How to get SSL into Apache?

Tutorial at Open Source Convent

99/08/22
Holger Reif

What is SSL?

- SSL = Secure Socket Layer
- Socket == concept of Unix network API
- Means for securing The Web?
- Means for securing _any_ connection oriented communication
- Most often used to secure HTTP (transport protocol for the web)
- Other Protocols use SSL as well (NNTP, POP3, IMAP, ...)

Properties

- authenticates server
- optionally authenticate client
- protect confidentiality (privacy)
- protect integrity (reliability)
- end-2-end security for communication

Typical Applications that use SSL

- e-commerce - ordering
  - ⇒ protect form input sent to server
  - ⇒ sensitive personal data?
- payment
  - ⇒ protect credit card information
  - ⇒ SET is different
- secure web-based intranet access
  - ⇒ secure transmission of confidential content

What can SSL do for you?

- protect data transmitted to and from your server
- reduce some spoofing attacks
- provide reliable authentication
- show your security and privacy awareness

What can SSL *not* do for you?

- protect your server from being hacked
- protect data on your server from being stolen
- provide non-repudiation
- make you a security genius
- secure your server!!!

Competitor: Secure HTTP (S-HTTP)

- Attempt by Terisa Systems 1994-1996
  - $\Rightarrow$ http://www.terisa.com/shttp/intro.html
- Variety of features
  - Client/Server Authentication
  - Spontaneous Encryption
  - Request/Response Nonrepudiation
- better concept then HTTPS and fitted needs better for HTTP
- no browser support
- it‘s dead!

Comparison HTTP, HTTPS, SHTTP

	unprotected content	protected content
unprotected lines	HTTP	S-HTTP
protected lines	HTTPS	

History of SSL

- developed and introduced by Netscape
- 1994 version 1 had serious security flaws and was never used
- 1994 version 2 in Navigator 1
- 1995 battle against PCT by MS (Private Communications Technology)
- 1996 version 3 in Navigator 3 - public review
- 1996-1999 TLS (transport layer security) a.k.a. SSL 3.1 by IETF

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!

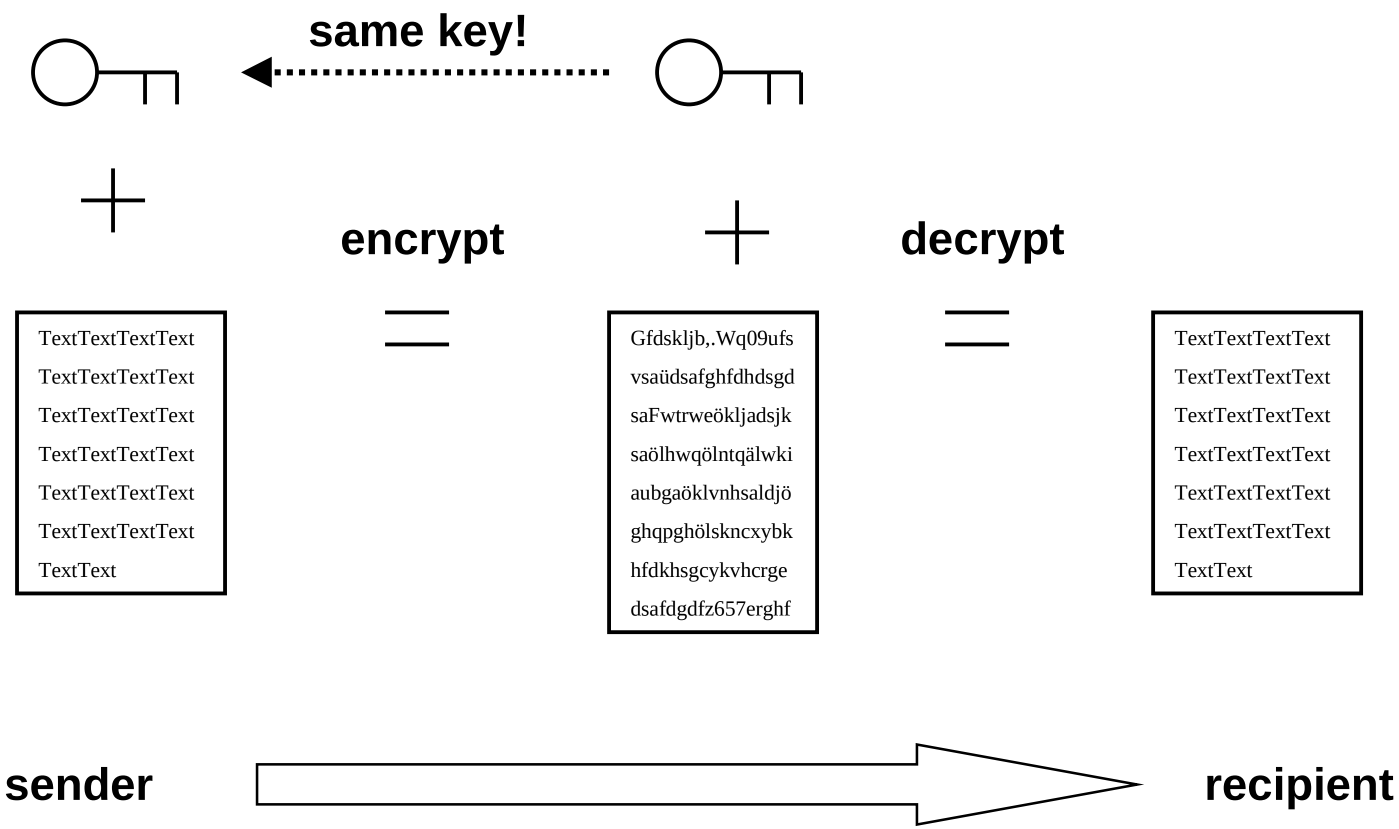
Ways to reach the goal

- Symmetric Encryption
- Asymmetric Authentication
  - digital signatures
- clear specification
- Parameter negotiation
  - Handshake
  - parameter reuse

Excursion: Crypto Primer

- Basic Algorithms
  - Symmetric (Secret Key) Cryptography
  - Asymmetric (Public Key) Cryptography
  - Hash Functions
- Their Use
  - Session keys / Enveloping
  - Digital signatures
  - Certificates, x.509, CAs, cert chains and CRLs
- Strength of ciphers
  - Key length

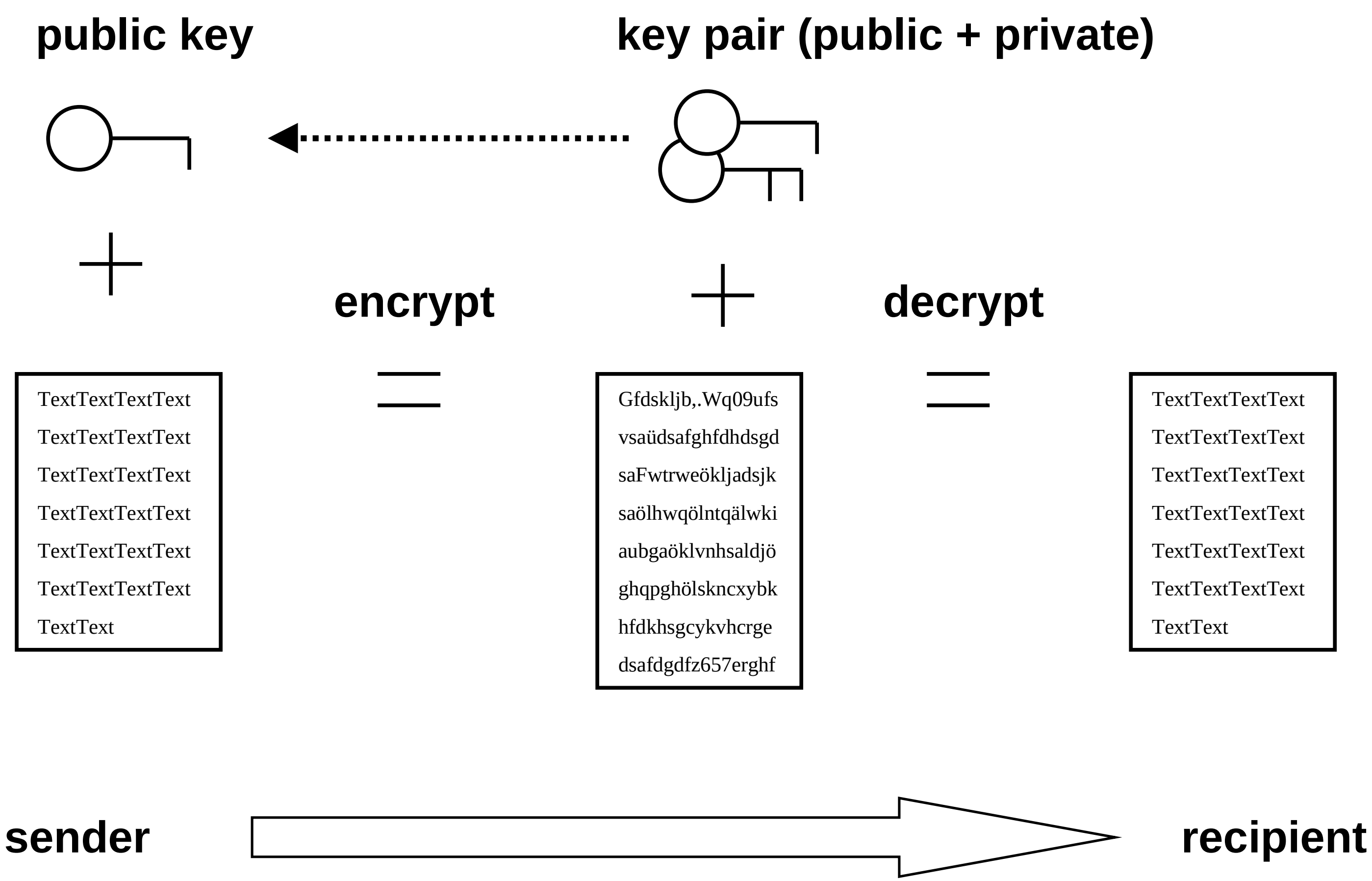
Symmetric (Secret Key) Cryptography



Symmetric (Secret Key) Cryptography (contd.)

- same key for both sender and recipient
- key needs to kept secret!!!
- Problem: key distribution doesn‘t scale
- work fast (Mbytes per second)
- mostly bit shuffling, depending on key
- examples: DES (56 bit), 3-DES (112 bit), RC4 (128 Bit)

Asymmetric (Public Key) Cryptography



Asymmetric (Public Key) Cryptography (contd.)

- Recipient has key pair (public + private part)
- Sender needs public key
- Public key needs only to kept authentic (not secret!)
- underlaying math problem
  - e.g. factoring a product of two large primes (RSA)
- you can make public key operations fast
- slow private key operations (dozends a sec.)
- Examples: RSA, Diffie-Hellmann

Hash Functions

- no keys involved
- one way function
  - can’t regenerate original from the hash
- compresses size down to 128 or 160 Bit
- generates a fingerprint of the text
  - changing one bit in the text changes half the bits of the hash
- work fast (Mbytes per second)
- Examples: MD5, SHA-1

Use of cryptographic functions

- Session keys / Enveloping
- Digital signatures
- Certificates, x.509, CAs and cert chains

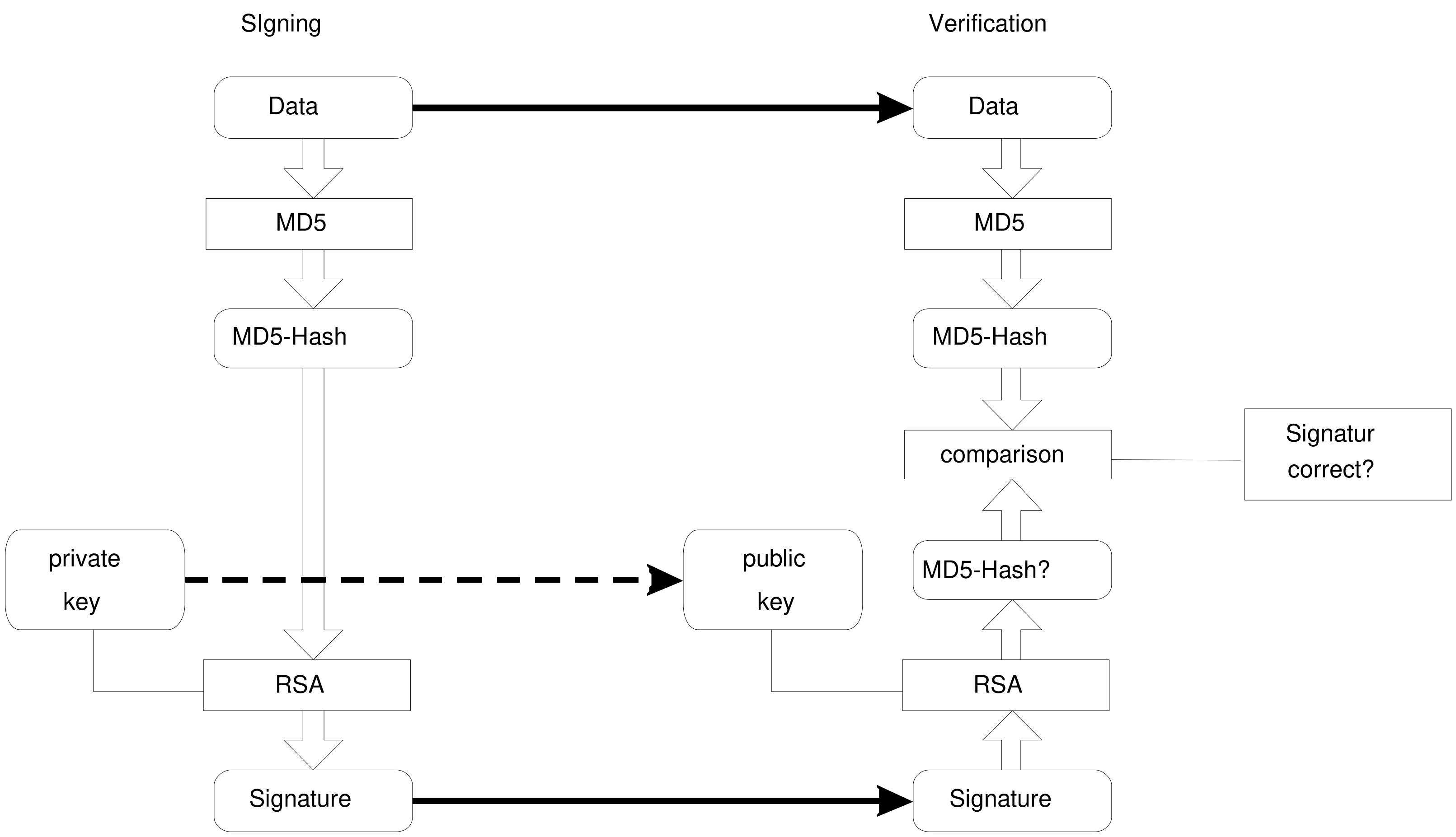
Data encryption: Session keys and Enveloping

- **Problem**
  - secret key crypto: key distribution
  - public: speed
- **Solution: put your text in an envelope!**
- **securely agree on a (symmetric) session key with your Public Key**
- **fast bulk encryption with that symmetric key**
- **long term key exchange key (rarely used)**
- **compromise of short term Session key harmless**

Digital signatures

- Idea: use your private key to show your will
- Problem: private key operations slow on huge documents
- Solution:
  - generate a hash
  - encrypt this hash with your private key

Digital signatures (contd.)



Certificates

- **Problem: how do I know it's *your* pub key?**
- **Solution: use a passport!**
- **Certificate is a digital passport**
  - **contains identity of holder**
  - **public key of the holder**
  - **tied together with an issuers signature**
  - **opt.: name of the issuer**
  - **opt.: administrative information like date of issuance and validity**
  - **opt.: allowed usage (type of pass)**

Certificates (contd.)

- How to prove to be the valid holder of the cert (no picture included)?
  - prove possession of private key
  - encrypt a challenge
  - verifier checks with pub key taken from cert

- “Your key is your identity”

X.509

- Important format for certificates
- Coined by ITU (Intl. Telcom. Union) 1988-1997
- broad usage
  - ⇒ SSL/TLS
  - ⇒ S/MIME
  - ⇒ IPSec
- current version: V3
  - v1 starting
  - v2 provided for better identification of used keys
  - v3 provides for extensions

Example: X.509 certificate

Data:

Version: 1 (0x0)

Serial Number: 15179 (0x3b4b)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc, OU=Certification Services Division, CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Jan 22 15:14:10 1999 GMT

Not After : Feb 5 15:14:10 2000 GMT

Subject: C=DE, ST=Bavaria, L=Dachau, O=The OpenSSL Project, OU=Security Services Division, CN=www.openssl.org

Example: X.509 certificate (contd.)

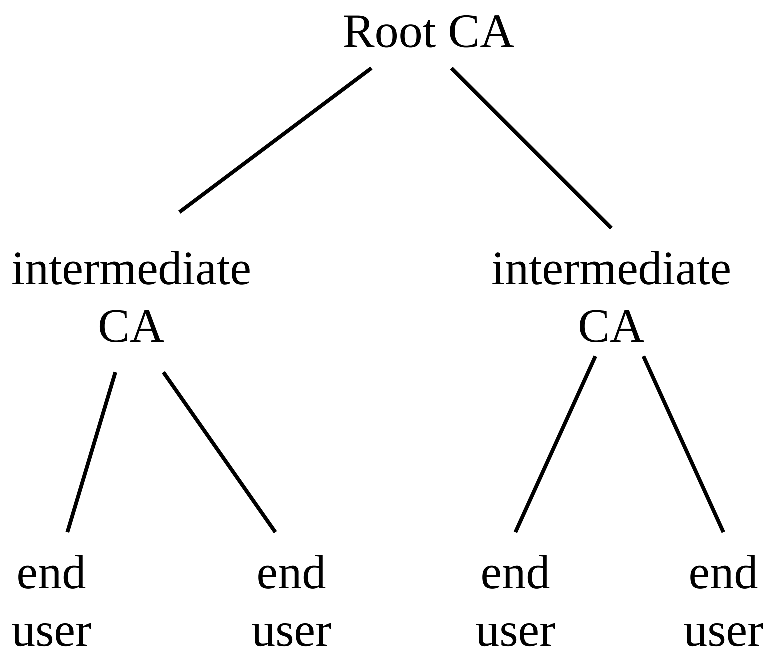
```
Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        RSA Public Key: (1024 bit)
           Modulus (1024 bit):
              00:cd:04:8f:03:df:f1:dc:0c:30:63:bf:14:e8:59:
              ........
              11:05:ac:8c:42:2a:97:10:da:ba:01:7e:62:aa:39:
              51:2f:93:fa:f0:48:36:47:ed
           Exponent: 65537 (0x10001)
  Signature Algorithm: md5WithRSAEncryption
     8d:5d:b5:cf:0e:b1:d3:11:1a:5f:bf:fa:7b:ed:b8:e1:24:2e:
     ........
     35:3b:19:d4:30:13:b8:6c:2c:ea:b4:c2:3f:e8:98:8e:35:44:
     c5:79
```

Certification Authorities

- Problem: how do I verify the validity of the cert?
- Solution: check the signature of the issuer
- Problem: How do I know issuer’s pub key?
- Doesn’t he has a certificate itself...
- Endless loop???
- You need to trust somebody
  - ⇒ Trusted third parties identify the individual for you and issue them with a certificate
  - ⇒ Certification Authorities
  - ⇒ embedded into browsers or email clients

Certificate chains

- **How to adopt the model of hierarchies?**
  - **Organizations**
  - **different roles and pruposes**
- **Solution: hierarchical CAs**
- **Consequence: certificate chains**



Certificate chains example

C=ZA
ST=Western Cape
L=Cape Town
O=Thawte Consulting
OU=Certification Services Division
CN=Thawte Personal Basic CA
Validity 1995.12.31 to 2020.12.31

the root cert

the intermediate (issuing) cert

C=ZA
ST=Western Cape
L=Durbanville
O=Thawte Consulting
OU=Thawte SB RSA IK 1998.9.16 17:55
CN=Thawte Server Basic RSA Issuer 1998.9.16
Validity 1998.09.16 to 2000.09.15

my personal cert

CN=Thawte Freemail Member
Email=Holger.Reif@SmartRing.de
Validity 1999.07.24 to 2000.07.23

Certificate Revocation Lists (CRL)

- Problem: What happens if a certified key is stolen and can be misused?
- Solution: issuer has a blacklist of invalid (revoked) certificates -> CRL
- Must be checked additionally
- CRL contains
  - ⇒ Issuer's name
  - ⇒ date of issue
  - ⇒ serial numbers of revoked certificates
  - ⇒ signature of issuer

Strength of ciphers

- how to attack a cipher
- key length
- key length comparison

How to attack a cipher?

- search through all possible solutions (brute force)
  - go through key space
  - SSL challenge 8/1996
    - ⇒ break 40-Bit RC4
  - DES challenge at RSA conference in January
    - ⇒ break 56 bit DES
- find a principle problem in the algorithm
  - complete compromise
  - shortcut to be faster than brute force

Key length

- strength of modern crypto only lies in keys (otherwise: security by obscurity)
- key length ==> strength
- strength == effort to break a cryptographically secure message
- effort measured in time, space, computing power etc.
- relation length/strength different various algorithms

Key length comparison

- symmetric encryption: only brute force
  - 128 Bit == 2^128 possible keys
- asymmetric encryption: mathematical tricks (factoring, ...)
  - 1024 Bit == 200 digit number factoring
  - provided no better technique than factoring known!
- Digests: birthday attack
  - 160 Bit

Legal

- Patents, trade secrets and the like
- export control
- usage restrictions

Patents, trade secrets and the like

- RSADSI has a patent on RSA algorithm (expiring 2000/09/20) in USA
- IDEA is patented across Europe (Ascom)
- RC2, RC4 is considered a trade secret by RSADSI
- several other cryptographic algorithms are protected as well
- Ask your lawyer!

Export control

- (strong) crypto is treated as munitions
- munitions can’t be easily exported from USA
  - except for weak cryptography
  - except to Canada (general exception)
  - except for foreign subsidiaries of US companies (on request)
  - except for financial purposes (on request)
  - "export crippled"
- Wasenaar agreement
  - public domain software is not export controlled
  - countries may have stricter handling (cf. USA)
- Ask your lawyer!

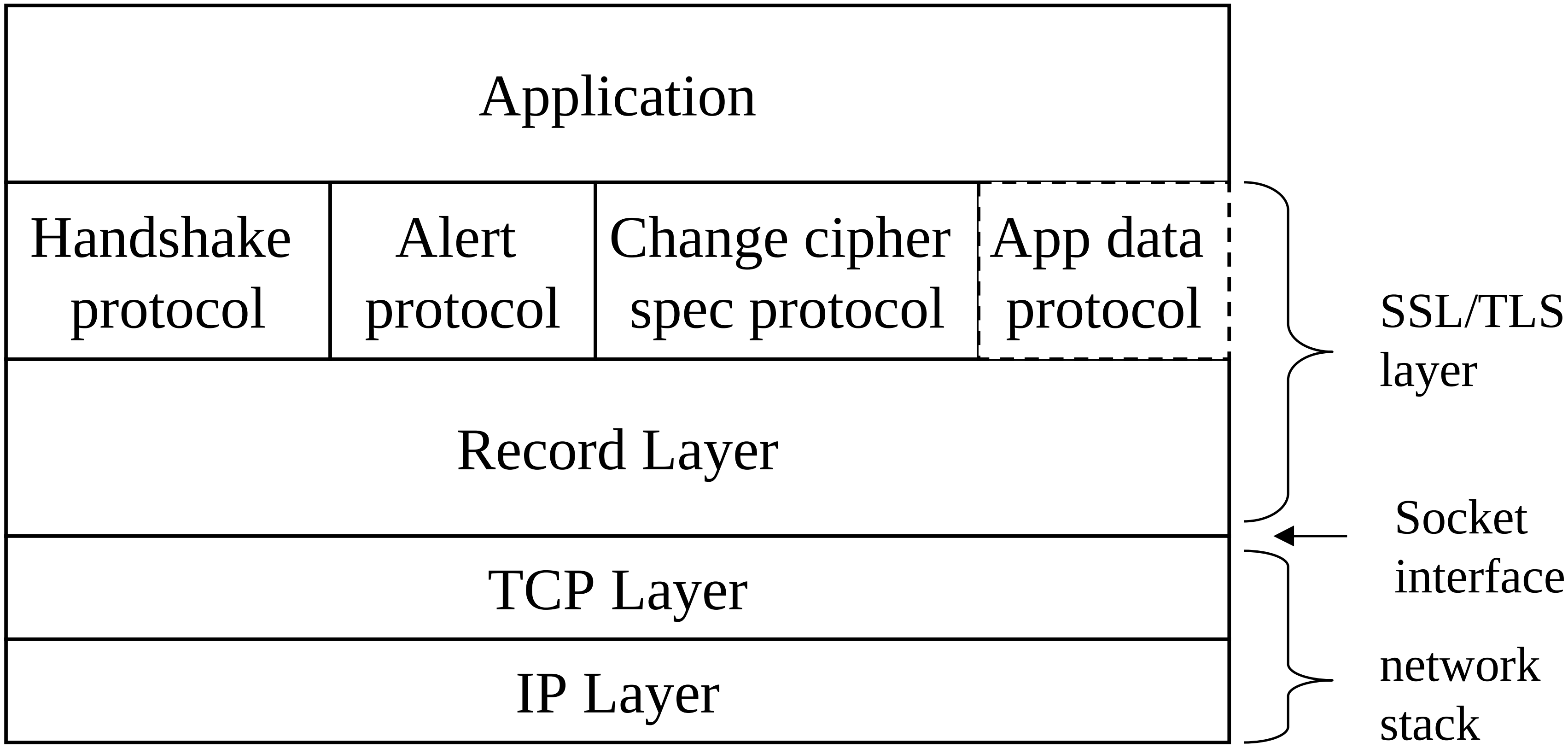
Usage restrictions

- **e.g. France had restrictions in use of crypto until currently (now only declaration is needed)**
- **e.g. Russia demands a license to develop or use crypto**
- **check "Crypto Law Survey" by Bert-Jaap Koops at** http://cwis.kub.nl/~frw/people/koops/lawsurvy.htm
- **Ask your lawyer!**

SSL Protocol Overview

- Protocol components
- Session concept
- Ciphersuite concept
- Handshake Protocol

Protocol components



Protocol components (contd.)

- 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

Session concept

- **secret values**
  - **premaster / master secret**
- **Ciphersuite**
  - **see next slide**
- **cryptographic parameters**
  - **encryption keys**
  - **integrity preserving keys**
  - **initialization vectors**
    - $\Rightarrow$ **Session Keys / Enveloping!!!**

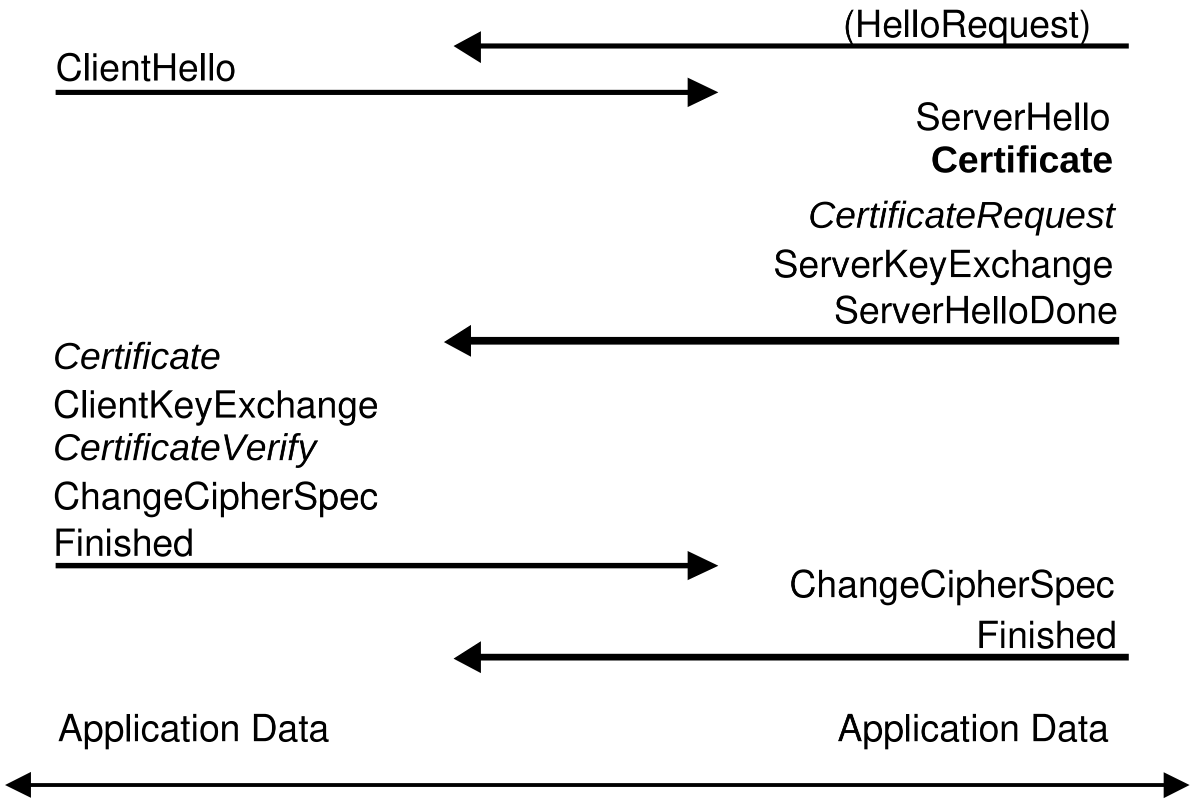
Ciphersuite concept

- compression algorithm
- key exchange algorithm
- authentication algorithm
- encryption algorithm
- MAC algorithm
- examples
  - EXP-RC4-MD5
    - ⇒ Kx=RSA(512), Au=RSA, Enc=RC4(40), Mac=MD5, exp
  - DES-CBC3-SHA
    - ⇒ Kx=RSA, Au=RSA, Enc=3DES(168), Mac=SHA1

Handshake Protocol

- hello messages
- Certificate messages
- Key Exchange Messages
- Finished messages

Handshake Protocol (graphic)



bright messages for session continuing not needed

bold messages only for server authentication

italic messages only for client authentication

Handshake Protocol (contd.)

- **hello messages**
  - start with negotiation
  - exchange random values
  - agree on algorithms
  - check for session resumption.

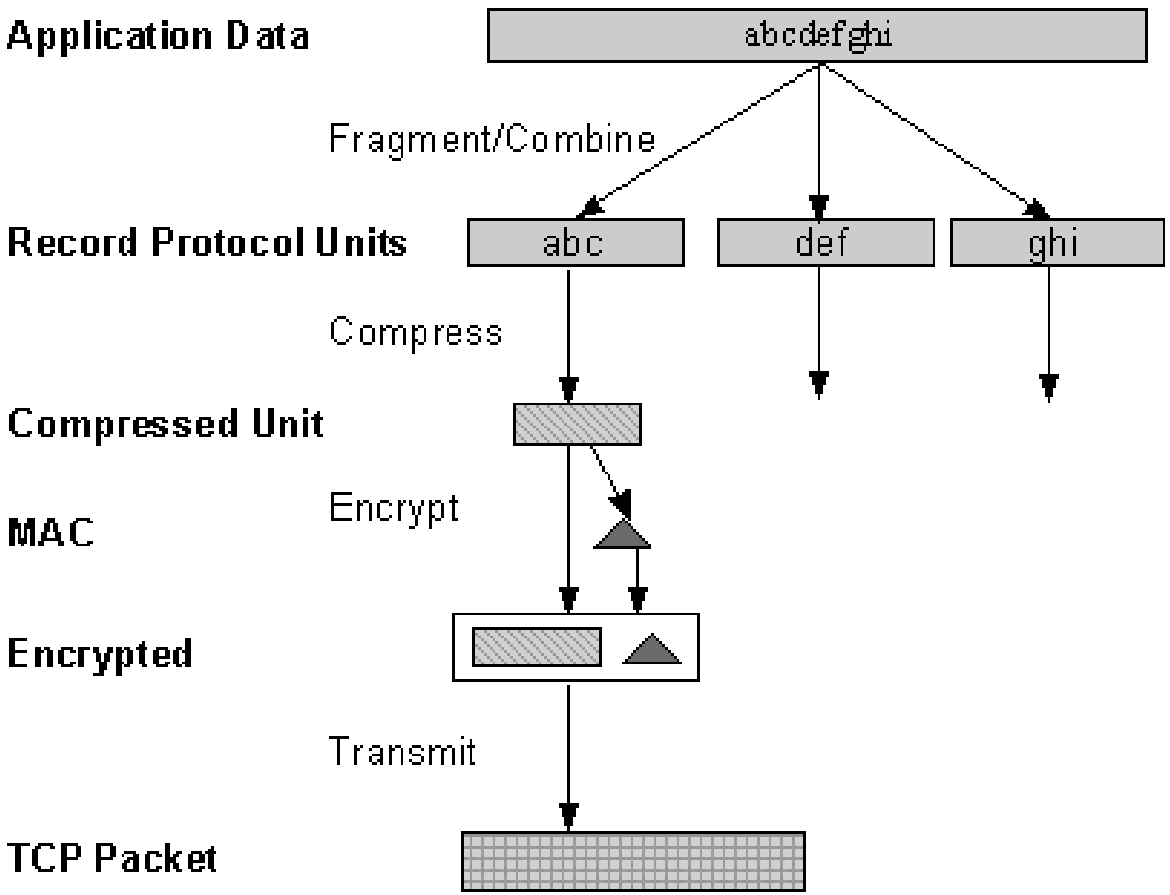
Handshake Protocol (contd.)

- **Certificate messages. Key Exchange Messages**
  - **Exchange certificates and cryptographic information to**
    - ⇒ **allow the client and server to authenticate themselves.**
  - **Exchange the necessary cryptographic parameters to allow**
    - ⇒ **the client and server to agree on shared key**

Handshake Protocol (contd.)

- **Finished messages**
  - **Allow the client and server to verify that their peer has**
    - ⇒ **calculated the same security parameters and that the handshake**
    - ⇒ **occurred without tampering by an attacker.**

SSL processing



picture taken from mod_ssl manual

SSL and Browsers

- The obvious picture
- How to get a better browser
- Proxy solutions

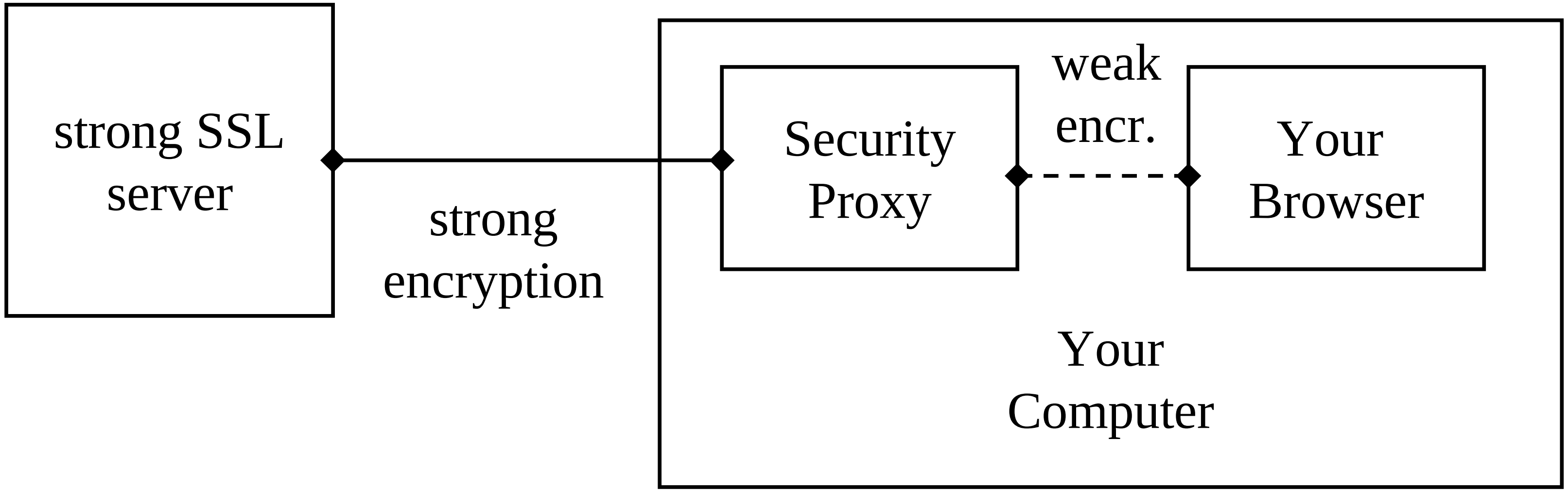
The obvious picture

- most popular browsers (Netscape Navigator, MS Internet Explorer) are from USA
  - ⇒ export crippled
- Opera from Norway
  - ⇒ only for Win platform (yet)
- Lynx+SSL (patches are from US....)
  - ⇒ Lynx enhanced with OpenSSL
  - ⇒ Does anybody really uses this?
- curl
- Mosaic 2.6 was patched with SSLeay 0.4.1 ...

How to get a better browser?

- Fortify
  - Patch your Netscape
  - www.fortify.net
- Have a full strength Crypto Service provider
  - Win + MSIE
- Get US-Versions of the browsers
  - ftp.hacktic.nl
  - ftp.replay.com

Proxy solutions



- **SafePassage**
  - **www.c2.net**
- **stunnel**
  - **http://mike.daewoo.com.pl/computer/stunnel/**

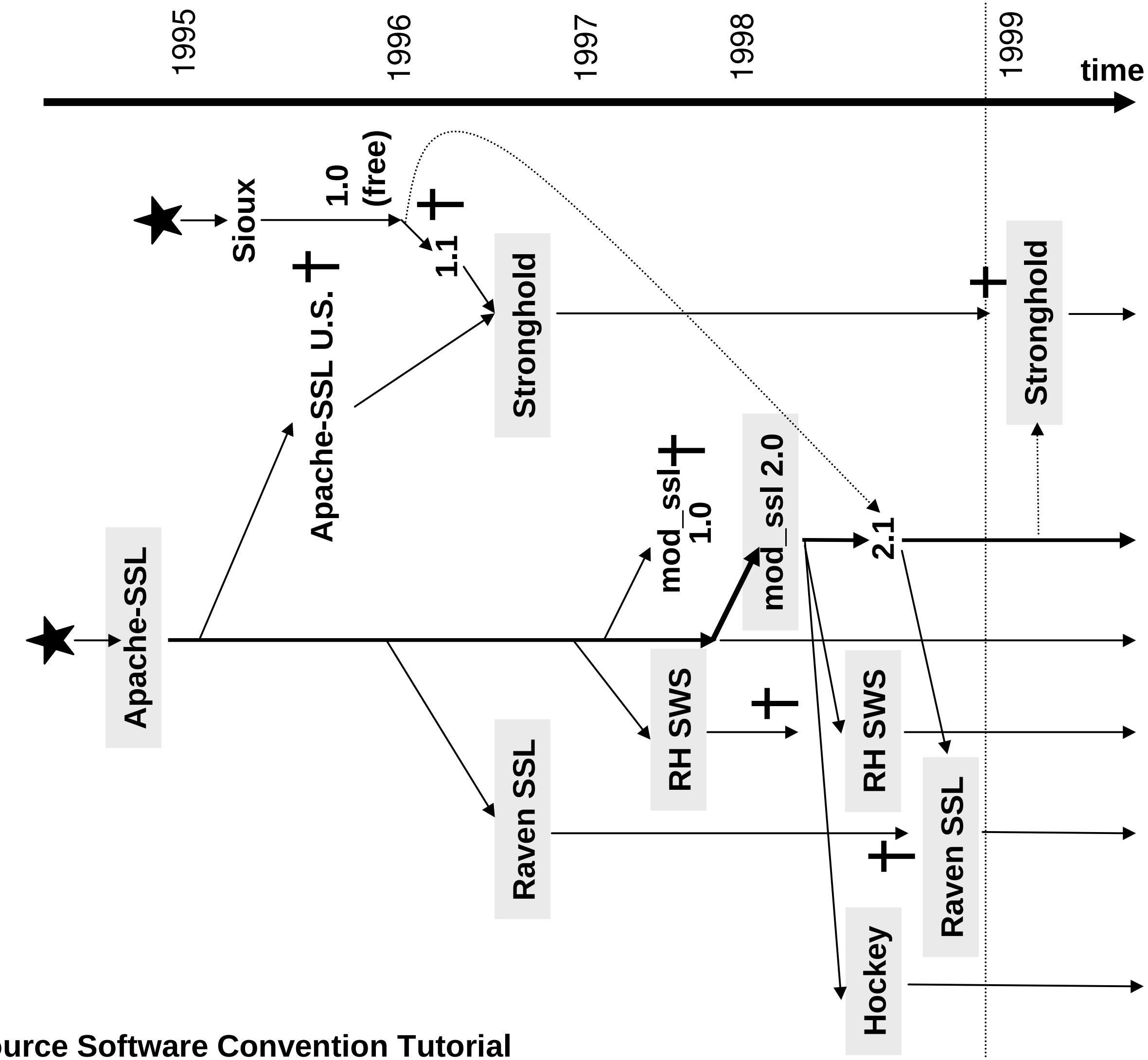
SSL and Apache

- History
- Architecture

History

- Past development
- comparison of solutions

Development in past

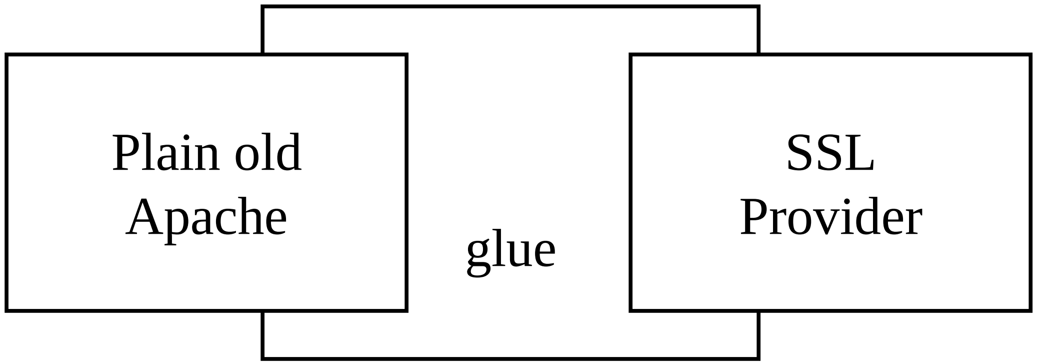


Comparison

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

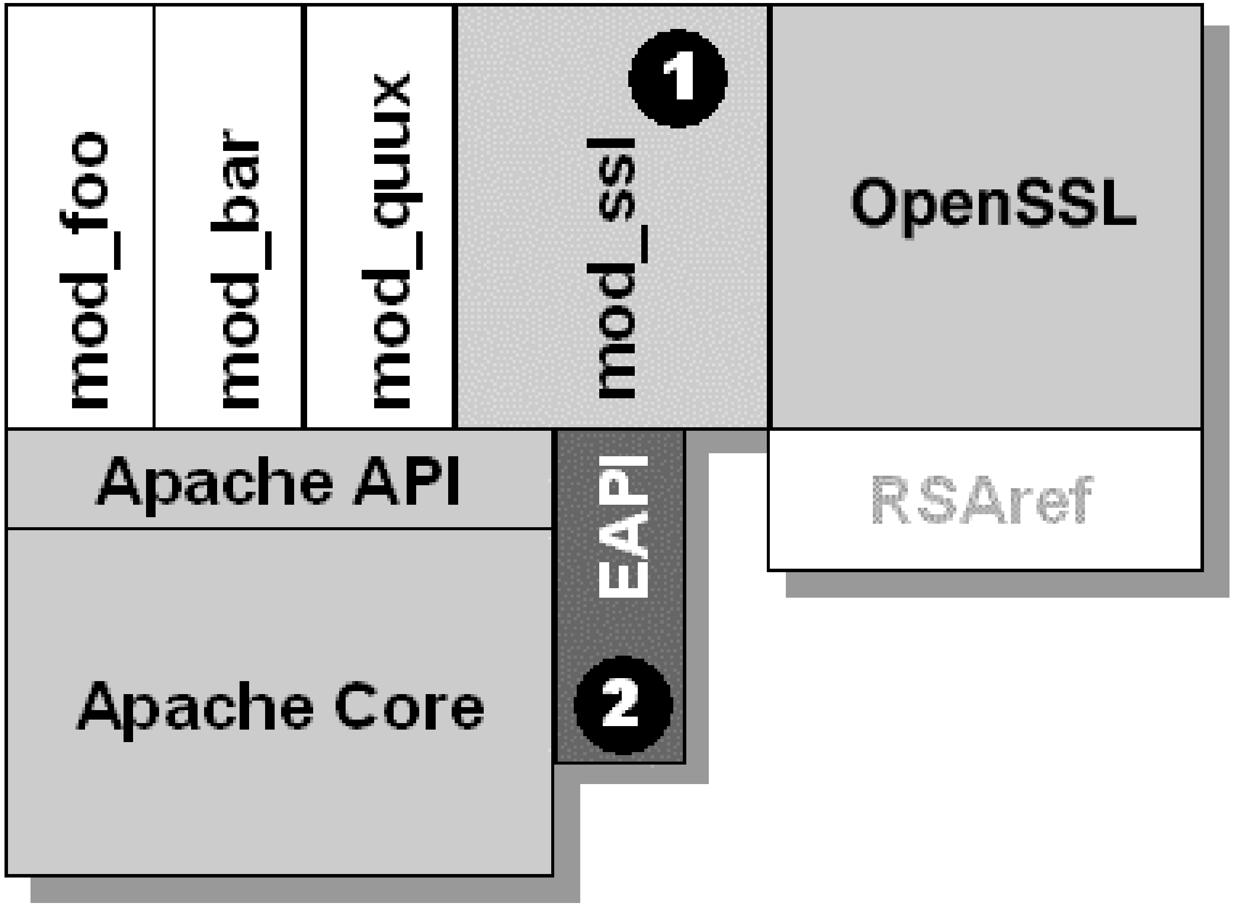
General Architecture

- **3 parts: Apache, SSL provider, glue**



- **why not just one distribution?**
  - **remember: munitions is export controlled**
  - **“Hooks” are disallowed as well**

mod_ssl specific architecture



OpenSSL

- How to build
- History
- Components
- recent developments

How to build (in real time)

- ./Configure
- make
- make test (optional)
- make install (optional)

History of OpenSSL / SSLeay

- SSLeay
  - done by Eric Young and (partly) Tim Hudson
  - First public release 0.4 at 1995/05/31
  - major releases each year
  - only free tool for implementing SSL
  - development partly founded by C2Net
  - Development stopped mid 1999
- OpenSSL
  - started end of 1998
  - active development committed to Open Source

Components

- General purpose Big Number library
- General Purpose Basic Crypto library (ciphers)
- ASN.1 functions
- General Purpose High Level Crypto library (PKCS)
- X.509 library
- General purpose utilities (openssl command)
- SSL/TLS library

Recent development

- better support for various standards
  - PKCS#7 (S/MIME)
  - PKCS#12 (certificate import/export)
- renegotiations handling
- certificate extension handling
- new algorithms
  - OAEP
- cleanups (compiler warnings, configuration handling)

mod_ssl

- recent development
- how to build
- Basic Configuration
- Where do I get my certificate from?
- Advanced Topics
- Future development

Recent mod_ssl development

- shared memory session cache
- DH/DSA support
- CRL support
- session renegotiation
- SSL related status available
- more SSL env vars for CGI programs

How to build (in real-time)

- configure
- cd ../apache
- make

How to configure

- it’s easy, really!
- examining adopted httpd.conf

How to configure - Basic Options

- SSLEngine on
- SSLMutex file:/app/sources/apache_1.3.6/logs/ssl_mutex

How to configure - See what happens

- SSLLog /app/sources/apache_1.3.6/logs/ssl_engine_log
- SSLLogLevel info

How to configure - Set Key and certificate

- SSLCertificateFile /app/sources/apache_1.3.6/conf/ssl.crt/server.crt
- SSLCertificateKeyFile /app/sources/apache_1.3.6/conf/ssl.key/server.key
- SSLPassPhraseDialog builtin

How to configure - Decrease load

- SSLSessionCache dbm:/app/sources/apache_1.3.6/logs/ssl_scache
- SSLSessionCacheTimeout 300

How to configure - Improve Security

- **SSLRandomSeed startup builtin**
- **SSLRandomSeed connect builtin**

How to configure - The certificate

- Make cert (real time demonstration)
- Request a cert from Thawte

The certificate - rolling your own

- **make certificate**

The certificate - request one at Thawte

- You should have
- Generate a key pair
  - $ openssl genrsa -des3 -out server.key 1024
- See what's the content
  - $ openssl rsa -noout -text -in server.key
- Remove the protection from the key
  - $ openssl rsa -in server.key -out server.key.unsecure
- Generate the request from existing key
  - $ openssl req -new -key server.key -out server.csr
- Examine the content
  - $ openssl req -noout -text -in server.csr
- Go to a CA of your choice

The certificate - request at Thawte (contd.)

- Proceed through different steps (see Annex)
  - paste in cert request
  - select contact persons
  - payment and address information
  - finished and summary
  - status page
  - load cert
  - install cert

What to do next

- Some more advanced examples and their effect
  - Client auth
    - ⇒ Check the certs of your clients
  - PRNG seeding
    - ⇒ provide source for real randomness
- Advanced features
  - GID
    - ⇒ get strong encryption from newer export browsers
  - DSO
    - ⇒ compile mod_ssl as dynamic module

Future developments

- 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**