TLS/SSL protocol design

Nate Lawson
nate@rootlabs.com

Presented at Cal Poly
Nov. 29, 2007

root labs

Overview

- Introduction to SSL/TLS
  - Focus on SMTP+SSL
- Design goals and result
- Cryptography primer
  - Desired properties
  - Primitives for implementing them
- Protocol walkthrough in detail
- Attacks and mitigation

My background

- Root Labs founder
  - Design and analyze security systems
  - Emphasis on embedded, kernel, and crypto
- Previously, Cryptography Research
  - Paul Kocher’s company (author of SSL 3.0)
  - Co-designed Blu-ray disc security layer, aka BD+
- Crypto engineer at Infogard Labs
- FreeBSD committer

Security is hard but rewarding

- Protocols and crypto are susceptible to very minor mistakes
- Example: SSL timing attacks over the Internet

- Hard = fun and $
  - Breaking and building things is exciting
  - Security is a desired skill for any resumé

SSL history

- SSL (Secure Sockets Layer) v2.0 (1994)
  - Serious security problems including incomplete MAC coverage of padding
  - Designed by Netscape
- SSL v3.0 (1996)
  - Major revision to address security problems
  - Paul Kocher + Netscape
- TLS (Transport Layer Security) 1.0 (1999)
  - Added new crypto algorithm support
  - IETF takes over
- TLS 1.1 (2006)
  - Address Vaudenay's CBC attacks on record layer
  - Provide implementation guidance

Layered model

- SSL provides security at the transport layer (OSI model L4)
  - Stream of bytes in, private/untampered stream of bytes out
  - Application logic is unmodified
  - Can be adapted to datagram service also (DTLS)
- Compare to IPSEC
  - Mostly used as an L3 protocol

SMTP over SSL

- HTTP, SMTP, POP, IMAP, etc. all have SSL variants
- Two design choices to add SSL
  - Use alternate port since SSL session establishment differs from original protocol
    - SMTPS (TCP port 465 and 587)
  - Add protocol-specific message to toggle SSL mode
    - STARTTLS over port 25 (RFC 3207)
- SMTP session over SSL is unchanged

Security goals

- Privacy
  - Data within SSL session should not be recoverable by anyone except the endpoints
- Integrity
  - Data in transit should not be modified without detection except by the endpoints
- Authentication
  - No endpoint should be able to masquerade as another

Attacker capabilities

- Sorted by increasing power
- Normal participant
  - Can talk to server that is also talking to other parties
- Passive eavesdropping
  - Observe any or all messages sent by other parties
- Active (Man in the Middle)
  - Insert or replay old messages
  - Modify
  - Delete or reorder
- Secure protocols must address all these threats

Crypto property: privacy

- No one other than the intended recipient of a message can determine its contents
- Caveats
  - Adversary could have powers of knowing or choosing plaintext
  - Traffic analysis
    - Length, latency, unencrypted data like IP or Ethernet addresses
    - Side channel attacks: power consumption, EM, timing of operations

Crypto property: integrity

- Any change made to a message after it has been sent will be detected by the recipient
  - Corollary: reordering, replay, insertion, or deletion of messages will also be detected
- Caveats
  - Privacy is not integrity protection
  - Error recovery
    - You can't always terminate the session
  - Root of trust (shared system?)

Crypto property: authentication

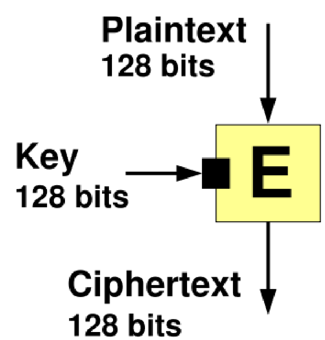
- Messages can be associated with a given identity with high level of confidence
- Caveats
  - Managing identification
    - Lost keys, forgotten passwords, laptop walks away
    - Revocation of old keys and refreshing to new ones
  - Bootstrapping: what is your root of trust?

Security goal implementation

- Privacy
  - Data is encrypted with block cipher (e.g., AES)
  - Cipher key is exchanged via public key crypto (e.g., RSA)
- Integrity
  - Data is protected by a MAC (e.g., SHA1-HMAC)
- Authentication
  - Server and/or client identity is verified via certificates

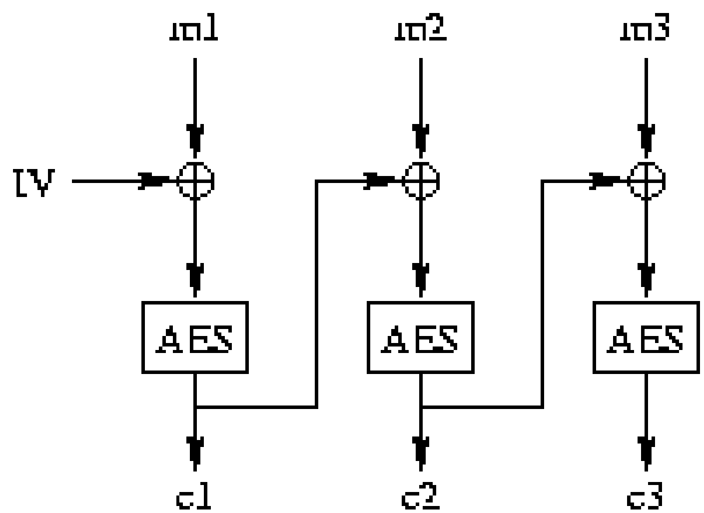
Primitive: symmetric crypto

- Block ciphers turn plaintext block into ciphertext using a secret key
  - Recipient inverts (decrypts) block using same key
- Examples: AES, 3DES, RC5



Primitive: symmetric crypto

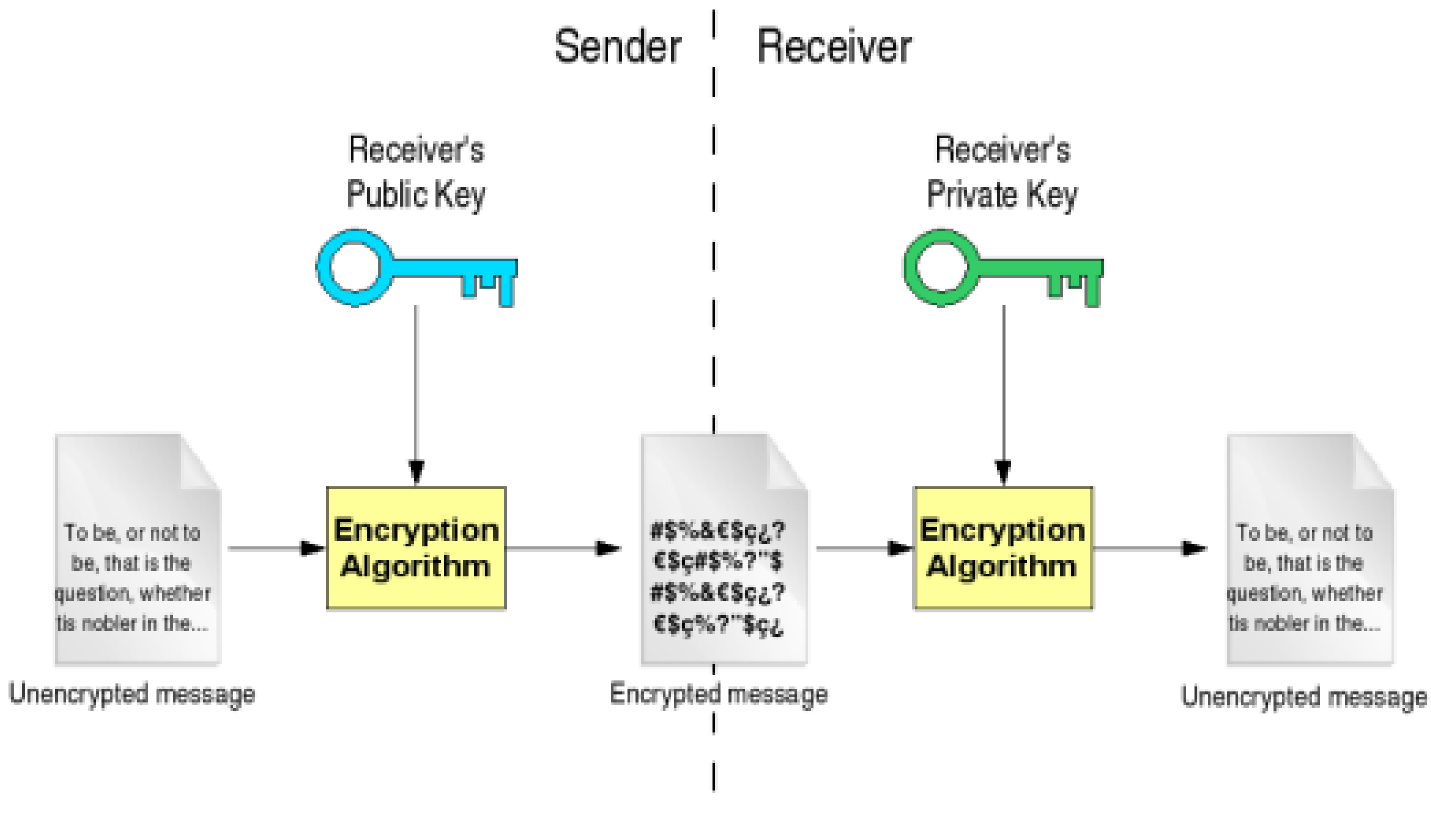
- Often requires "chaining" to encrypt messages longer than a single block
- This does *not* provide integrity protection



Primitive: public key crypto

- Data transformed with one key can only be inverted with the other key (asymmetric)
- Examples: RSA, Diffie-Hellman, DSA
  - And elliptic curve variants
- Can encrypt data to a recipient without also being able to decrypt it afterward
- Can sign data by encrypting it with one key and publishing the other

Primitive: public key crypto



Primitive: certificates

- Associate a name with a public key
  - Trusted party uses private key to sign the message "joe.com = 0x09f9…"
  - Public key of trusted party came with your web browser
- Key management still a problem
  - Expire certs and explicitly revoke them if a private key is compromised (CRL)
  - Or, check with the trusted party each time you want to use one (OCSP)

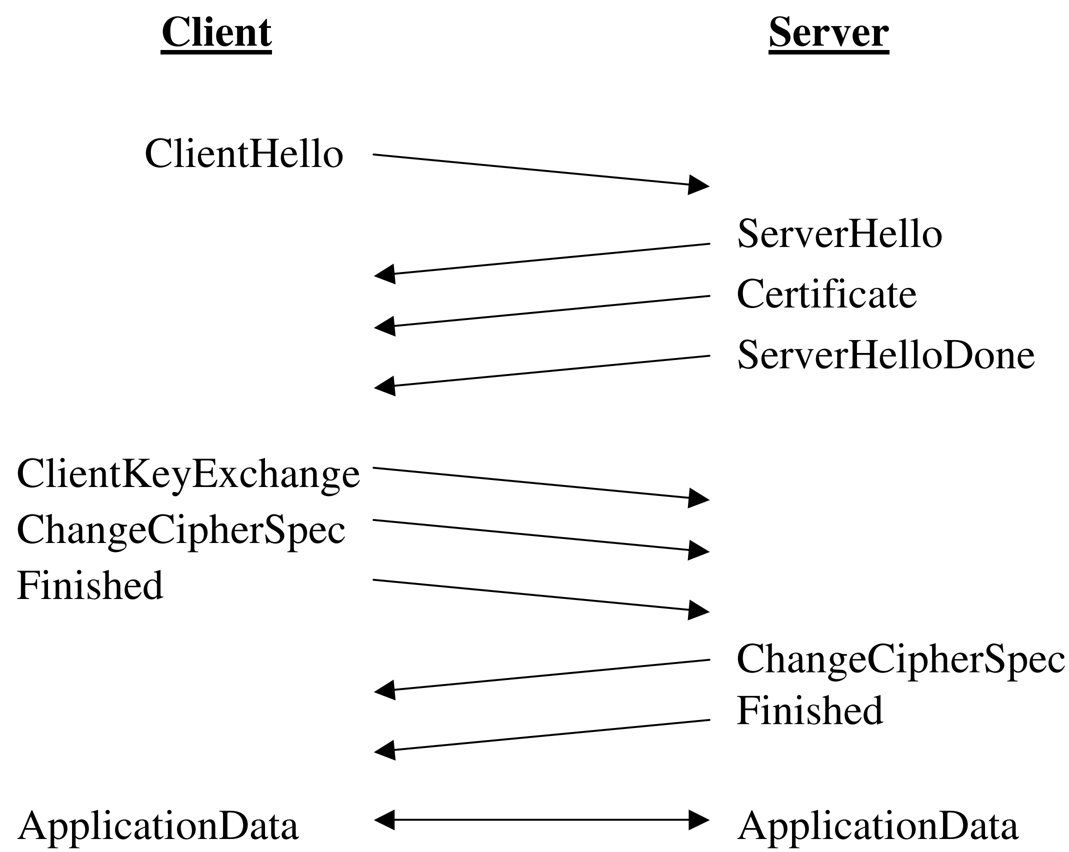
Primitive: message authentication code

- A MAC combines a hash function and secret key with the data to protect
  - Resulting MAC is transmitted with message
  - Recipient performs same process and verifies result matches
- Attacker cannot…
  - Modify message without changing its hash
  - Forge a new MAC value without knowing the key
- Examples: SHA1-HMAC, AES CMAC

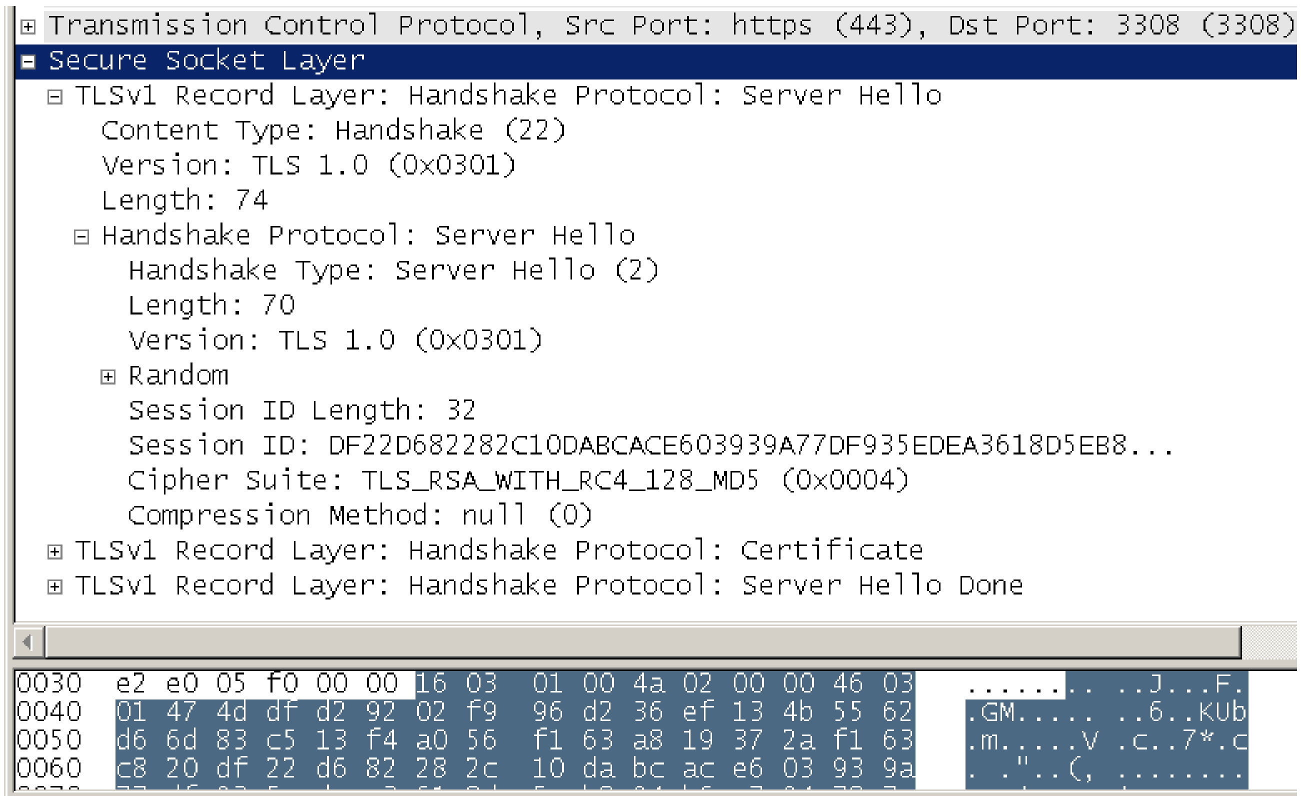
Primitive: secure PRNG

- Outputs a cryptographically-strong, pseudo-random stream of data based on initial seed
  - Initial seed needs to have enough entropy
  - PRNGs used many places (key generation, IVs, nonces)
- Examples: /dev/random, Yarrow
  - Often based on a hash function like SHA-1

Overview of typical session



Decoding with WireShark



Message: Client/ServerHello

- Initiates connection and specifies parameters
  - Initiator sends list (i.e., CipherSuites) and responder selects one item from list
  - SessionID is used for resuming (explained later)

Client/ServerHello

```
Version
RandomData
SessionID
CipherSuites
CompressionMethods
```

Message: Certificate

- Provides a signed public key value to the other party
  - Almost always the server although clients can also authenticate with a cert
  - Other side must verify information in cert (i.e., the DN field is myhost.com = IP address in my TCP connection)

Certificate

ASN.1Cert

Message: ServerHelloDone

- Signifies end of server auth process
  - Allows multi-pass authentication handshake
  - Otherwise unimportant
- Cert-based auth is single-pass

Message: ClientKeyExchange

- Client sends encrypted premaster secret to server
  - Assumes RSA public key crypto (most common)
  - Server checks ClientVersion matches highest advertised version

ClientKeyExchange

```
RSA-PubKey-Encrypt(
    ClientVersion
    PreMasterSecret[48]
)
```

Message: ChangeCipherSpec

- Indicates following datagrams will be encrypted
  - Disambiguates case where next message may be error or encrypted data
- Each side now calculates data encryption key (K)

MasterSecret computation

```
Hash(
     PreMasterSecret
     ClientRandom
     ServerRandom
)
```

Message: Finished

- Indicates all protocol negotiation is complete and data may be exchanged
  - First encrypted message of each party
  - Includes hashes of all handshake messages seen by each side
    - Also, magic integers 0x434C4E54 or 0x53525652 (why?)

Finished

```
AES-K-Encrypt(
      Magic
      MD5(handshake_messages)
      SHA1(handshake_messages)
)
```

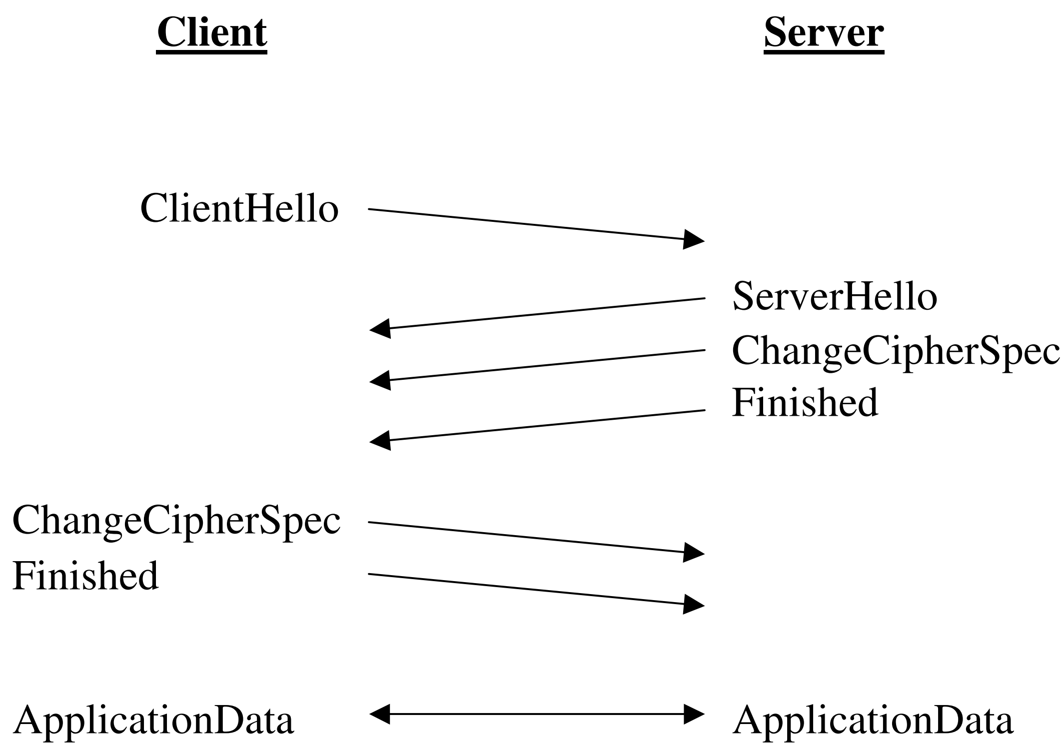
Message: ApplicationData

- Encapsulates encrypted data
  - Includes MAC for integrity protection
  - Can span multiple TCP packets

ApplicationData

```
AES-CBC-K-Encrypt(
     Type
     Version
     Length
     Data
     MAC
     Padding
     PaddingLength
)
```

Session resumption

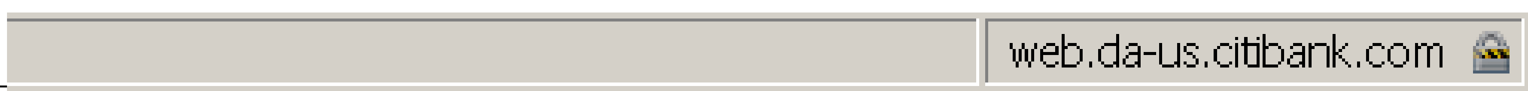


Formal verification of protocol security

- Goal: formal system for finding any security problems in design
  - BAN logic (BAN89)
    - Formalized authentication with primitives like “X said” and “Y believes”
  - Model checking (MMS98)
    - Build a FSM model of the system and enumerate states
- Difficult and time consuming but worth it if your protocol is important

Attack: similarly-named certs

- What if server has valid certificate but a similar name to another server?
  - Example: Microsoft vs. Micr0soft
- Outside the scope of SSL but relevant
- Used all the time with phishing emails
  - But few bother with SSL currently
  - Yellow lock JPEG on page sufficient
  - Now, please enter your PIN



Attack: side channel

- Side effects of handling secure data can be indirectly observed
- Example: timing attack on server’s private key [BB03]
  - Observe how long the server takes to return an error when sending invalid ClientKeyExchange
  - Bits of the key can slowly be discovered … over the Internet
- Tricky problem to be sure you’ve solved adequately

Conclusions

- SSL provides a well-tested secure transport layer
- Security protocols require careful interdependence of primitives
  - Privacy
  - Integrity protection
  - Authentication
- Easy to make mistakes designing security and crypto in particular

- This stuff is a lot of fun!

Recommended reading

- [TLS06] The Transport Layer Security (TLS) Protocol, Version 1.1. http://tools.ietf.org/html/rfc4346
- [Resc02] Rescarola, E. Introduction to OpenSSL programming. http://www.rtfm.com/openssl-examples/
- [WS96] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 Protocol. 1996. http://citeseer.ist.psu.edu/wagner96analysis.html
- [MMS98] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0. In Seventh USENIX Security Symposium, pages 201 - 216, 1998. http://citeseer.ist.psu.edu/mitchell98finitestate.html
- [BAN90] Burrows, M., Abadi, M., and Needham, R. M. "A Logic of Authentication", ACM Transactions on Computer Systems, Vol. 8, No. 1, Feb 1990, pp. 18 - 36. A Formal Semantics for Evaluating Cryptographic Protocols p 14. http://citeseer.ist.psu.edu/burrows90logic.htm
- [BB03] D. Boneh and D. Brumley. Remote Timing Attacks are Practical. Proceedings of the 12th USENIX Security Symposium, August 2003. http://citeseer.ist.psu.edu/article/boneh03remote.html

Fixing v2.0: downgrade attacks

- Backwards compatibility with insecure protocol is difficult
  - Active attacker could change ServerHello to advertise v2-only
- Clever solution: put 64 bits of 0x3 in the RSA padding
  - Attacker cannot substitute their own key without breaking the server cert
  - Luckily, SSL v2 only supported RSA key exchange