(in)Secure Internet Communications

Trust issues on TLS, PKI, DNSSEC

(Alt title: 20 years of fail)

George Kargiotakis
@kargig
0x897C03177011E02C

black d@ck 06/06/2014

whois

$ id

uid=1000(kargig) gid=1000(sysadmin)
groups=1(HELLUG),2(HTFv6),3(Hackerspace.gr),4(DLN.gr),5(),6(),...

$ last kargig

GRNET – System Administration

Gennet – Linux-based broadband CPEs (IPv6 capable)

University of Ioannina – System Administration

$ apropos kargig

iloog – Greek gentoo-based livecd

GrRBL – Greek AntiSpam Blacklists

Greek AdBlock Plus filter – self-explanatory

Greek HTTPS Everywhere rules

Tormap

void.gr – My Blog

Secure digital communications

- Security in the digital world needs strong crypto

- Counterparties:
  - How to authenticate ? → PKI, DNSSEC
  - How to encrypt ? → TLS

- **TLS + PKI || TLS + DNSSEC (+PKI)**

Trust

Who do you actually trust when you visit an HTTPS site ?

"Security is as strong as the weakest link in the chain"

So who's the weakest link in our secure comms?

SSL/TLS

- Secure Sockets Layer / Transport Layer Security

  designed to provide security over the Internet

  - Use of X.509 certificates (public key crypto) to **authenticate** counterparty
  - Exchange a symmetric session key which encrypts data
    - Data **Confidentiality**
    - Message Authentication Codes (**Integrity**)

- Client indicates need of TLS connection
  - Different service port (443, 465, 993, etc)
  - Same service port using STARTTLS

TLS Handshake

1. ClientHello (version, ciphers, etc)

2. ServerHello (version, ciphers, certificate)

3. Client checks Server certificate

4. Client creates a secret pre-master key and encrypts it with server's public key, then sends the key to the server

5. Server decrypts pre-master key (using private key of certificate) and generates a master key. Client performs the same steps to generate a master key.

6. Client and server use master key to generate (symmetric) session-keys.

7. Client informs server of using session key. Client-side handshake ends.

8. Server informs client of using session key. Server-side handshake ends.

9. Start exchanging data!

TLS History

SNP (Secure Network Programming API) ~ 1993

SSL 1.0 ~ 1994

SSL 2.0 ~ 02/1995 (Hickman, Kipp, "The SSL Protocol", Netscape)

SSL 3.0 ~ 11/1996 (draft-ietf-tls-ssl-version3-00)

TLS 1.0 ~ 01/1999 (RFC2246)

(aaaand 7 $%@$# years later...)

TLS 1.1 ~ 04/2006 (RFC4346)

TLS Extensions ~ 04/2006 (RFC4366)

TLS Handshake Message for Supplemental Data ~ 09/2006 (RFC4680)

TLS 1.2 ~ 08/2008 (RFC5246)

TLS Renegotiation Indication Extension ~ 02/2010 (RFC5746)

TLS Authorization Extensions ~ 05/2010 (RFC5878)

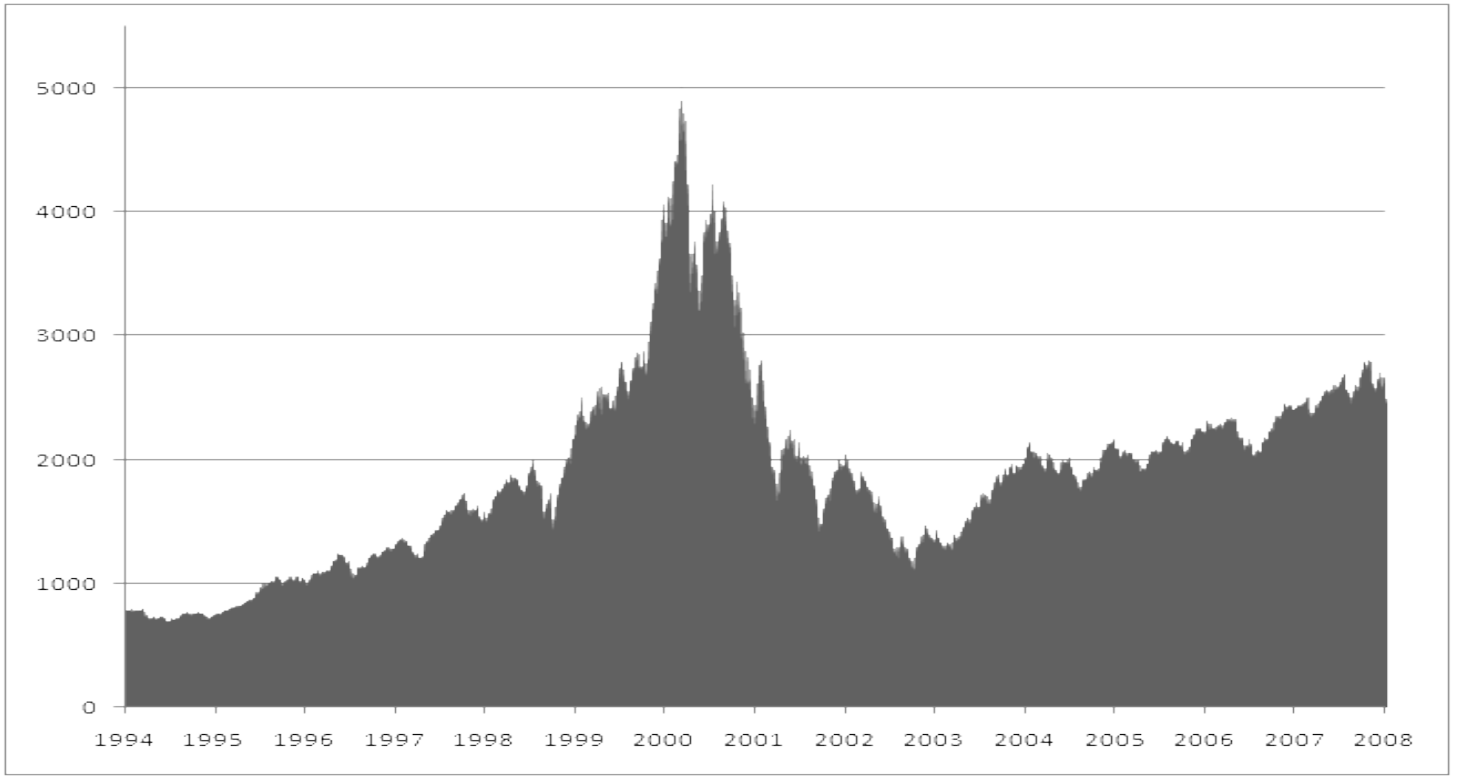
TLS Extensions: Extension Definition ~ 01/2011 (RFC6066)

Using OpenPGP keys for TLS Authentication ~ 02/2011 (RFC6091)

TLS Application Layer Protocol Negotiation (ALPN,ex NPN) ~ 03/2014 (draft-ietf-tls-applayerprotoneg-05)

TLS History (extra)

So what happened from TLS1.0 to TLS1.1 ?



SSL/TLS version diff

- SSL 2.0 **BAAAAD**

  Prohibit use of SSL 2.0 ~ 03/2011 (RFC6176)
  - **MAC uses MD5**
  - **MITM during handshake**
  - **Message Integrity + Encryption uses the same key**
  - **No separate session end message (MITM sends a TCP FIN and session gets dropped)**

- SSL 3.0 vs SSL 2.0
  - Different handshake flows (SSL 2.0 has no pre-master key)
  - SSL 2.0 client picks the cipher vs SSL 3.0 server picks cipher
  - SSL 3.0 uses BSAFE 3.0 from RSA Data Security -> SHA-1
  - Protects against handshake MITM
  - No single certificate name (useful for vhosts)

- TLS 1.0 vs SSL 3.0
  - Key-Hashing for Message Authentication (HMAC vs MAC)
  - Enhanced Pseudorandom Function (PRF) uses XORed (MD5+SHA1)
  - Improved finished message verification (prf+hmac values)
  - Consistent certificate handling
  - Better alert messages
  - TLS 1.0 connections can be downgraded to SSL 3.0 (interoperability vs security)

SSL/TLS version diff #2

- TLS 1.1 vs TLS 1.0
  - Protection against Cipher Block Chaining attacks
    - implicit vs explicit IV
    - handling of padding errors

- TLS 1.2 vs TLS 1.1
  - Replace MD5/SHA-1 combination in PRF with SHA-256 + cipher-suite-specified PRFs
  - Cleanup to the client's and server's ability to specify which hash + signature algorithms they will accept
  - Addition of support for authenticated encryption (GCM) with additional data modes
  - TLS Extensions definition and AES Cipher Suites were merged in
  - Tightened requirements
  - Extensive alerts
  - Mandatory TLS_RSA_WITH_AES_128_CBC_SHA cipher suite
  - Added HMAC-SHA256 cipher suites
  - Removed IDEA and DES cipher suites - they are now deprecated

The quest for Forward Secrecy

- Keep session keys secure in the case of a compromised private key
  - W/O FS if private key is compromised all future and past stored communications can be decrypted!
- Needs **DHE** (Diffie-Hellman Ephemeral) key exchange (TLS 1.0+)
- **DOES NOT protect against cryptanalysis of ciphers!**
- May 2014 ~ 9% of TLS-enabled websites provide proper FS (according to SSL Pulse)

CAUTION!
TLS Session tickets always stored with AES128-CBC-SHA256
if that is compromised say bye bye to FS!

TLS Present (extra)

TLS 1.2 in Browsers

- Google Chrome 30+
- Mozilla Firefox 27+
- Internet Explorer 11+
- Opera 17+
- Safari (iOS) 5+
- Safari (OS X) 7+

Popular Implementations

- OpenSSL (OpenSSL Project)
- GnuTLS (GnuTLS Project)
- NSS (Mozilla)
- Secure Transport (Apple)
- Secure Channel/Security Support Provider Interface (Microsoft)
- LibreSSL (not yet!)

TLS Issues

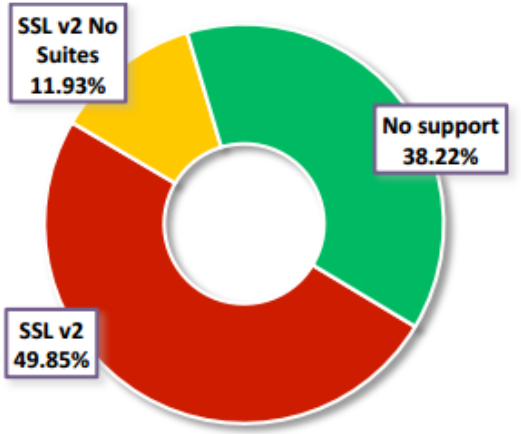
Implementation (mostly) failures

- Predictable Netscape Seed (1994)
  seed = time of day + process ID + parent ID
- OpenSSL RSA keys timing attack (2003)
- Debian + OpenSSL md_rand (2008)
- * Certificates with MD5 signatures (2008)
- Certificate Null Byte Poisoning (2009)
- OpenSSL OCSP Stapling (2011)
- * DUAL EC DRBG (backdoor?) (2013)
- Java SecureRandom (2013)
- Secure Transport "goto fail" (2014)
- Ruby insecure defaults (SSL2.0) (2014)
- GnuTLS "goto cleanup" (2014)
- OpenSSL Heartbleed (2014)
- ChangeCipherSpec Attack (**05/06/2014**)

(more) Protocol failures

- Injection during renegotiation attack (RFC5746)
- Downgrade attacks (to use weaker keys)
- BEAST (2011) (found 2002 by Rogaway)
  - CBC attack
  - mitigation: use a stream cipher -> RC4 or use TLS 1.1+
- CRIME (2012) (found 2002 by Kelsey)
  - data compression attack
  - mitigation: disable TLS compression
- Padding Oracle - Lucky 13 (2013) (originally found in 2002 by Vaudenay)
- Truncation attacks (2013)
  - attacker injects TCP FIN to the server on client logout

TLS Usage (2010)



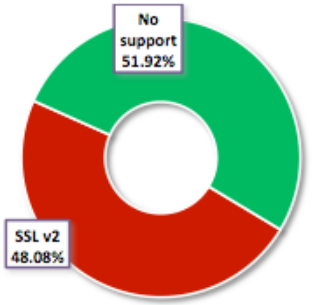
Protocol	Support	Best protocol
SSL v2.0	302,886	-
SSL v3.0	607,249	3,249
TLS v1.0	604,242	603,404
TLS v1.1	838	827
TLS v1.2	11	11

TLS Usage (2011)

Protocol Support

Half of all trusted servers support the insecure SSL v2 protocol

- Modern browsers won't use it, but wide support for SSL v2 demonstrates how we neglect to give any attention to SSL configuration
- Virtually all servers support SSLv3 and TLS v1.0
- Virtually no support for TLS v1.1 (released in 2006) or TLS v1.2 (released in 2008)

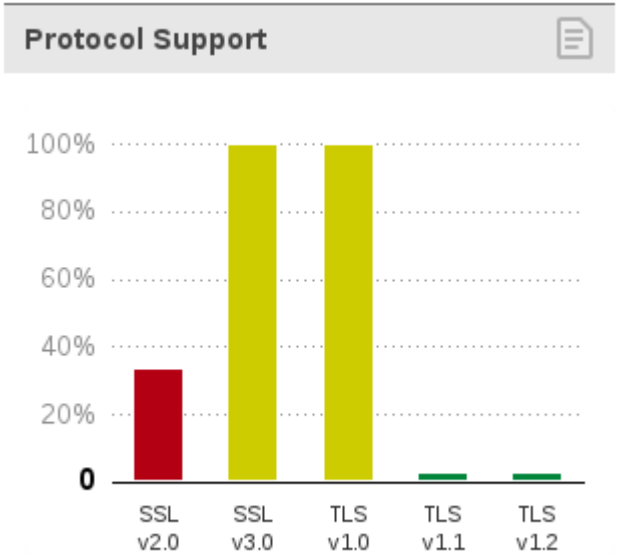


Protocol	Support	Best protocol
SSL v2.0	143,591	110
SSL v3.0	298,078	5,205
TLS v1.0	293,286	292,366
TLS v1.1	916	854
TLS v1.2	69	69

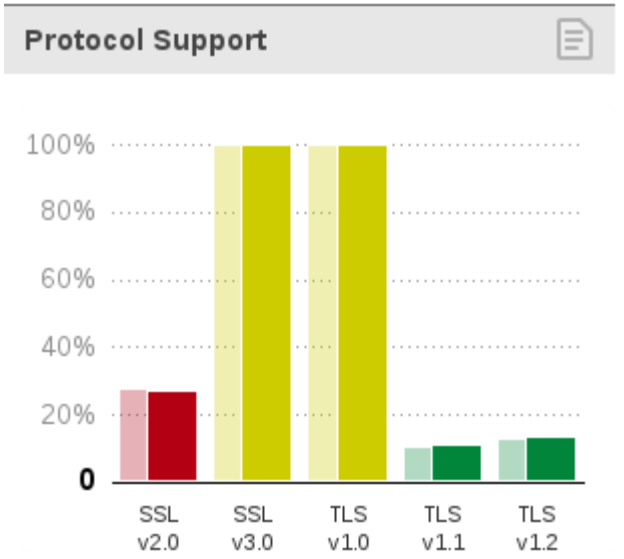
QUALYS

TLS Usage

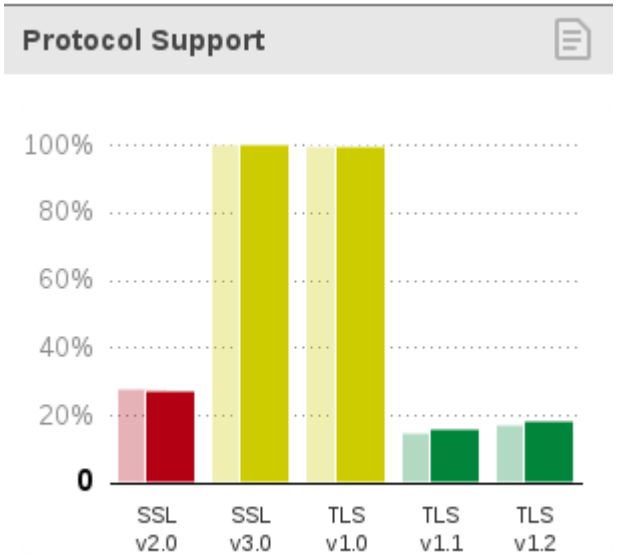
04/2012



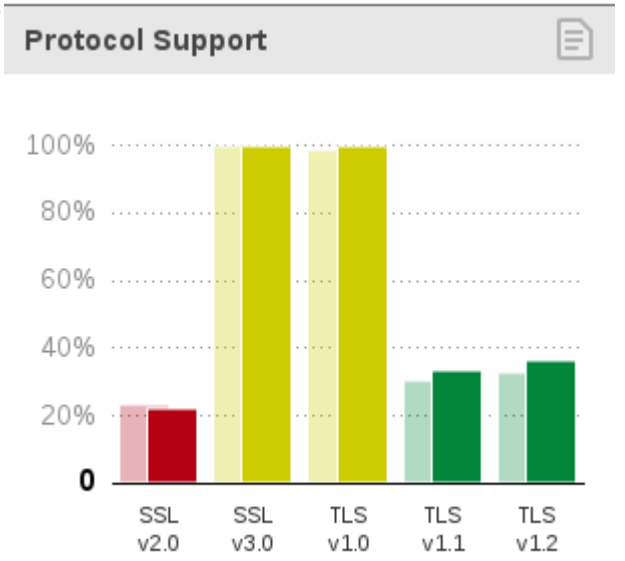
04/2013



09/2013



05/2014



Security Engineering

- Cryptographic research ~ 10 years ahead of security engineering
  - MD5 was known to be vulnerable (collisions) in the 90s, yet wasn't replaced with SHA-1 until mid-2000
  - Side channel attacks (BEAST, CRIME) known since mid-2000, yet fixed in 2010+
- **Nothing gets re-engineered until a *practical* attack is made PUBLIC!**

...or people get worried (Snowden)

PKI

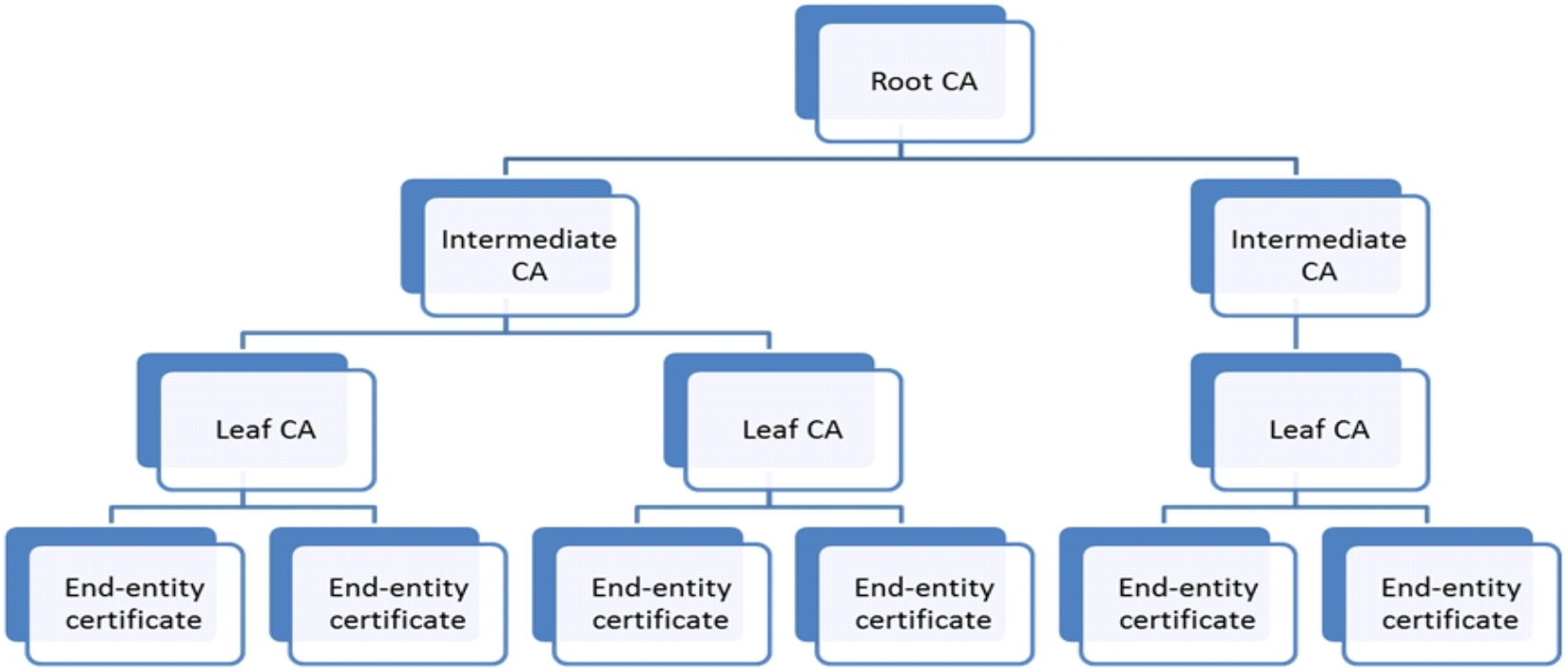
- X.509 (RFC5280 & RFC6818)

  ITU-T standard for PKI
  - public key certificates
  - certificate revocation lists
  - authorization certificates
  - certificate path validation algorithm
- part of X.500 standard (Electronic Directory Services) from the 80s...
- Strict hierarchical system (vs OpenPGP "web of trust")
- Certificate = signed(Identity + public key)

CAs

- CA validates (Identity) + signs certificate
- Public parts of CAs root certificates in our browsers → signature validation
- SubCAs
  - CA's private key is **TOO** valuable
  - CAs create subCAs (intermediate certificate authorities)
- Certificate chain: CA → subCA X → subCA Y → Certificate
- Validity
  - CA certificate: ~30 years
  - SubCA certificate: ~10-15 years
  - Certificate: 1-3 years

PKI Hierarchy

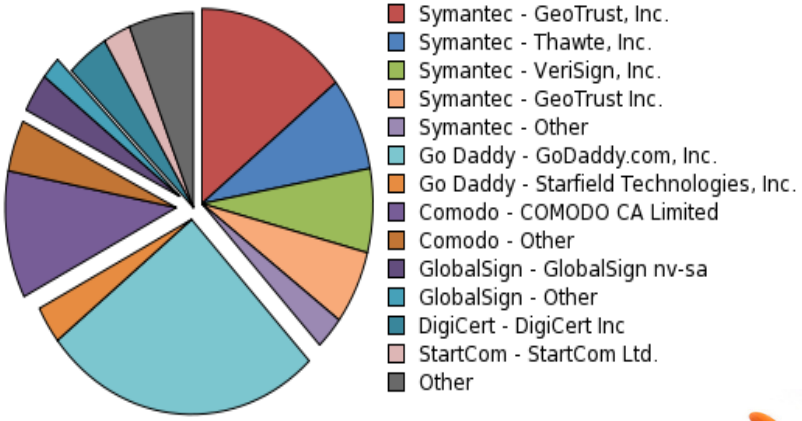


CA history

- There used to be only one CA... **VeriSign**
  - And it was a spin-off of RSA Security (1995)
- Then **Thawte** was created (1995)
- Then came GlobalSign (1996)
- Then came Comodo (1998)
- Then **VeriSign bought Thawte** (1999)
- Then came GeoTrust
- Then **VeriSign also bought Geotrust** (09/2006).
- Then **Symantec bought VeriSign** (2010)
  - VeriSign had already signed >3.000.000 certificates

Market share in 12/2013 (*)

Symantec: 38.1%
Comodo: 27.5%
GoDaddy: 14.3%
GlobalSign: 10.5%
StartCom: 2.7%



(*) Market share percentages is by w3techs and graph by netcraft (both for 2013 by they differ regarding GoDaddy vs Comodo size

http://w3techs.com/technologies/overview/ssl_certificate/all

http://www.netcraft.com/internet-data-mining/ssl-survey/

CA Usage

EFF's SSL Observatory results

- How many CAs do we **trust** ?

  >100 Trust Roots (~60 Organizations)

  1482 Trustable CAs by MS or Mozilla

  651 Different Organizations with trustable CAs

- >50 Countries have their own CAs

  among them: CN, IR, TR, IN, etc

  in GR we have HARICA

  (certificate name constraints: .gr, .eu, .edu, .org)

subCAs

What can you do with a subCA ?

- Be a CA!
- Generate a valid certificate for **any** domain

Who has subCAs ?

- DHS (safe!)
- Etisalat (who wouldn't trust them?)
- Vodafone (remember GR spying?)
- Booz Allen Hamilton (did I say spying ?)
- Marks and Spencer (!??! lol why?)

  We can't know!

How many subCAs per CA ? (SSL observatory)

- DT: 252
- CyberTrust: 93
- AddTrust: 72
- GlobalSign:63

SubCA + DPI = L.F.E. <3

Famous CA Failures

- CAs have signed certificates for 'localhost' (>6000 valid localhost certifications!)
- VeriSign gave a Code Signing Certificate to an individual who claimed he was from Microsoft (2001)
- VeriSign was repeatedly hacked in 2010 but they revealed it in 02/2012!
- "RSA Security 1024 V3" CA certificate (2010)
- Comodo Hacks (2011) by ComodoHacker
- DigiNotar Hacks (2011) by ComodoHacker
- GlobalSign web server was also hacked by ComodoHacker (2011)
- Trustwave gets caught selling subCA to a DLP solution (2012)
- TURKTRUST subCA signs *.google.com certificate (2013)
  - subCA was placed inside a DPI
  - Google catches them red handed using certificate pinning
- ANSSI subCA signs certificates for Google domains “by mistake”(2013)

  “the intermediate CA certificate was used in a commercial device, on a private network, to inspect encrypted traffic with the knowledge of the users on that network.”

Leap of faith

Can we know whether a certificate is “valid” ?

- Online Certificate Status Protocol (OCSP) RFC6960
  - Alternative to CRLs
  - It also doesn't really work/scale
  - Privacy concerns (client reqs over HTTP)
- OCSP Stapling (TLS Certificate Status Request) (RFC6066)
  - Instead of CAs replying to client OCSP reqs, server obtains a signed, time-stamped, OCSP response to send to clients
  - Only Firefox and Microsoft support it (Google doesn't like OCSP)
- Certificate pinning
  - Stick certain certificate fingerprints inside browser's code
  - !scale
- Certificate Transparency (RFC6962)
  - Public, auditable logs of certificates
- Perspectives / Convergence
  - Distributed checking of certificate fingerprints

DNSSEC

Who controls DNS

- DNS root?
  - IANA of ICANN (US organization)
- VeriSign (bought Network Solutions in 2000) runs .com/.org
- Every country owns(?) their own TLD(s)

DNSSEC Intro

Problem: how do we know DNS replies are to be trusted ?

Solution: let's sign zone records with public key certificates

- NEW RR:

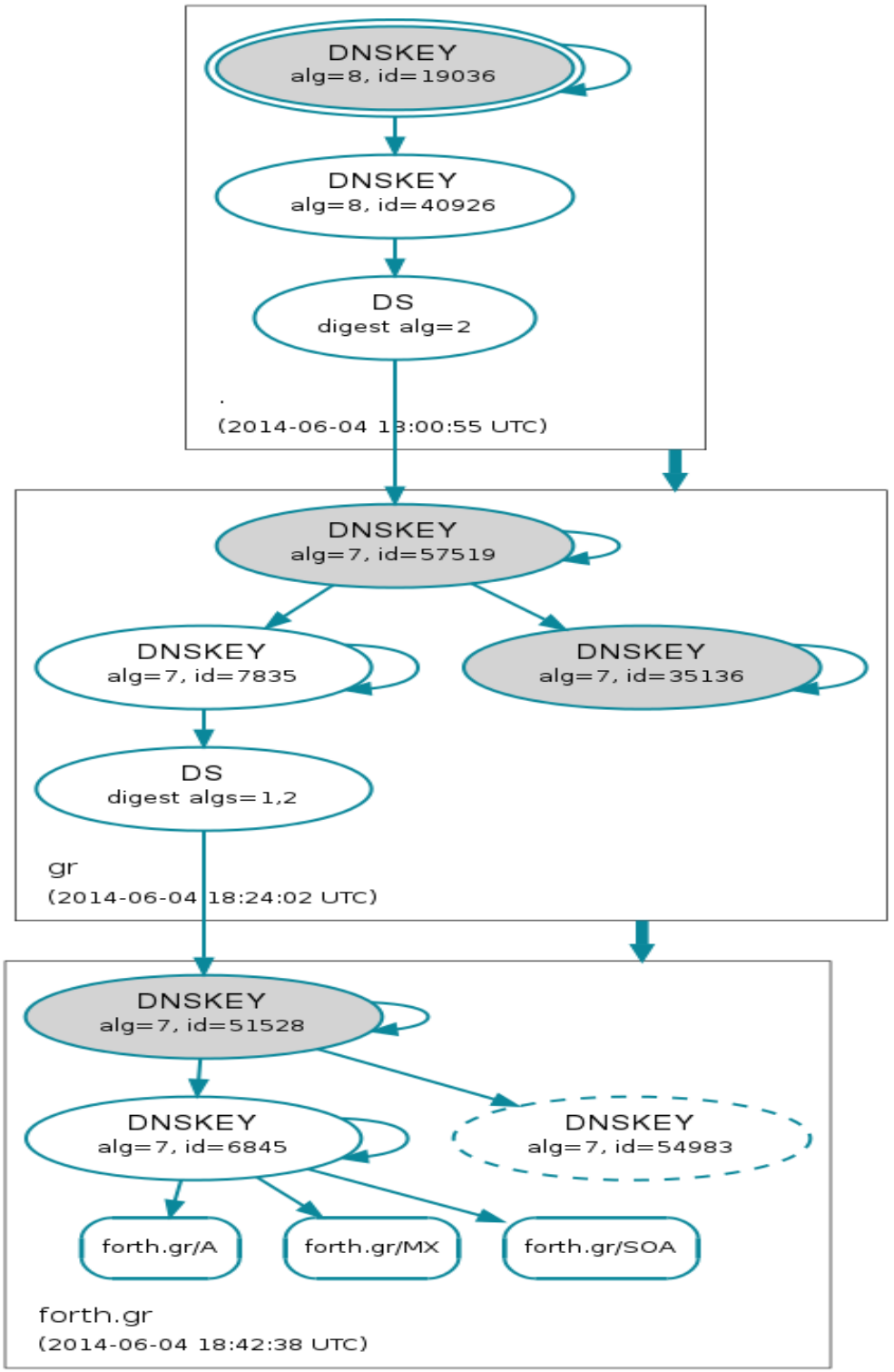
  RRSIG: signed record of A,AAAA,NS,MX,TXT,etc

  DNSKEY: public keys (KSK,ZSK) in the zone

  NSEC/NSEC3: a way to handle NXDOMAIN

  DS Record: Reference of DNSKEY (KSK) in the parent zone.

- ZSK: key that signs the zone records
- KSK: key that signs the zone keys (can be exported as DS)


DNSKEY
alg=8, id=19036
DNSKEY
alg=8, id=40926
DS
digest alg=2
.
(2014-06-04 13:00:55 UTC)
DNSKEY
alg=7, id=57519
DNSKEY
alg=7, id=7835
DNSKEY
alg=7, id=35136
DS
digest algs=1,2
gr
(2014-06-04 18:24:02 UTC)
DNSKEY
alg=7, id=51528
DNSKEY
alg=7, id=6845
DNSKEY
alg=7, id=54983
forth.gr/A
forth.gr/MX
forth.gr/SOA
forth.gr
(2014-06-04 18:42:38 UTC)

DNSSEC History

- Domain Name System Security Extensions ~1995 (RFC2065)
- Domain Name System Security Extensions ~1999 (RFC2535)
- DNS Security Introduction and Requirements ~2006 (RFC4033/4034/4035)
- Minimally Covering NSEC Records and DNSSEC On-line Signing ~2006 (RFC4470)
- DNS Security (DNSSEC) Hashed Authenticated Denial of Existence ~2008 (RFC5155)

- **Root zone signed/published ~ 07/2010**

DNSSEC + Certificates

We can verify DNS records, why not use them for PKI ?

- Storing Certificates in the Domain Name System (DNS) ~2006 (RFC4398)
- The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA ~2012 (RFC6698)
- DNS Certification Authority Authorization (CAA) Resource Record ~2013 (RFC6844)

DANE

New RR: TLSA

- Fingerprint of a certificate signed by a CA
- Fingerprint of a self-signed certificate
  - so who needs CAs any more ?

- Symantec(VeriSign) biggest CA and .com/.org owner
  - No incentive to promote DNSSEC

DNSSEC trust

- ICANN (they control the root keys)
- "Every country owns(?) their own TLD(s)"
- Registrars!

Ever heard of a security aware registrar !?!?

PKI vs DNSSEC

Trust every (sub)CA vs Trust every registrar

Trust

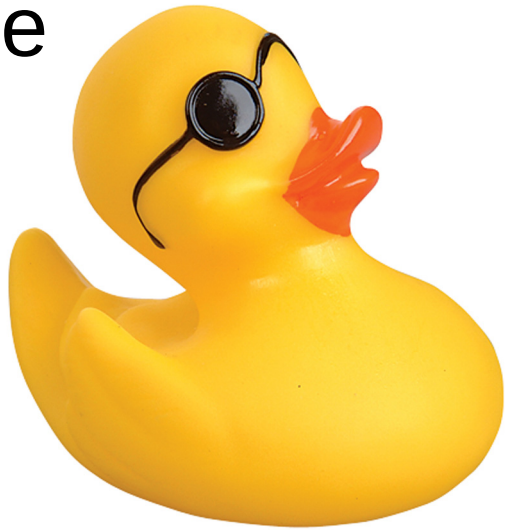
Who do you actually trust when you visit an HTTPS site ?

How many organizations can you trust in the path ?

For how long ?

Security in the interwebs

Theory vs Practice



is everything useless?

NO!

Protocols and implementations will **ALWAYS** have flaws

We need better and more crypto!

The Future

Centralized trust has failed again and again...

Can I has decentralized trust plz ?

Thank You!

Questions ?

Interesting Links

At least

https://www.trustworthyinternet.org/ssl-pulse/

https://www.ssllabs.com/projects/ssl-survey/

https://blog.skullsecurity.org/2013/padding-oracle-attacks-in-depth

http://resources.infosecinstitute.com/beast-vs-crime-attack/

https://media.blackhat.com/us-13/US-13-Smyth-Truncating-TLS-Connections-to-Violate-Beliefs-in-Web-Applications-WP.pdf

http://perspectives-project.org/

http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update/black-tulip-update.pdf

http://w3techs.com/technologies/overview/ssl_certificate/all

http://www.nic.cz/public_media/IT11/prezentace/den2_04_peter_eckersley.pdf

https://www.eff.org/files/colour_map_of_CAs.pdf

https://bugzilla.mozilla.org/show_bug.cgi?id=581901

https://www.harica.gr/documents/CPS-EN.pdf

https://www.eff.org/files/DefconSSLiverse.pdf

https://s3.amazonaws.com/files.cloudprivacy.net/ssl-mitm.pdf

https://bugzilla.mozilla.org/show_bug.cgi?id=724929

https://www.imperialviolet.org/2014/04/29/revocationagain.html