Appendix E. Backward Compatibility

E.1. Compatibility with TLS 1.0/1.1 and SSL 3.0

Since there are various versions of TLS (1.0, 1.1, 1.2, and any future versions) and SSL (2.0 and 3.0), means are needed to negotiate the specific protocol version to use. The TLS protocol provides a built-in mechanism for version negotiation so as not to bother other protocol components with the complexities of version selection.

TLS versions 1.0, 1.1, and 1.2, and SSL 3.0 are very similar, and use compatible ClientHello messages; thus, supporting all of them is relatively easy. Similarly, servers can easily handle clients trying to use future versions of TLS as long as the ClientHello format remains compatible, and the client supports the highest protocol version available in the server.

A TLS 1.2 client who wishes to negotiate with such older servers will send a normal TLS 1.2 ClientHello, containing { 3, 3 } (TLS 1.2) in ClientHello.client_version. If the server does not support this version, it will respond with a ServerHello containing an older version number. If the client agrees to use this version, the negotiation will proceed as appropriate for the negotiated protocol.

If the version chosen by the server is not supported by the client (or not acceptable), the client MUST send a "protocol_version" alert message and close the connection.

If a TLS server receives a ClientHello containing a version number greater than the highest version supported by the server, it MUST reply according to the highest version supported by the server.

A TLS server can also receive a ClientHello containing a version number smaller than the highest supported version. If the server wishes to negotiate with old clients, it will proceed as appropriate

for the highest version supported by the server that is not greater than ClientHello.client_version. For example, if the server supports TLS 1.0, 1.1, and 1.2, and client_version is TLS 1.0, the server will proceed with a TLS 1.0 ServerHello. If server supports (or is willing to use) only versions greater than client_version, it MUST send a "protocol_version" alert message and close the connection.

Whenever a client already knows the highest protocol version known to a server (for example, when resuming a session), it SHOULD initiate the connection in that native protocol.

Note: some server implementations are known to implement version negotiation incorrectly. For example, there are buggy TLS 1.0 servers that simply close the connection when the client offers a version newer than TLS 1.0. Also, it is

known that some servers will refuse the connection if any TLS extensions are included in ClientHello. Interoperability with such buggy servers is a complex topic beyond the scope of this document, and may require multiple connection attempts by the client.

Earlier versions of the TLS specification were not fully clear on what the record layer version number (TLSPlaintext.version) should contain when sending ClientHello (i.e., before it is known which version of the protocol will be employed). Thus, TLS servers compliant with this specification MUST accept any value {03,XX} as the record layer version number for ClientHello.

TLS clients that wish to negotiate with older servers MAY send any value {03,XX} as the record layer version number. Typical values would be {03,00}, the lowest version number supported by the client, and the value of ClientHello.client_version. No single value will guarantee interoperability with all old servers, but this is a complex topic beyond the scope of this document.

E.2. Compatibility with SSL 2.0

TLS 1.2 clients that wish to support SSL 2.0 servers MUST send version 2.0 CLIENT-HELLO messages defined in [SSL2]. The message MUST contain the same version number as would be used for ordinary ClientHello, and MUST encode the supported TLS cipher suites in the CIPHER-SPECS-DATA field as described below.

Warning: The ability to send version 2.0 CLIENT-HELLO messages will be phased out with all due haste, since the newer ClientHello format provides better mechanisms for moving to newer versions and negotiating extensions. TLS 1.2 clients SHOULD NOT support SSL 2.0.

However, even TLS servers that do not support SSL 2.0 MAY accept version 2.0 CLIENT-HELLO messages. The message is presented below in sufficient detail for TLS server implementors; the true definition is still assumed to be [SSL2].

For negotiation purposes, 2.0 CLIENT-HELLO is interpreted the same way as a ClientHello with a "null" compression method and no extensions. Note that this message MUST be sent directly on the wire, not wrapped as a TLS record. For the purposes of calculating Finished and CertificateVerify, the msg_length field is not considered to be a part of the handshake message.

```
uint8 V2CipherSpec[3];
struct {
    uint16 msg_length;
    uint8 msg_type;
    Version version;
    uint16 cipher_spec_length;
    uint16 session_id_length;
    uint16 challenge_length;
    V2CipherSpec cipher_specs[V2ClientHello.cipher_spec_length];
    opaque session_id[V2ClientHello.session_id_length];
```

```
      opaque challenge[V2ClientHello.challenge_length;
  } V2ClientHello;
```

msg_length The highest bit MUST be 1; the remaining bits contain the length of the following data in bytes.

msg_type This field, in conjunction with the version field, identifies a version 2 ClientHello message. The value MUST be 1.

version Equal to ClientHello.client_version.

cipher_spec_length This field is the total length of the field cipher_specs. It cannot be zero and MUST be a multiple of the V2CipherSpec length (3).

session_id_length This field MUST have a value of zero for a client that claims to support TLS 1.2.

challenge_length The length in bytes of the client's challenge to the server to authenticate itself. Historically, permissible values are between 16 and 32 bytes inclusive. When using the SSLv2 backward- compatible handshake the client SHOULD use a 32-byte challenge.

cipher_specs This is a list of all CipherSpecs the client is willing and able to use. In addition to the 2.0 cipher specs defined in [SSL2], this includes the TLS cipher suites normally sent in ClientHello.cipher_suites, with each cipher suite prefixed by a zero byte. For example, the TLS cipher suite {0x00,0x0A} would be sent as {0x00,0x00,0x0A}.

session_id This field MUST be empty.

challenge Corresponds to ClientHello.random. If the challenge length is less than 32, the TLS server will pad the data with leading (note: not trailing) zero bytes to make it 32 bytes long.

Note: Requests to resume a TLS session MUST use a TLS client hello.

E.3. Avoiding Man-in-the-Middle Version Rollback

When TLS clients fall back to Version 2.0 compatibility mode, they MUST use special PKCS#1 block formatting. This is done so that TLS servers will reject Version 2.0 sessions with TLS-capable clients.

When a client negotiates SSL 2.0 but also supports TLS, it MUST set the right-hand (least-significant) 8 random bytes of the PKCS padding (not including the terminal null of the padding) for the RSA encryption of the ENCRYPTED-KEY-DATA field of the CLIENT-MASTER-KEY to 0x03 (the other padding bytes are random).

When a TLS-capable server negotiates SSL 2.0 it SHOULD, after decrypting the ENCRYPTED-KEY-DATA field, check that these 8 padding bytes are 0x03. If they are not, the server SHOULD generate a random value for SECRET-KEY-DATA, and continue the handshake (which will eventually fail since the keys

will not match). Note that reporting the error situation to the client could make the server vulnerable to attacks described in [BLEI].