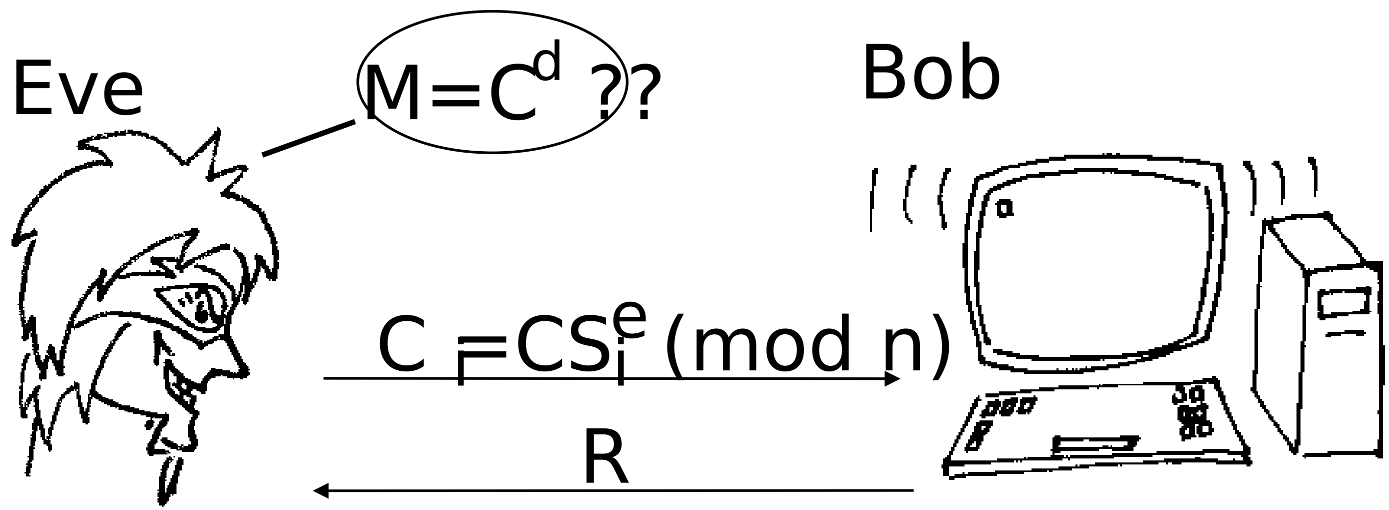
Note all the cruft though!



# Bleichenbacher

- “Materials” section has a full paper & ppt
  - If you **completely** understand the attack, you're doing very well!
- Basis
  - PKCS#1 v1.5 padding adds formatting to the data
  - If I have an “oracle” that'll attempt decryption and tell me when the recovered plaintext has the PKCS#1 v1.5 padding then I gain knowledge
- This happened with TLS!
  - “Million message” attack

# How the attack works: Overview




If a message $C_I$ is PKCS conforming then

$$2*256**(k-2) \quad -1 < MS < 3*256**(k-2)$$

- This is an adaptive chosen-ciphertext attack
- RFC3218 describes the attack and ways to avoid it.

# Deployment of TLS

- Mid '90's: Used in Netscape Commerce Server
  - International “Step-up” encryption: Strong crypto for international banks, with special Verisign certificate
  - Built into Communicator 4.0 and later
- Now: everywhere, standard part of Web browsers, servers and all development platofrms (PHP, python, golang etc.)
  - online trading, banking, commerce…
- “foo”/TLS/TCP on port 443 is becoming a de-facto baseline

# Recent(ish) TLS-relevant Stuff

- DANE RFC 6698
  TLSA RR: hashes of keys in DNS (4 options)
    - Requires DNSSEC
    - Provide additional assurance that the right key is being used for TLS, or avoids use of CAs
- Key-pinning in HTTP (HPKP): RFC 7469
- HTTP Strict Transport Security (HSTS): RFC 6797
- OSCP Stapling (RFCs 6066, 6961)
- MTA Strict Transport Security (RFC 8461)
- Encrypted Client Hello/SNI
    – https://tools.ietf.org/html/draft-ietf-tls-esni

# More Recent-ish TLS Stuff

- Datagram TLS (DTLS) for use with connectionless applications (e.g. RTP) is RFC 6347
- IETF TLS WG no longer in maintenance mode since 2014 when work on TLSv1.3 started
  - Might go back to maintainance mode in a bit
- Much more on all that shortly...

# IP Security (Ipsec)

- Took soooooo loooooong and produced such a complex outcome that they have a 2011 document roadmap (RFC 6071).
- Start with RFC 4301, the architecture
- Main moving parts: AH (RFC 4302), ESP (RFC 4303), and IKEv2 (RFC 7296)
    - AH mostly deprecated
    - IKE -> IKEv2 due to IKE/ISAKMP/... complexity, do NOT bother with IKEv1
- So, we'll only look at ESP & IKEv2

# Ipsec overview

- “Security Architecture” RFC 4301
  - Key exchange, data encryption and data integrity mechanisms at the IP layer
- Optional as part of an IPv4 stack
  - (Used to be) Mandatory to implement as part of IPv6 stacks
  - Note: Mandatory to implement (MTI) != “MUST use”
- Tunnel/Transport modes
  - VPNs using tunnel mode common
- Security policy DB
  - How to handle (un)protected packets

# ESP – Encapsulated Security Payload

```
0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ----
|               Security Parameters Index (SPI)                 | ^Auth.
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|                      Sequence Number                          | |erage
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ | ----
|                    Payload Data* (variable)                   | |   ^
~                                                               ~ |   |
|                                                               | |Conf.
+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|               |     Padding (0-255 bytes)                     | |erage*
+-+-+-+-+-+-+-+-+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |   |
|                               |  Pad Length   | Next Header   | v   v
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ------
|                 Authentication Data (variable)                |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

# ESP terms

- Specification: RFC 4303
- Security Parameters Index (SPI) selects amongst different
  - Algorithms; Keys; etc.
  - SPI (+ dest. IP) = Security Association (SA)
- Seq# for anti-replay
  - Never cycles (new SA needed)
- Next header specifies payload protocol
- SA's are directional
  - So we use different keys between Alice and Bob as compared to between Bob and Alice

# ESP – some features

- ESP allows some basic level of traffic hiding
  - Padding (up to 255 bytes) to disguise data length
  - Tunnel mode between gateways
- Data expansion
  - About +19 bytes per packet, possibly worse
  - Can be serious (telnet)

# IKE - Internet Key Exchange

- IKEv1 (RFC 2409) is a mess of:
  - ISAKMP (rfc2408)
  - OAKLEY (rfc2412) and SKEME (not an rfc)
  - DOI (rfc2407)
- IKEv2 (RFC 7296) is a single document that's much better
  - But is 142 pages long;-(
- Wasn't that simple!

# IKEv2 (1)

- Provides protocol
  - Mutually authenticate
  - Exchange keys
- Based on:
  - D-H and
  - Pre-shared secrets or RSA

# IKEv2 (2)

- Establishes an IKE-SA and one (or more) CHILD-SA
- Generally requires 4  or 6 messages
  - IKE_SA_INIT (req/rep)
  - IKE_AUTH (req/rep)
  - CREATE_CHILD_SA (req/rep)

# IKEv2 Phase1
# IKE_SA_INIT & IKE_AUTH

```
Initiator                                  Responder
-----------                                ---------
HDR, SAi1, KEi, Ni   -->

          <--    HDR, SAr1, KEr, Nr, [CERTREQ]

HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,]
 AUTH, SAi2, TSi, TSr}     -->

          <--    HDR, SK {IDr, [CERT,] AUTH,
                              SAr2, TSi, TSr}
```

# IKEv2 Phase2 CREATE_CHILD_SA

```
Initiator                                  Responder
-----------                                ---------
HDR, SK {SA, Ni,[Kei],[TSi,Tsr]} -->

         <-- HDR, SK {SA, Nr,[Ker],[Tsi, TSr]}
```

# Some IKEv2 Features

- Not IKEv1!
- Uses ESP to protect things
- Misc:
  - PFS, NAT-capable, Traffic Selectors, Liveness checks, Cookies
- Still a bit clunky though!
  - E.g. Compression (rfc2393), ESP and then AH nested SAs

# IPsec summary

- Tunnel mode widely used for VPNs and works just fine
  - Transport mode hardly used at all
- IKE interop today isn't perfect (with certs), but is ok
  - Some vendor-specific stuff, e.g. For RADIUS/legacy auth
- Deployment issues:
  - Windows, NAT, Firewall, ECN, Opportunistic Keying, APIs
  - IKEv2 work aims to address a bunch (but not all!) of these

# Kerberos

- Originally developed as part of MIT project athena
  - RFC4120 (but RFC1510 may be easier)
- Designed for many users working with few(-ish) servers
  - Largely symmetric key (shared secret) based
- Based around clients interacting with a Key Distribution Centre (KDC) aka:
  - Authentication server (AS)
  - Ticket Granting Server (TGS)

# Kerberos

- Uses ASN.1 again! (sort-of)
- Latest spec: RFC 4120
- Typical use:
  - Client sends AS_REQ to KDC gets AS_REP containing TGT
  - Client sends TGS_REQ to KDC (includes TGT) and gets TGS_REP (including Ticket)
  - Client sends KRB_SAFE to server including Ticket

# A Ticket

```
Ticket ::= [APPLICATION 1] SEQUENCE {
tkt-vno [0] INTEGER (5),
realm [1] Realm,
sname [2] PrincipalName,
enc-part [3] EncryptedData -- EncTicketPart }

-- Encrypted part of ticket

EncTicketPart ::= [APPLICATION 3] SEQUENCE {
flags [0] TicketFlags,
key [1] EncryptionKey,
crealm [2] Realm,
cname [3] PrincipalName,
transited [4] TransitedEncoding,
authtime [5] KerberosTime,
starttime [6] KerberosTime OPTIONAL,
endtime [7] KerberosTime,
renew-till [8] KerberosTime OPTIONAL,
caddr [9] HostAddresses OPTIONAL,
authorization-data [10]
         AuthorizationData OPTIONAL }
```

# Kerberos Flows

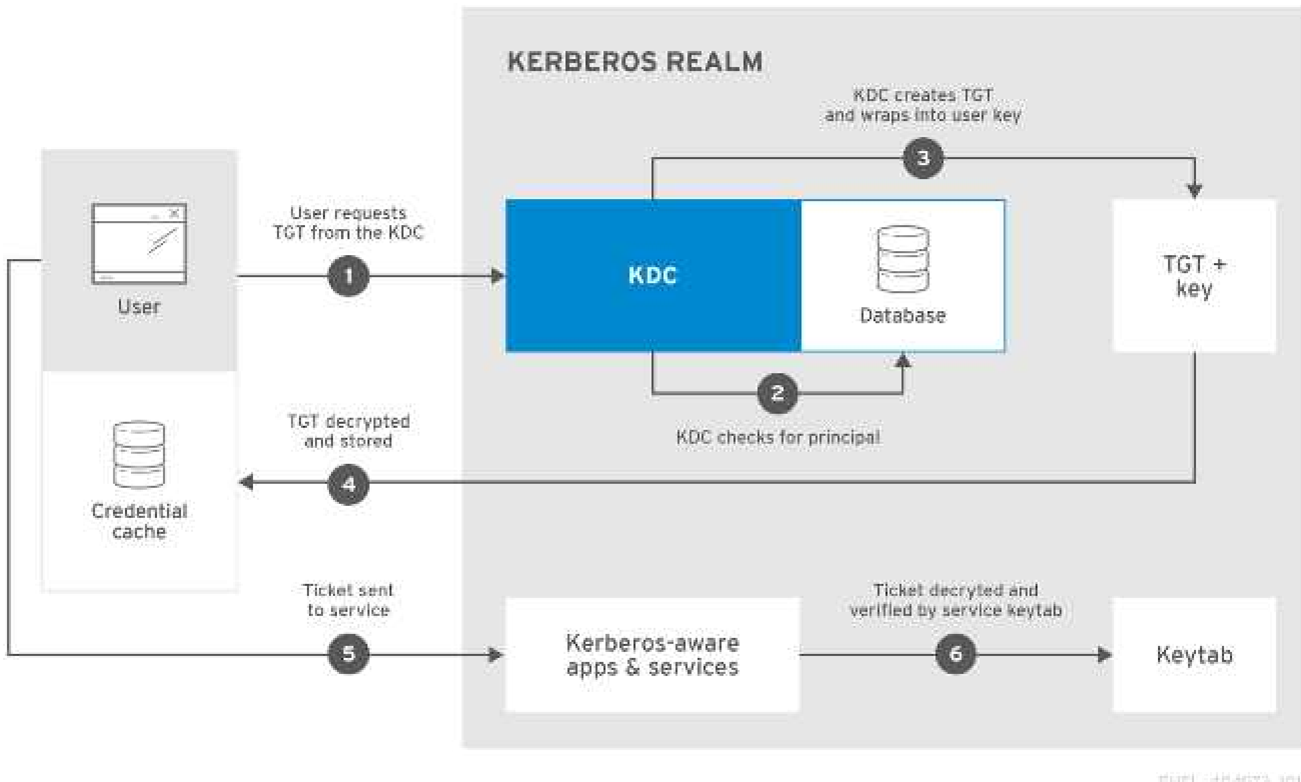



**Figure 11.1. Kerberos Authentication**

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system-level_authentication_guide/using_kerberos

# Kerberos Things to Note

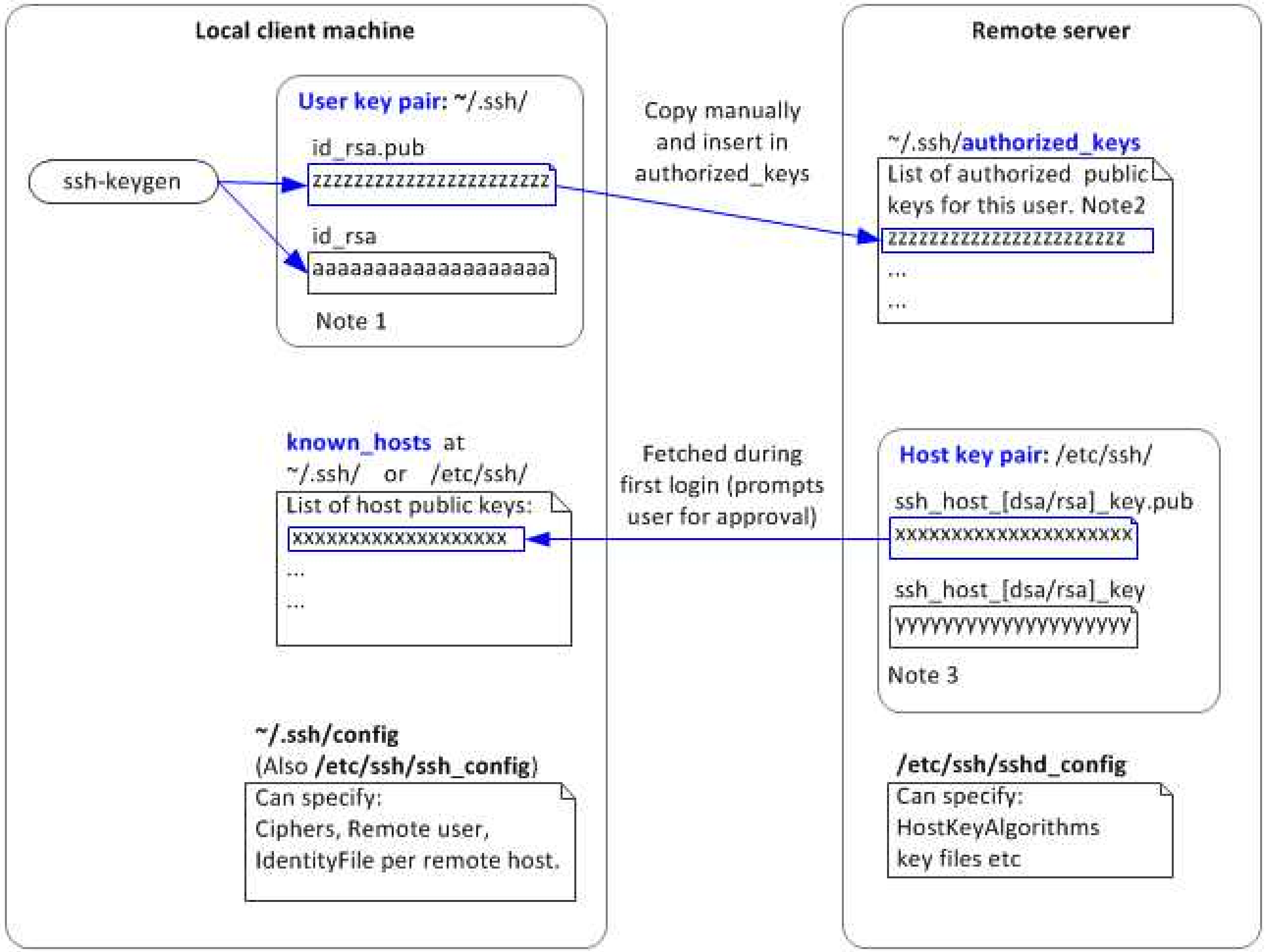
- Loose clock sync. Required
- Kerberos realm is like but not the same as a DNS domain
- Inter-realm and public key based operation defined but not used that often
- Used in Windows NT security and later (KDC is part of ActiveDirectory)
- Various authorization data extension have been done over the years
- AS_REQ can contain dictionary attackable password or something better (usually the latter in modern implementations)

# Secure Shell (SSH)

- Architecture: RFC 4251
- Details in RFCs 4252 (transport protocol), 4253 (user auth), 4254 (multiplexing)
- Updates in RFC 8308, 8388 and others
- Entirely typical flows – do DH, authenticate in encrypted tunnel, application data in channel in encrypted tunnel
- Tends to be more driven by widely-used implementations rather than the RFCs, so generally implementations are ahead of the RFCs, sometimes significantly so esp. for extensions and algorithms, e.g.: https://www.openssh.com/specs.html
- Next 2 slides from: https://serverfault.com/questions/935666/ssh-authentication-sequence-and-key-files-explain

## Files involved in SSH connection: preparation

gwideman 2018-10-14




**Notes:**

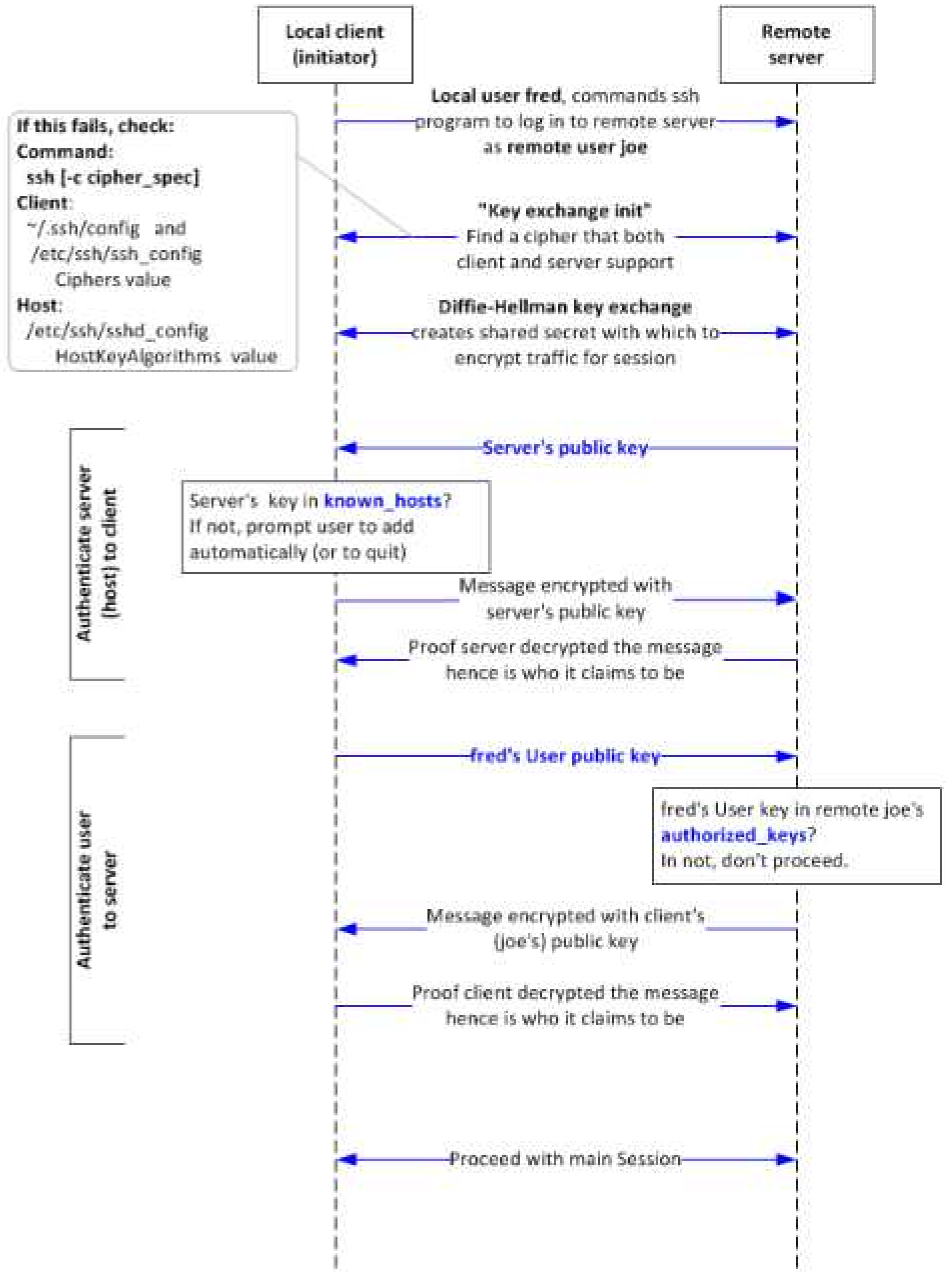
**1. A user can, and often will, have more than one user key pair, in files with distinctive names.** Commands that use key files allow specifying which files to use.

**2. User account on the remote host need not have the same username as that on the local initiating machine.** Ie: In local ssh or rsync command, specify the remote user account by which to log in.

**3. Host key pair is created** at previous installation time, such as when installing openssh.

Files involved in SSH connection: in use

gwideman 2018-10-14



# SSH Messages

Old style message descriptions, e.g. RFC 4252 describes the value of 'signature' as a signature by the corresponding private key over the following data, in the following order:

```
string    session identifier
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service name
string    "publickey"
boolean   TRUE
string    public key algorithm name
string    public key to be used for authentication
```

# SSH things to note

- SSH is the epitome of the Trust On First Use (TOFU) model in how it handles known host keys on the client
- Be careful of VMs – easy to clone host key accidentally, e.g. if using puppet or similar
- In almost all cases: turn off password authentication and only allow public key based auth for systems you manage
- Be careful of user key pairs – it's very easy to end up with lots of old keys lying about e.g. in authorized_keys files
- SSH tunnelling is a fine thing – use a jump-off host to which you have a login to connect to others inside the LAN of the jump-off host – but as usual be careful in allowing this if you're a sysadmin

# Wireguard-1

- Newish (2016), non-standard, work-in-progress VPN tunnelling protocol, but now (Jan 2020) accepted into upstream Linux kernel
- Protocol description: https://www.wireguard.com/papers/wireguard.pdf
- Design goals: simpler, quicker than IPsec or OpenVPN using only modern crypto, ~4K LOC in kernel
- `wg` tool acts like `iwconfig` for managing interfaces, e.g. `wg0`, use normal OS tools (e.g. `ip`) to create i/f and handle addresses and routes
- Local interfaces are associated with a static curve25519 key pair, remote peers with a public key, ephemeral curve25519 keys are generated in 1RTT handshake
- Has a cookie mechanism for DoS mitigation, triggered if responder is "under load"
- Optional PSK can be bound with key exchange for post-quantum future-proofing
- Traffic is protected using Chacha20poly1305 and encapsulated in UDP

# Wireguard-2

- Public key distribution is out of scope – requires some other tooling, same as SSH
- Wireguard doesn't allocate IP addresses for clients – that's also considered out of scope and needs other tooling (apparently, haven't tried) – claimed to be problematic for some VPN operators who don't want to configure/log anything per-client
  - https://git.zx2c4.com/wg-dynamic being developed, not sure of status
- Various timers and optional keepalives bullt-in (for NAT)
- If you want firewalling – just use a firewall (e.g. `ufw`) as if the traffic were in clear (no IPsec policy DB here:-)
- Protocol (designer) is v. opinionated – no crypto agility at all within protocol versions (that's also "modern":-)
- Performance and attack surface look good, will be interesting to see how this evolves

# Mix'n'match

- PKI vs. Shared-secret vs. Trusted public keys
- TLS vs IPsec vs S/MIME vs CMS vs PGP vs Kerberos vs SSH vs Wireguard
- When should you pick which?

# Recent hour(s)...

- PKI model and protocols
- SMIME formats and secure email
- **TLS protocol (TLS1.2 or TLS1.3)**
- IPsec
- Kerberos
- SSH
- Wireguard

Knowing **one** of these in detail is enough for exam purposes – you **will** need to read the source materials