The Beautiful Features of SSL And Why You Want to Use Them?

Holger Reif <holger@reif.net>

Open Source Software Convention
1999/08/24

Content

- What is SSL?
- Apache based SSL servers
- mod_ssl
- Crypto basics
- SSL basics
- Server certificates and CAs
- Session concept and caching
- SSL and Authentication
- Client certificates
- Selected aspects

What is SSL?

- SSL = Secure Socket Layer
- ancestor of TLS
- What is TLS?
  - Transport Layer Security
- Protocol that sits between TCP/IP socket and application
- developed since 1994
- TLS published as RFC
- current version: TLS 1.0 (SSL 3.1)

What can SSL?

- secure your data transport
  - secure tunnel for applications
- provide secured access to protected content (intranet usage)
  - better authentication mechanisms
- protect from some types of spoofing attacks
  - handshake needs interaktion

What can SSL not?

- enhance your overall server security
  - at the tunnel’s end the data are clear again
- process credit cards
  - you can only secure the transport
- provide for non-repudiation
  - application data are not secured themselves

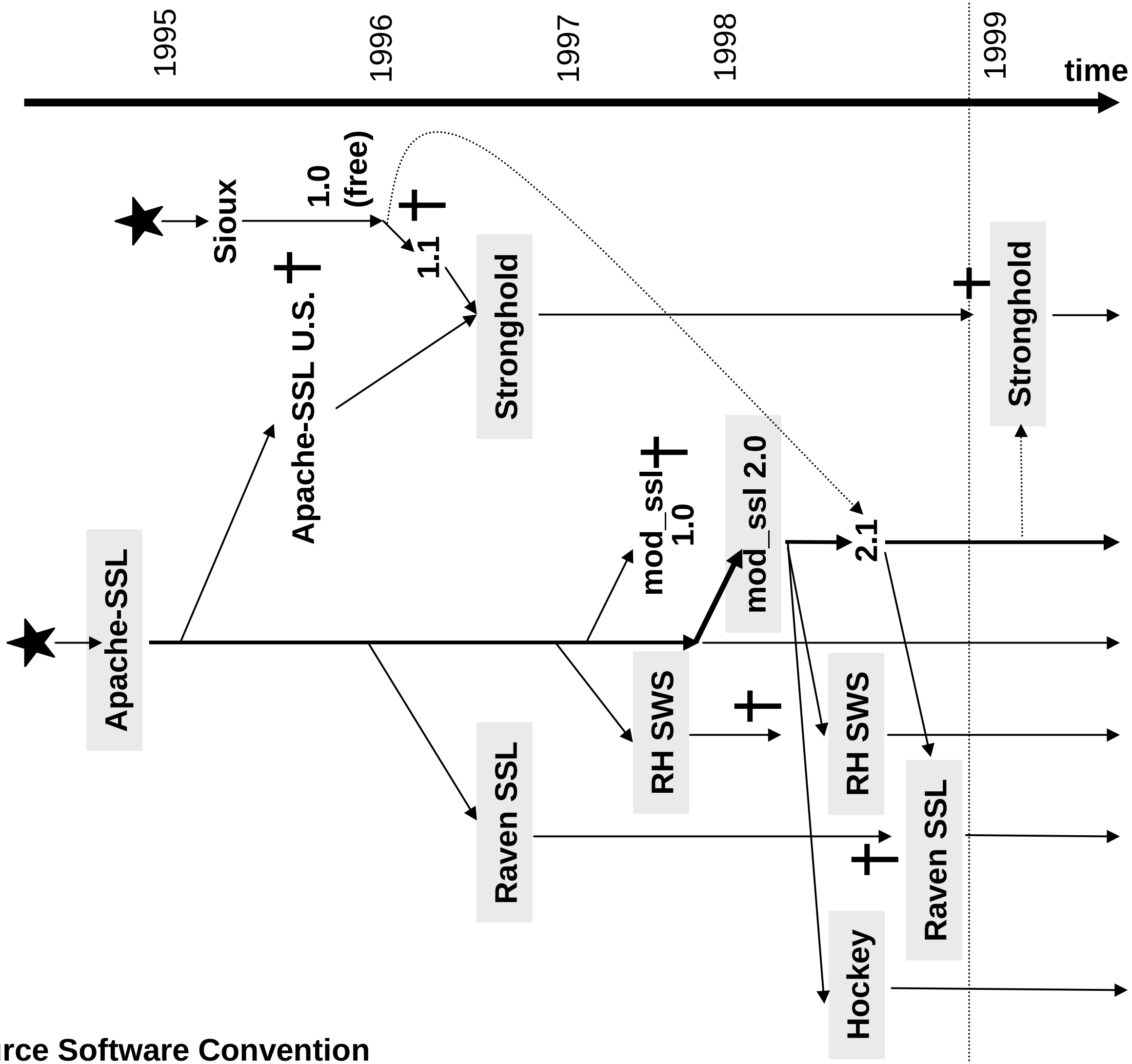
Design goals of SSL

- Cryptographic secure
  - to much snake oil out there
- Interoperability
  - Can two person speaking same protocol communicate?
- Extensibility
  - What about new requirements?
- Relative efficiency
  - don’t require to much resources!

Apache based SSL server

- History
- What is available - a comparison
- Suggestions for arguments when you need to choose

Apache based SSL server history



Holger Reif <holger@reif.net>

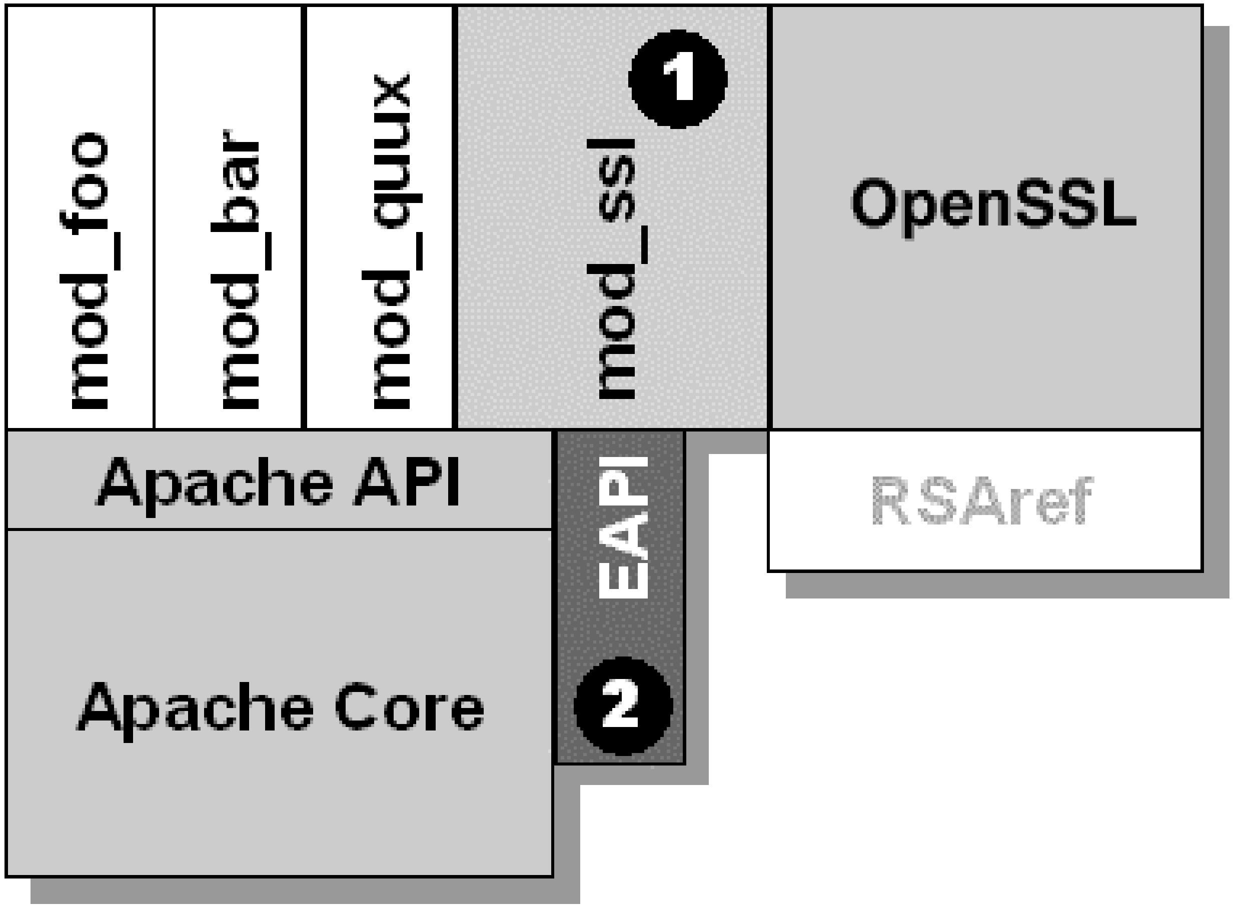
What is available - a comparison

Product	*Apache-SSL*	*mod_ssl*	*RH SS*	*Raven SSL*	*Stronghold*	*Hockey*
Author	B. Laurie	R. Engeschall	RedHat	Covalent	C2 Net	M. Steiger
Location	UK	DE	US	US	US	US
License	open-source	open-source	commercial	commercial	commercial	commercial
Price	$0	$0	$249 (bundle)	$357	$995	$149
Availability	world wide	world wide	US only	US only	world wide	US only
US Usage	restricted	restricted	unlimited	unlimited	unlimited	Unlimited
Support	voluntary, always free	voluntary, always free	conceding, 90 d. free	conceding, 90 d. free	conceding, 90 d. free	conceding, 90 d. free
SSL Engine	OpenSSL (+ RSAref)	OpenSSL (+ RSAref)	OpenSSL + BSafe	OpenSSL + BSafe	SSLeay	OpenSSL + BSafe
Version	1.38	2.3.10	2.0		2.4.2	2.2.8

Suggestions for arguments when you need to choose

- Legal - crypto export control
  - Apache can’t contain cryptography at all
  - US Products are not available elsewhere
- Legal - intellectual property
  - RSA algorithm patented in USA by RSADSI
  - RC2 treated as trade secret of RSADSI
- quality of documentation?
- probable future developments?
- mandatory support needed?
- flexibility / integration of other modules?

mod_ssl architecture



Holger Reif <holger@reif.net>

Installation of mod_ssl

extract sources

```
$ gunzip -c apache_1.3.6.tar.gz | tar xf -
$ gunzip -c mod_ssl-2.3.10-1.3.6.tar.gz | tar xf -
$ gunzip -c openssl-0.9.4.tar.gz | tar xf -
$ gunzip -c mm-1.0.10.tar.gz | tar xf -
```

build openssl

```
$ cd openssl-0.9.4
$ ./config
$ make
$ cd ..
```

build mm lib

```
$ cd mm-1.0.10
$ ./configure --disable-shared
$ make
$ cd ..
```

Installation of mod_ssl (contd.)

apply mod_ssl to apache

```
$  cd mod_ssl-2.3.10-1.3.6
$  ./configure
         --with-apache=../apache_1.3.6 \
         --with-ssl=../openssl-0.9.4
         --with-mm=../mm-1.0.10
$  cd ..
```

build apache

```
$  cd apache-1.3.6
$  make
$  make certificate
$  make install
$  cd ..
```

test the server

```
$  /usr/local/apache/sbin/httpd -DSSL
$  netscape https://localhost
```

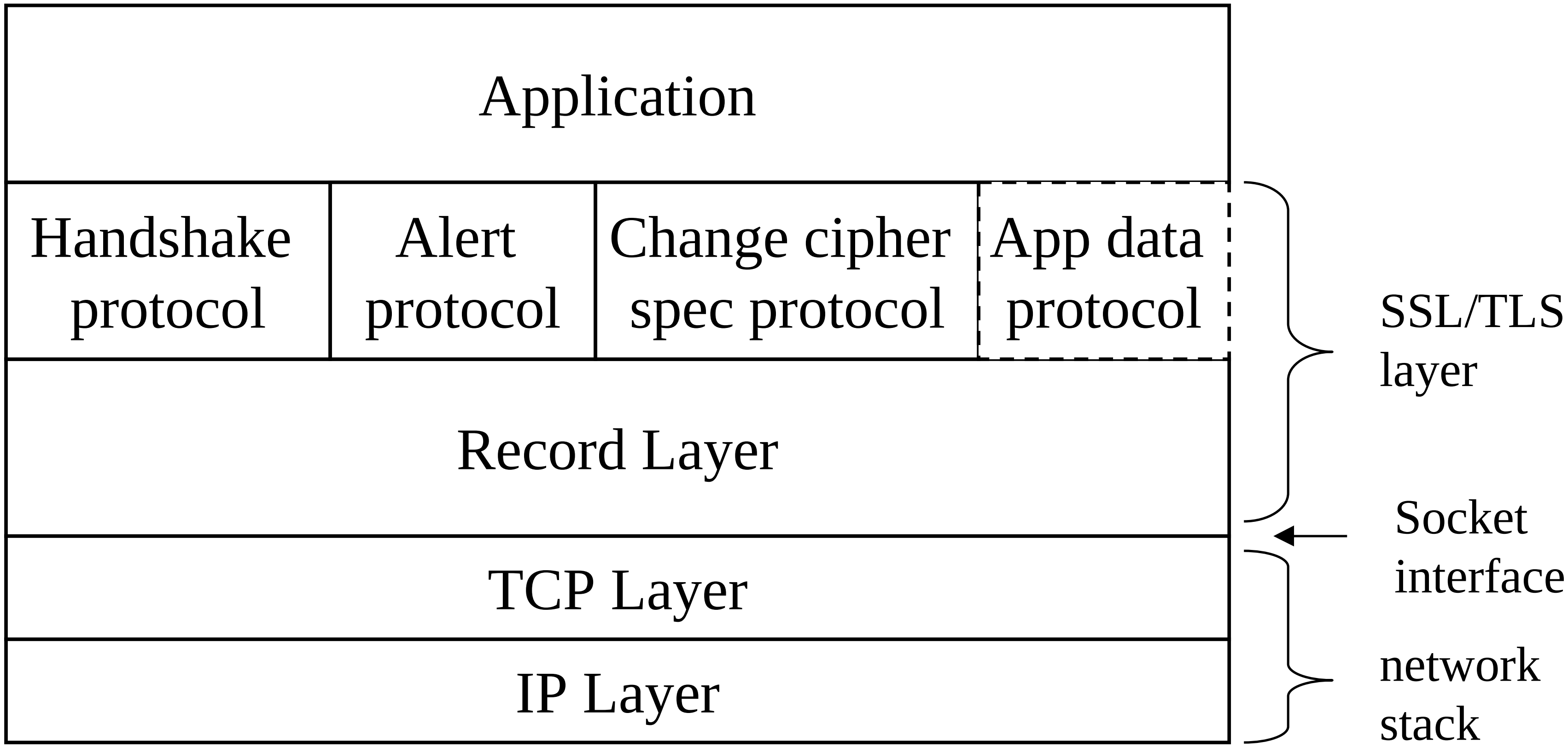
Crypto basics

- Symmetric Cryptography
  – both partners share the same key
- Asymmetric Cryptography
  – key pair: private is secret, public wellknown
  – efficient scaling - PKI (public key infrastructure)
- Hash functions
  – calculates short but unique fingerprint of data
- different combinations in use
  – key exchange

SSL basics

- How is SSL structured?
- The different protocols
- Record Layer
- A full handshake

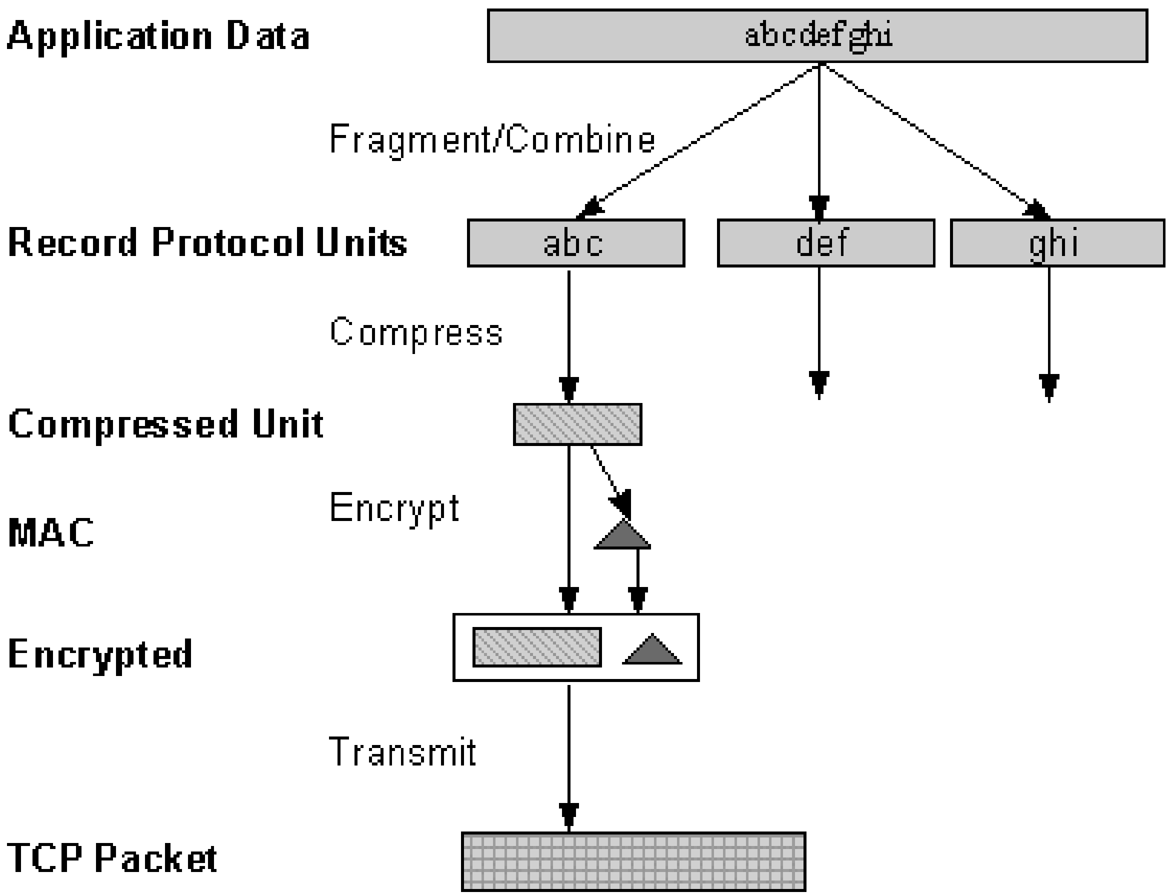
How is SSL structured?



The different protocols

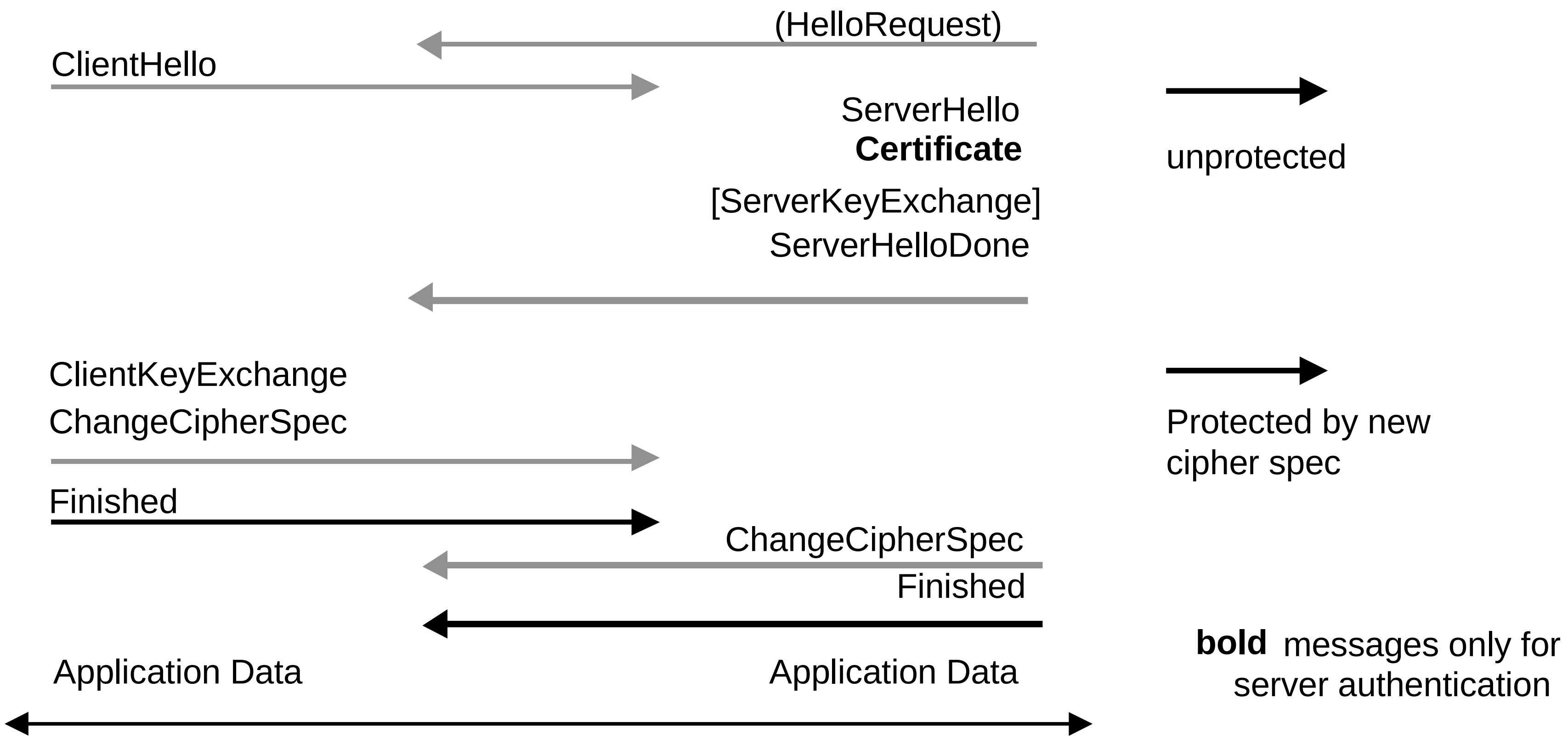
- Record layer (Record protocol)
  - requires *reliable* transport (no missing packets, correct order)
  - Blocking, compression, encryption, integrity
- Handshake protocol
  - (Re-)Negotiate parameters
- Alert protocol
  - Notify about possible problems
- Change cipher spec protocol
  - short cut

Record Layer



picture taken from mod_ssl manual

A full handshake



Why do I need a server certificate?

- Certificate = digital passport
  - your name
  - your (public) key
  - certification authority’s name
  - signature of that authority
- authenticating yourself in the web world
- security to the wrong person is no security at all!

A sample certificate

- screenshot of security info of https://holger.reif.net

Why do I need a CA?

- **CA assures your identity**
- **but you don't need one**
  - **build your own (see later)**
- **Question: Do you accept ID cards issued by an unknown golf club?**
- **popular Browsers have preconfigured CAs**
  - **Verisign, Thawte, lots of others...**
- **You are not recognized *automatically* if you don't have a cert issued by them**

session concept and caching

- Why is the handshake expensive?
- Session concept - Find a way to avoid usage of server's private Key
- An abbreviated handshake (session resume)
- session caching concepts

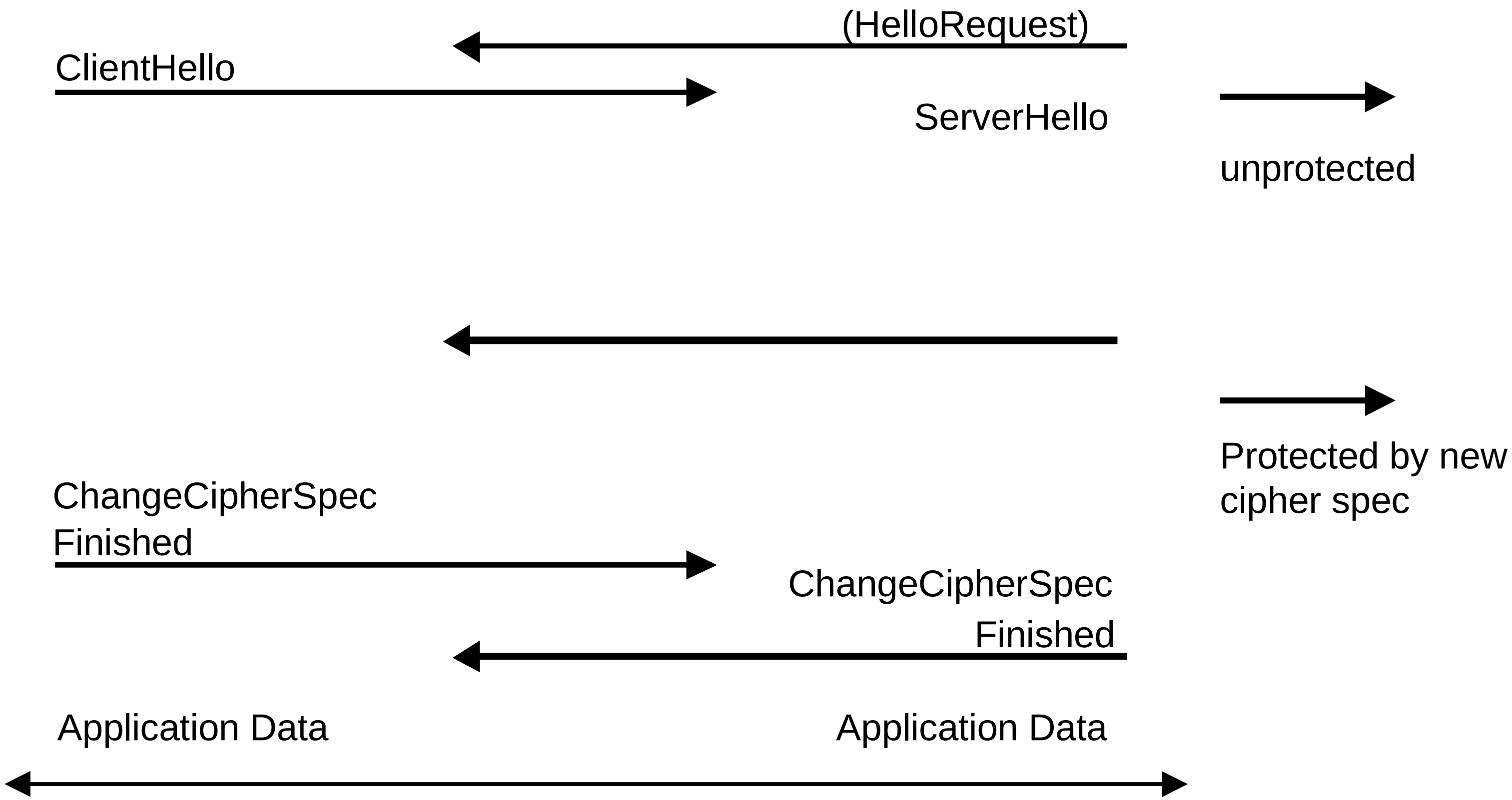
Why is the handshake expensive?

- Client side
  - generating random nonce
  - generating a random secret
  - checking a signature with CA’s public key
  - encrypting random secret with server public key
  - calculating key from raw material (hash)
- Server side
  - generating random nonce
  - decrypting random secret with private key
  - calculating key from raw material (hash)

Session concept - Find a way to avoid usage of server’s private Key

- secret values
  - premaster / master secret
- Ciphersuite
  - compression, key exchange, authentication, encryption, MAC
- cryptographic parameters
  - encryption keys
  - integrity preserving keys
  - initialization vectors
    - $\Rightarrow$ Session Keys

An abbreviated handshake (session resume)



Session caching concepts

- Separate process Approach (Apache-SSL)
  - *gcache* connected over socket (TCP/IP or UNIX domain socket)
    - ⇒ SSLCacheServerPath /path/to/gcache_exe
    - ⇒ SSLCacheServerPort 12345 | /path/to/socket
  - can (in principle) work across multiple servers
- DBM approach (mod_ssl)
  - locally stored in vendor or mod_ssl supplied DBM library
    - ⇒ SSLSessionCache dbm:/path/to/dbmfile
  - stable solution w/o problem of child processes

Session caching concepts (contd.)

- **Shared memory (mod_ssl)**
  - **hash table in memory**
    - $\Rightarrow$ **SSLSessionCache shm:/path/to/keyfile**
  - **extremely fast**
  - **not very portable**
  - **not available on every platform**

SSL and Authentication

- **host based authentication**
- **cookie based authentication**
- **Basic authentication**
- **client based authentication - *FakeBasicAuth***
- **client based authentication - *SSLRequire***

Host based Authentication

- only certain IP addresses allowed
- usual problem: IP-Spoofing
  - addressed by handshake
- remaining problem: man in the middle (MITM) sitting at intermediate router

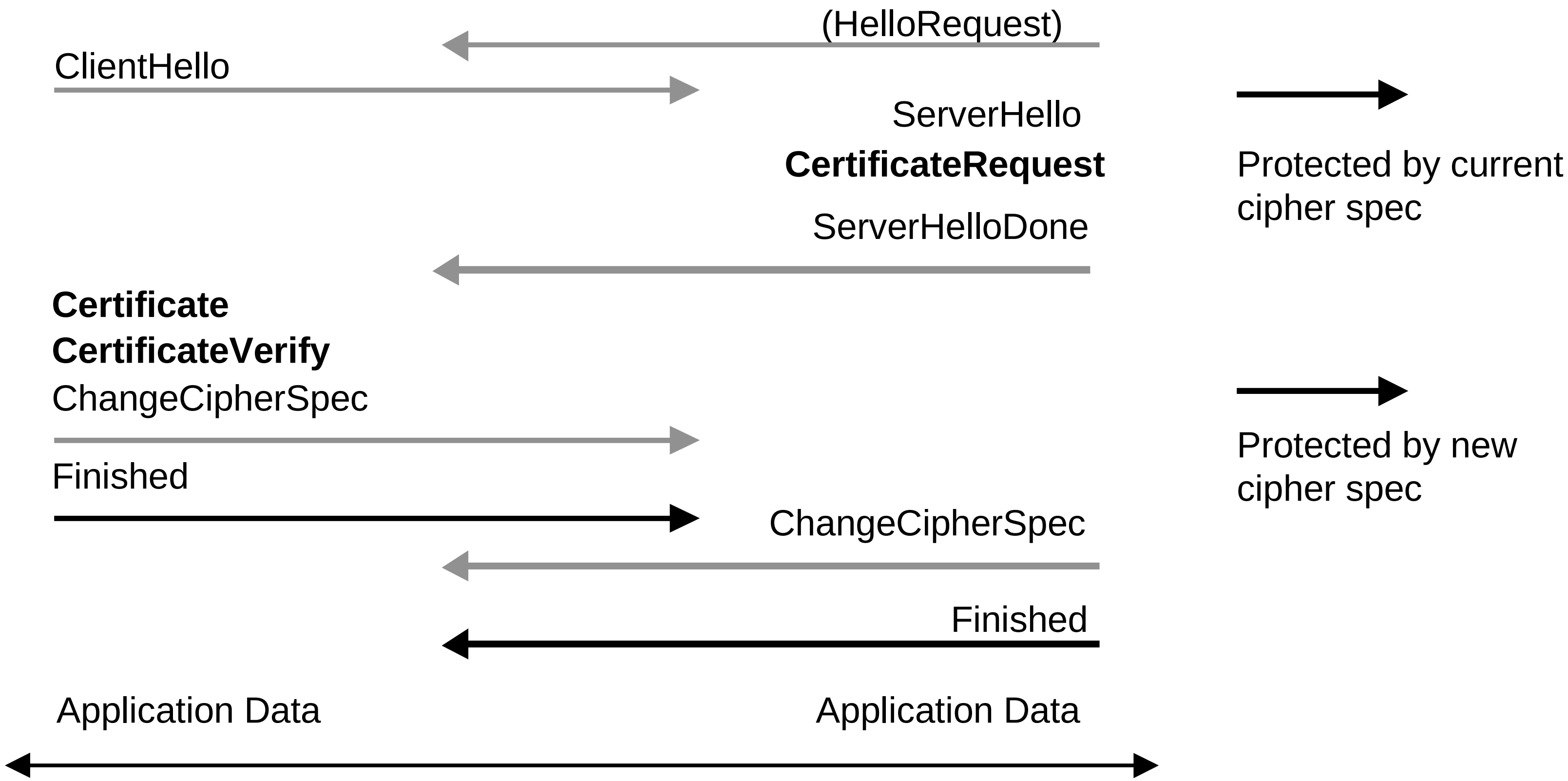
Cookie based authentication, Basic authentication

- none can observe authentication data
- none can spoof data
  - improved security!
  - But cookies are stored unprotected on user's disk
- works the same way as with non-SSL hosts
  - already understood
  - easy utilization
- First starters step-up

SSL state-of-the-art authentication: client certificates

- overcomes the problems of passwords and stored cookies
- Contains authenticated information (e.g. name, age, affiliation)
  - no need for further questions
- user has better control over his information
  - can provide different certificates to different servers
- deploys PKI (public key infrastructure)
- step to single logon
  - instead of multiple passwords a single certificate

Renegotiation (with client certificate required)



Traditional FakeBasicAuth directive

- Simplest approach
- maps subject's DN into Basic Authentication user name
  - password always set to "password" (encrypted: xxj31ZMTZzkVA)
  - inflexible
    - ⇒ No distinction between different CAs
    - ⇒ No grouping according to structure in DN
  - Just works...

Authentication check within CGI

- SSL modules export a lot of environment variables
- access to whole certificate (opt.)
- fine grained access to certificate content via variables
- information about cipher strength via variables

SSLRequire Approach of mod_ssl

- mod_ssl comes with new directive: SSLRequire
- requirement is a regular expression
- CGI variables available
- incorporates aspects from host based access
- can be combined with other requirements
- not only for client authentication usable

Renegotiation again

- SSLRequire might force a renegotiation
  - cipher not strong enough
  - client cert not requested during initial handshake
  - client cert issued by special CA wanted
- per directory requirements not known during initial handshake (cf. name based SSL Host problem)
- supported by all OpenSSL based Apache solutions
- feature needed for “Global Server IDs”

Example: SSLRequire Approach of mod_ssl

- **access for bearers with a recently issued client certificate**
- **with strong cryptographically protected SSL connection**
- **during normal working hours**
- **or access from the intranet**

```
SSLRequire (
   (     %{SSL_CIPHER} !~ m/^(EXP|NULL)-/ \
     and %{SSL_CLIENT_S_DN_O} eq “Snake Oil, Ltd.” \
     and %{SSL_CLIENT_S_DN_OU} in { “staff”, “CA”, “Dev”}
   \
     and %{SSL_CLIENT_V_START} >= 19990504 \
     and %{TIME_WDAY} >= 1 and %{TIME_WDAY} <= 5 \
     and %{TIME_HOUR} >= 8 and %{TIME_HOUR} <= 20 \
   ) or  %{REMOTE_ADDR} =~ m/^192\.76\.162\.[0-9]+$/
```

Client certificates - be your own CA

+ full control over issuing process
+ ability to control the cert content
+ low price for additional certs
+ tight integration of identification

- need for secure key storage
- fight with CA management software
- fight with browser “compatibility”
- keep it running

Open Source software for your own CA

- OpenSSL: ca utility
  - customization with configuration file
  - several support scripts available
  - no full life cycle management
  - just simple
- pyhton-ca (by Michael Ströder)
  - better user interface
- OpenCA project
  - not completed yet, but have a look at it

Outsourcing the CA task

+ trusted third party (TTP) identifies your clients and issues them with certificates
+ TTP is specialized to deliver cutting edge PKI technology
+ many to choose from
  – “standard” internet CAs
  – local companies
+ more competition than on server cert market
  – get the best price for your requirements
- cost intensive

Thawte Strong Extranet

- Hybrid approach
  - Thawte operates the CA facilities
  - you do the identification
- based on Thawte’s Freemail cert program with extended enrollment process
- cert extensions contain “zones” with information controlled by you
- cert extension can be grabbed by CGI programs to do authorization
- supported by mod_ssl (and Apache-SSL?)

Certificate Revocation

- **exclude specific users from access**
  - **private key lost or stolen**
  - **individual left organisation**
- **check them within SSLRequire directive**
  - **inflexible and costly**
- **use a black list issued by the CA: CRL (Certificate Revocation List)**
- **check them automatically**
- **SSLCARevocation{File|Path}**

Selected Aspects

- Randomness aspects
- Security of private server key
- Global Server IDs
- Architectural aspects of mod_ssl

Randomness aspects

- needed for
  - random values in handshake sequence
  - temporary keys
    - ⇒ 1024 Bit server key and export cipher
    - ⇒ seldom: server cert contains only signature key
- server have few random sources
- go for external sources!
  - SSLPassPhraseDialog exec:/your/rng/program
- use operating system resources
  - SSLPassPhraseDialog file:/dev/(u)random

Security of private server key

- password protected single files (or DBM file)
  - automatic startup (and even graceful restart) problematic
  - passphrase caching of mod_ssl simplifies the task
  - PassphraseDialog makes more sophisticated scenarios for secure start
- clear text keys protected by file system
  - only root can read these files
- used keys are in memory anyway

Global Server IDs

- support for “128 Bit certificates” included
- lot of engineering necessary
  - spec not open source :-(
  - no real certs for testing available
  - renegotiation support by OpenSSL needed
- explicit certificate chain required
  - special Verisign root signed intermediate cert
  - intermediate key signs server cert

Additional features of mod_ssl

- EAPI: patch once, use many
- EAPI provides
  - Context Attachment Support for Data Structures
  - Loosely-coupled Hook Interface for Inter-Module Communication
  - Direct and Pool-based Shared Memory Support
  - Additional Apache Module Hooks
  - Specialized EAPI Goodies

Apache based SSL server (contd.)

- EAPI decouples Apache and mod_ssl (and OpenSSL) development
  - one EAPI version per Apache release
  - mod_ssl itsself uses only (E)API calls and doesn't touch the source
- DSO support (*mod_ssl* as dynamic object!)
- provides EAPI based vendor hooks
- makes life easier for package maintainers

Future development

- Improved per directory renegotiations
  - less cryptographic operations
- Full HTTPS support for mod_proxy
  - gather data from a SSL hosts
- SSLListen Directive
  - add the SSL with just one directive
- LDAP support
- Improved stability
- See README.Wishes

Closure

- **Any questions?**
  - **holger@reif.net**
  - **www.modssl.org (+ mailing list)**
  - **www.apache-ssl.org (+ mailing list)**
  - **www.openssl.org (+ mailing list)**
  - **comp.www.servers.unix**