RFCDiff

闭源网络驱动的 RFC 文档匹配

谢远峰，王跃林，徐一洋

天津大学-智能与计算学部-网络安全

2024-09-09

1. 场景选择
2. 方法流程
3. RFC 状态机构建
4. 函数关联方法
5. 方法有效性统计
6. 函数规则匹配可行性实验
7. 函数优化的必要性

1. 场景选择

目标场景

- SSL/TLS 协议实现 RFC 合规性模糊测试
  - 协议：SSL(1.0,2.0,3.0)/TLS(1.0,1.2,1.3)-公钥加密
  - 目标：找出代码实现和 RFC 文档的不一致
    1. 协议的实现通常基于相应的 RFC（TLS1.3-RFC8446）
    2. RFC 中使用特定关键词（如"must"、"should"等）来描述实现协议时必须遵守的要求或事项
- motivation example:RFC 规则合规性样例说明
  1. > When multiple extensions of different types are present, the extensions MAY appear in any order, with the exception of ”pre shared key” (Section 4.2.11) which MUST be the last extension in the ClientHello (but can appear anywhere in the ServerHello extensions block). There MUST NOT be more than one extension of the same type in a given extension block.
  2. 处理 message 的 target: C:\Windows\System32\schannel.dll–CVE-2023-2823*
- 挑战
  - 测试用例构造：了解协议状态转换和消息序列的流程
  - 状态维护：确保接收方（被测试的协议实现）能够维持在期望的状态接收测试
  - 加密会话解析：不能仅依靠 RFC 生成会话消息，依赖于协议具体实现
  - 异常行为检测：违反 RFC 规则不会导致明显的错误，需要开发更加敏感和准确的方法来检测协议实现中的异常行为

函数分类

1. target: C:\Windows\System32\schannel.dll
2. 函数筛选与分类

- 总函数：1507
- CSSL3*:175（SSL3.0）
- CSSL*: 151（SSL1.0？/3.0 以下）
- CTLS*: 36 (TLS1.0？/1.3 版本以下)
- CTLS13*: 218 （TLS1.3）
- 工具类函数：927（未区分版本）

2. 选择对象：CTLS13*

- 按照消息类型(message type)和扩展(extension)分类

```
enum {
  client_hello(1),
  server_hello(2),
  new_session_ticket(4),
  end_of_early_data(5),
  encrypted_extensions(8),
  certificate(11),
  certificate_request(13),
  certificate_verify(15),
  finished(20),
  key_update(24),
  message_hash(254),
  (255)
} HandshakeType;
```

```
struct {
  HandshakeType msg_type;    /* handshake type */
  uint24 length;          /* remaining bytesin message */
  select (Handshake.msg_type) {
    case client_hello:          ClientHello;
    case server_hello:          ServerHello;
    case end_of_early_data:     EndOfEarlyData;
    case encrypted_extensions: EncryptedExtensions;
    case certificate_request:  CertificateRequest;
    case certificate:           Certificate;
    case certificate_verify:   CertificateVerify;
    case finished:              Finished;
    case new_session_ticket:   NewSessionTicket;
    case key_update:            KeyUpdate;
  };
} Handshake;
```

```
struct {
  ExtensionType extension_type;
  opaque extension_data<0..2^16-1>;
} Extension;

enum {
  server_name(0),                             /* RFC 6066 */
  max_fragment_length(1),                     /* RFC 6066 */
  status_request(5),                          /* RFC 6066 */
  supported_groups(10),                       /* RFC 8422, 79
  signature_algorithms(13),                   /* RFC 8446 */
  use_srtp(14),                               /* RFC 5764 */
  heartbeat(15),                              /* RFC 6520 */
  application_layer_protocol_negotiation(16), /* RFC 7301 */
  signed_certificate_timestamp(18),           /* RFC 6962 */
  client_certificate_type(19),                /* RFC 7250 */
  server_certificate_type(20),                /* RFC 7250 */
  padding(21),                                /* RFC 7685 */
  pre_shared_key(41),                         /* RFC 8446 */
  early_data(42),                             /* RFC 8446 */
  supported_versions(43),                     /* RFC 8446 */
  cookie(44),                                 /* RFC 8446 */
  psk_key_exchange_modes(45),                 /* RFC 8446 */
  certificate_authorities(47),                /* RFC 8446 */
  oid_filters(48),                            /* RFC 8446 */
  post_handshake_auth(49),                    /* RFC 8446 */
  signature_algorithms_cert(50),              /* RFC 8446 */
  key_share(51),                              /* RFC 8446 */
  (65535)
  } ExtensionType;
```

ServerContext_GenerateResponse 一级调用

- CTls13ServerContext::GenerateResponse(函数入口)
  （1435: GenerateResponse_CTls13ServerContext_Buffer）
  - **52 FinishCacheAddItem_CSessionCacheManager_SslContext**
  - **847 guard_xfg_dispatch_icall_nop**
  - **857 WPP**
  - **1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer**
  - **1431 GenerateHelloRetryRequest_CTls13ServerContext_Buffer**
  - **1432 GenerateNewSessionTicket_CTls13ServerContext_Buffer**
  - **1433 GeneratePostHandshakeCertificateRequest_CTls13ServerContext_Buffer**

GenerateClientHelloResponse

- CTls13ServerContext::GenerateResponse(函数入口)
  （1435: GenerateResponse_CTls13ServerContext_Buffer）
  - **1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer**
    1. **117 AllocateOutputBuffer_CSsl3TlsContext_Buffer;210 SetError_CSslContext_eSslErrorState**
    2. **572 security_check_cookie; 847 guard_xfg_dispatch_icall_nop;857 WPP; 858 WPP**
    3. **1299 InitUpdateClientHash_CSsl3TlsServerContext**
    4. 1327 ComputeCertificateMsgSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_slCredential
    5. 1329 ComputeCertificateRequestSize_CTls13ServerHandshake
    6. 1330 ComputeCertificateVerifySize_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient
    7. 1333 ComputeEncryptedExtensionsMsgSize_CTls13ServerHandshake
    8. 1335 ComputeFinishedSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer
    9. 1343 ComputeServerHelloOrHRRSize_CTls13ServerHandshake
    10. 1361 GenerateCertificateMsg_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer
    11. 1362 GenerateCertificateRequest_CTls13ServerHandshake
    12. 1364 GenerateCertificateVerify_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer
    13. 1369 GenerateEncryptedExtensionsMsg_CTls13ServerHandshake
    14. 1372 GenerateFinished_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer
    15. 1387 GenerateServerHelloOrHRR_CTls13ServerHandshake
    16. 1421 GenerateApplicationWriteKeys_CTls13Context_eSslErrorState
    17. 1423 GenerateCcsRecord_CTls13Context
    18. 1429 GenerateHandshakeWriteKeys_CTls13Context_eSslErrorState
    19. 1445 PopulateSecTrafficSecret_CTls13Context_TrafficType_CipherSuiteInfo
    20. **1452 SetCcsRecordInfo_CTls13Context_MessageInfo_CTlsRecord**
    21. **1614 imp_CertGetServerOcspResponseContext; 1758 imp_SslFreeObject**
    22. **1767 imp_GetLastError;1915 imp_RtlReleaseResource; 1917 imp_RtlAcquireResourceShared**

GenerateClientHelloResponse

- CTls13ServerContext::GenerateResponse(函数入口)
  （1435: GenerateResponse_CTls13ServerContext_Buffer）
  - **1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer**
    1. 1327 ComputeCertificateMsgSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_slCredential
    2. 1329 ComputeCertificateRequestSize_CTls13ServerHandshake
    3. 1330 ComputeCertificateVerifySize_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient
    4. 1333 ComputeEncryptedExtensionsMsgSize_CTls13ServerHandshake
    5. 1335 ComputeFinishedSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer
    6. 1343 ComputeServerHelloOrHRRSize_CTls13ServerHandshake
    7. **1361 GenerateCertificateMsg_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer**
    8. **1362 GenerateCertificateRequest_CTls13ServerHandshake**
    9. **1364 GenerateCertificateVerify_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer**
    10. **1369 GenerateEncryptedExtensionsMsg_CTls13ServerHandshake**
    11. **1372 GenerateFinished_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer**
    12. **1387 GenerateServerHelloOrHRR_CTls13ServerHandshake**
    13. **1421 GenerateApplicationWriteKeys_CTls13Context_eSslErrorState**
    14. **1423 GenerateCcsRecord_CTls13Context**
    15. **1429 GenerateHandshakeWriteKeys_CTls13Context_eSslErrorState**
    16. 1445 PopulateSecTrafficSecret_CTls13Context_TrafficType_CipherSuiteInfo
  - 1431 GenerateHelloRetryRequest_CTls13ServerContext_Buffer
  - 1432 GenerateNewSessionTicket_CTls13ServerContext_Buffer
  - 1433 GeneratePostHandshakeCertificateRequest_CTls13ServerContext_Buffer

GenerateHelloRetryRequest

- CTls13ServerContext::GenerateResponse(函数入口)
  （1435: GenerateResponse_CTls13ServerContext_Buffer）
  - 1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer
  - 1431 GenerateHelloRetryRequest_CTls13ServerContext_Buffer
    1. **117 AllocateOutputBuffer_CSsl3TlsContext_Buffer**
    2. **210 SetError_CSslContext_eSslErrorState**
    3. **572 security_check_cookie**
    4. **847 guard_xfg_dispatch_icall_nop**
    5. **1299 InitUpdateClientHash_CSsl3TlsServerContext**
    6. **1343 ComputeServerHelloOrHRRSize_CTls13ServerHandshake**
    7. **1387 GenerateServerHelloOrHRR_CTls13ServerHandshake**
    8. **1423 GenerateCcsRecord_CTls13Context**
    9. **1427 GenerateClientHelloHash_CTls13Context**
    10. **1452 SetCcsRecordInfo_CTls13Context_MessageInfo_CTlsRecord**
  - 1432 GenerateNewSessionTicket_CTls13ServerContext_Buffer
  - 1433 GeneratePostHandshakeCertificateRequest_CTls13ServerContext_Buffer

GenerateHelloRetryRequest

- CTls13ServerContext::GenerateResponse(函数入口)
  （1435: GenerateResponse_CTls13ServerContext_Buffer）
  - 1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer
  - 1431 GenerateHelloRetryRequest_CTls13ServerContext_Buffer
    1. **1343 ComputeServerHelloOrHRRSize_CTls13ServerHandshake**
    2. **1387 GenerateServerHelloOrHRR_CTls13ServerHandshake**
    3. **1427 GenerateClientHelloHash_CTls13Context**
  - 1432 GenerateNewSessionTicket_CTls13ServerContext_Buffer
  - 1433 GeneratePostHandshakeCertificateRequest_CTls13ServerContext_Buffer

GenerateNewSessionTicket

- CTls13ServerContext::GenerateResponse(函数入口)
  （1435: GenerateResponse_CTls13ServerContext_Buffer）
  - 1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer
  - 1431 GenerateHelloRetryRequest_CTls13ServerContext_Buffer
  - 1432 GenerateNewSessionTicket_CTls13ServerContext_Buffer
    1. **117 AllocateOutputBuffer_CSsl3TlsContext_Buffer**
    2. **210 SetError_CSslContext_eSslErrorState**
    3. **572 security_check_cookie**
    4. **847 guard_xfg_dispatch_icall_nop**
    5. **1340 ComputeNewSessionTicketSize_CTls13ServerHandshake**
    6. **1380 GenerateNewSessionTicket_CTls13ServerHandshake**
    7. **1445 PopulateSecTrafficSecret_CTls13Context_TrafficType_CipherSuiteInfo**
  - 1433 GeneratePostHandshakeCertificateRequest_CTls13ServerContext_Buffer

GenerateNewSessionTicket

- CTls13ServerContext::GenerateResponse(函数入口)
  （1435: GenerateResponse_CTls13ServerContext_Buffer）
  - 1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer
  - 1431 GenerateHelloRetryRequest_CTls13ServerContext_Buffer
  - 1432 GenerateNewSessionTicket_CTls13ServerContext_Buffer
    1. **1340 ComputeNewSessionTicketSize_CTls13ServerHandshake**
    2. **1380 GenerateNewSessionTicket_CTls13ServerHandshake**
    3. **1445 PopulateSecTrafficSecret_CTls13Context_TrafficType_CipherSuiteInfo**
  - 1433 GeneratePostHandshakeCertificateRequest_CTls13ServerContext_Buffer

GeneratePostHandshakeCertificateRequest

- CTls13ServerContext::GenerateResponse(函数入口)
  （1435: GenerateResponse_CTls13ServerContext_Buffer）
  - 1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer
  - 1431 GenerateHelloRetryRequest_CTls13ServerContext_Buffer
  - 1432 GenerateNewSessionTicket_CTls13ServerContext_Buffer
  - 1433 GeneratePostHandshakeCertificateRequest_CTls13ServerContext_Buffer
    1. **117 AllocateOutputBuffer_CSsl3TlsContext_Buffer**
    2. **210 SetError_CSslContext_eSslErrorState**
    3. **572 security_check_cookie**
    4. **847 guard_xfg_dispatch_icall_nop**
    5. **1329 ComputeCertificateRequestSize_CTls13ServerHandshake**
    6. **1362 GenerateCertificateRequest_CTls13ServerHandshake**
    7. **1425 GenerateCertificateRequestContext_CTls13ServerContext**

GeneratePostHandshakeCertificateRequest

- CTls13ServerContext::GenerateResponse(函数入口)
  （1435: GenerateResponse_CTls13ServerContext_Buffer）
  - 1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer
  - 1431 GenerateHelloRetryRequest_CTls13ServerContext_Buffer
  - 1432 GenerateNewSessionTicket_CTls13ServerContext_Buffer
  - 1433 GeneratePostHandshakeCertificateRequest_CTls13ServerContext_Buffer
    1. **1329 ComputeCertificateRequestSize_CTls13ServerHandshake**
    2. **1362 GenerateCertificateRequest_CTls13ServerHandshake**
    3. **1425 GenerateCertificateRequestContext_CTls13ServerContext**

ServerContext_GenerateResponse 二级调用

- CTls13ServerContext::GenerateResponse(函数入口)（1435: GenerateResponse_CTls13ServerContext_Buffer）
  - 1428 GenerateClientHelloResponse_CTls13ServerContext_Buffer
    1. 1327 ComputeCertificateMsgSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_slCredential
    2. 1329 ComputeCertificateRequestSize_CTls13ServerHandshake
    3. 1330 ComputeCertificateVerifySize_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient
    4. 1333 ComputeEncryptedExtensionsMsgSize_CTls13ServerHandshake
    5. 1335 ComputeFinishedSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer
    6. 1343 ComputeServerHelloOrHRRSize_CTls13ServerHandshake
    7. **1361 GenerateCertificateMsg_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer**
    8. **1362 GenerateCertificateRequest_CTls13ServerHandshake**
    9. **1364 GenerateCertificateVerify_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer**
    10. **1369 GenerateEncryptedExtensionsMsg_CTls13ServerHandshake**
    11. **1372 GenerateFinished_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer**
    12. **1387 GenerateServerHelloOrHRR_CTls13ServerHandshake**
    13. **1421 GenerateApplicationWriteKeys_CTls13Context_eSslErrorState**
    14. **1423 GenerateCcsRecord_CTls13Context**
    15. **1429 GenerateHandshakeWriteKeys_CTls13Context_eSslErrorState**
    16. 1445 PopulateSecTrafficSecret_CTls13Context_TrafficType_CipherSuiteInfo
  - 1431 GenerateHelloRetryRequest_CTls13ServerContext_Buffer
    1. **1343 ComputeServerHelloOrHRRSize_CTls13ServerHandshake**
    2. **1387 GenerateServerHelloOrHRR_CTls13ServerHandshake**
    3. **1427 GenerateClientHelloHash_CTls13Context**
  - 1432 GenerateNewSessionTicket_CTls13ServerContext_Buffer
    1. **1340 ComputeNewSessionTicketSize_CTls13ServerHandshake**
    2. **1380 GenerateNewSessionTicket_CTls13ServerHandshake**
    3. **1445 PopulateSecTrafficSecret_CTls13Context_TrafficType_CipherSuiteInfo**
  - 1433 GeneratePostHandshakeCertificateRequest_CTls13ServerContext_Buffer
    1. **1329 ComputeCertificateRequestSize_CTls13ServerHandshake**
    2. **1362 GenerateCertificateRequest_CTls13ServerHandshake**
    3. **1425 GenerateCertificateRequestContext_CTls13ServerContext**

函数筛选与分类

1. 分类：函数名

- 内存操作类：
  <`vector **deleting** destructor'>,
  <**~CTls13Handshake**<CTls13ServerContext,CTls13ExtServer>>
- 属性计算类：
  <**Compute**CertificateEntryExtensions**Size**> <**Compute**CertificateEntry**Size**>
  <**Compute**CertificateList**Size**> <**Compute**CertificateMsg**Size**>
  <**Compute**Finished**Size**> <**Compute**GenericExtensions**Size**>
- 消息生成类：
  <**Generate**CertificateEntry> <**Generate**CertificateEntryExtensions>
  <**Generate**CertificateList> <**Generate**CertificateMsg> <**Generate**CertificateVerify>
  <**Generate**CertVerifySignature> <**Generate**Finished> <**Generate**GenericExtensions>
  <**Generate**HandshakeHeader> <**Generate**KeyShareEntry>
  <**Generate**SignatureAlgorithmsExtension>
- 消息解析类：
  <**Parse**CertificateMsg> <**Parse**CertificateVerify> <**Parse**Finished> <**Parse**KeyUpdate>
- 工具类：
  <DetermineCertVerifySignatureAlgorithm>(Parse 函数调用)

2. 方法流程

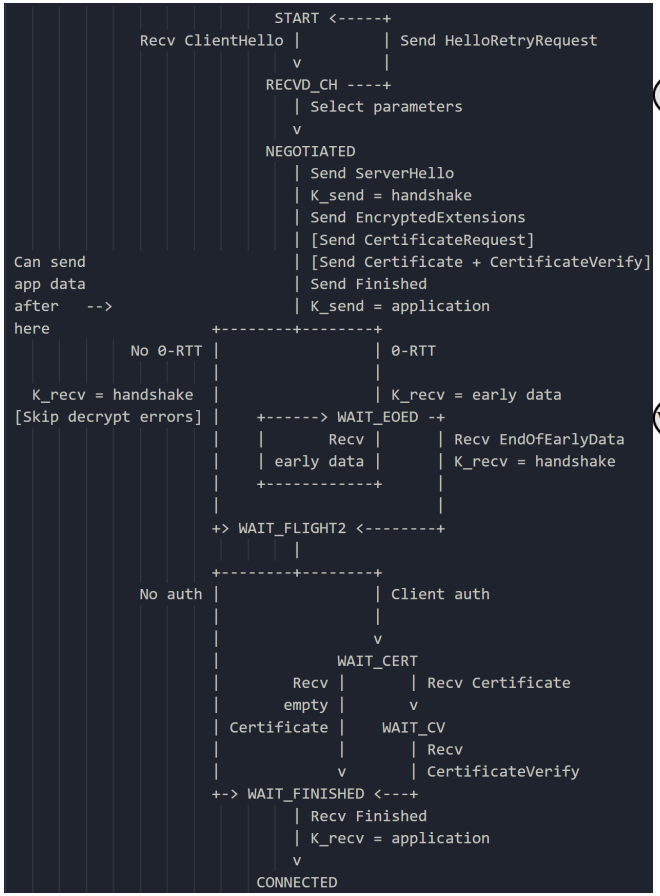
网络协议测试挑战

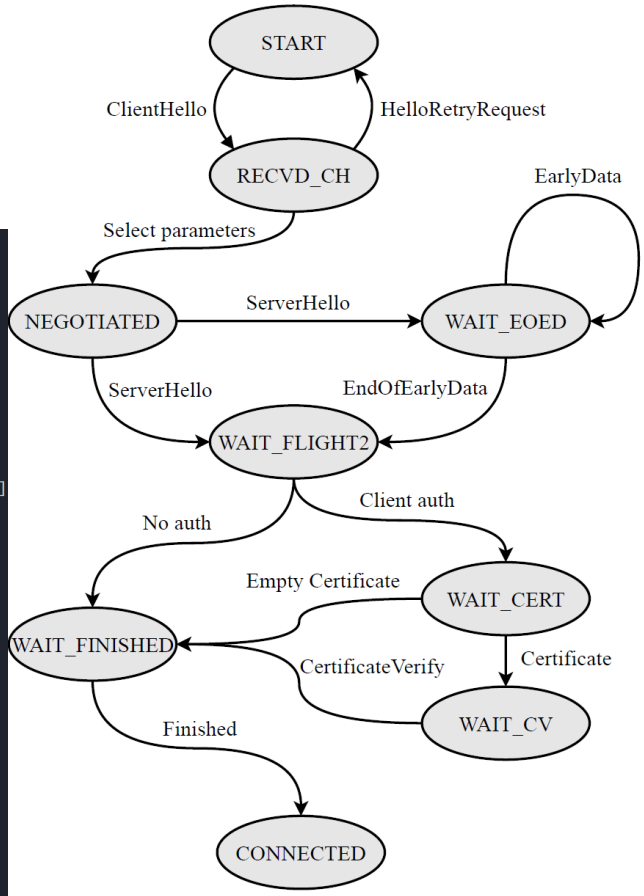
- 历史工作
  - RFC 规则抽取构建状态机
    - RFCNLP(22)：BERT 学习 RFC，构建 FSM，在代码表示上进行验证和路径探索，找到攻击点
    - ProtocolGPT(24)：LLM 逆向推断网络协议开源实现的 FSM（text-embeding, rag）
    - PROSPER(23)：LLM 实现自动化网络协议分析，基于提示词构建 FSM
  - 利用状态机构建文件进行测试
    - HDiff(22)：ABNF 文法解析 HTTP，差分测试
    - AFLNET(22):针对协议的灰盒模糊测试器，服务器状态反馈指导，充当客户端发送消息变体
    - ChatAFL(24)：LLM 抽取协议结构，填充内容，提高源码代码覆盖度
    - SADT(20)：模糊测试 SSL/TLS 中的 X.509 证书验证部分，差分测试
- survey 统一框架：
  1. protocol syntax acquisition and modeling（协议语法获取与建模）
  2. test case generation（测试样例生成）
  3. test execution and monitoring（测试执行与状态检测）
  4. feedback information acquisition and utilization（反馈信息获取与利用）
- network protocol fuzzing unique challenge（网络协议模糊测试特有挑战）
  1. Reliance On Network Links(依赖于网络链接)
     - 挑战描述：网络协议处理的网络流量是随着时间依次到达的，数据包是这些流量中的最小单位。模糊测试工具需要构建数据包并通过网络接口将其传输到协议软件。由于依赖网络链接，协议模糊测试工具必须能够使用标准网络接口来发送和接收网络数据包。
     - 工具现状：大多数现代协议模糊测试工具都具备发送和接收网络数据包的能力。

3. RFC 状态机构建

TLS1.3 协议状态机构建

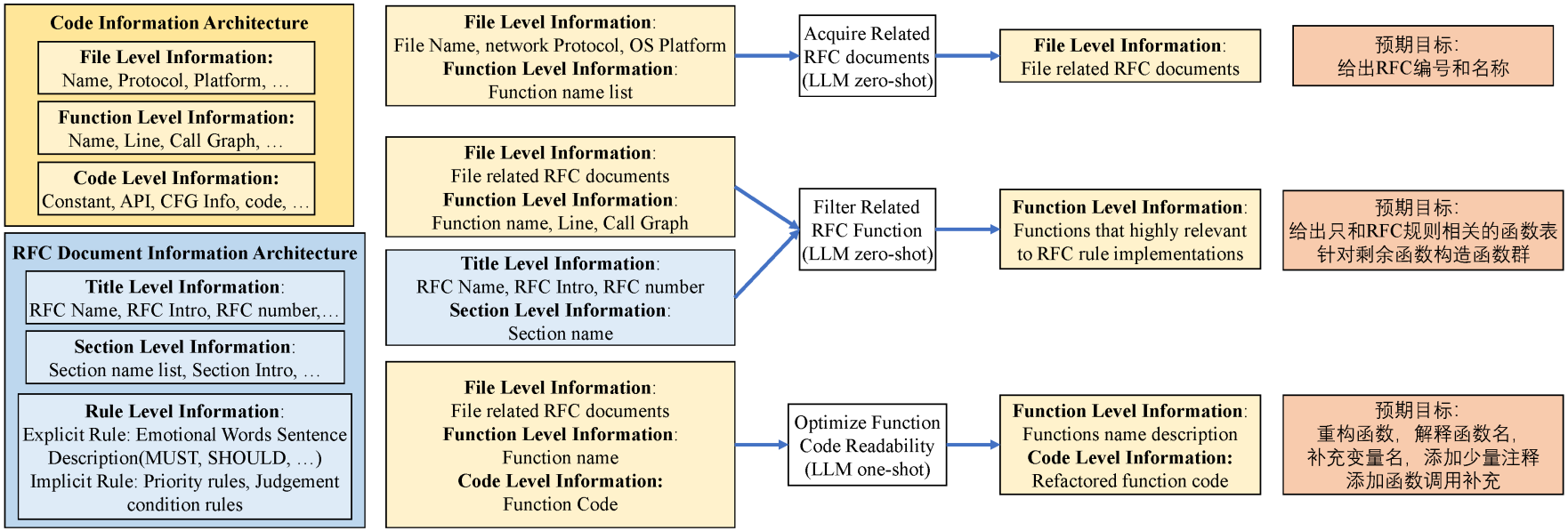
- 前提条件 1：Regular Protocol- constraint-enhanced regular expressions（http,udp,tls）
- 前提条件 2：Already have state machine description (tls)
- 方法
  - 输入：state machine description + Prompt
  - 输出：output code in C style
  - 输入：C style state machine code description
  - 输出：state machine figure
- 问题
  - 状态对应输入，处理，状态转换
  - 结构简单，输入约束，输出约束
- 补充内容
  1. 状态转换图（节点和边）
  2. 数据包结构
  3. server/client 状态编码
- 参考工作：
  -





4. 函数关联方法

方法流程：RFC 映射，函数过滤，函数重构



1. 代码信息架构
   - 文件层信息：文件名，涉及协议，所在 OS 平台
   - 函数层信息：函数名，函数大小，调用图
   - 代码层信息：函数体，常量，控制流信息
2. RFC 文档信息架构
   - 标题层信息：RFC 编号，标题，简介
   - 章节层信息：章节编号，标题
   - 规则层信息：显式规则，隐式规则，文本描述
3. 映射预处理：
   1. 文件 RFC 文档映射：利用文件逆向工程导出的函数列表知道 RFC 文档编号生成
   2. 函数过滤：过滤出和 RFC 规则强相关的函数，并划定函数群
   3. 函数重构：扩展函数名解释，补充变量名，添加少量注释，添加函数调用图

方法流程：RFC 映射

You are an expert in network protocols and Windows driver analysis. Given a list of function names obtained through reverse engineering of Windows platform driver files, your task is to identify the relevant network protocols and provide their corresponding RFC (Request for Comments) numbers and titles.

Example input:
1. TCP_SendPacket
2. HTTP_GET_Request
3. UDP_ReceiveData

Example output:
1. Protocol: TCP; RFC: 793 Title: Transmission Control Protocol
2. Protocol: HTTP; RFC: 2616 Title: Hypertext Transfer Protocol – HTTP/1.1
3. Protocol: UDP; RFC: 768 Title: User Datagram Protocol

Your task:

Analyze the provided list of function names. Internally, generate 5 separate outputs, each time identifying the network protocols based on these function names.

For each identified protocol in each internal output, provide:
1. The name of the network protocol
2. The corresponding RFC number
3. The exact title of the RFC document

Compare the results across all 5 internal outputs. In your final response, only include protocols that appear in at least 3 out of the 5 internal outputs.

Important constraints:

1. Only include protocols that clearly match the function names in the input list.
2. Do not provide explanations, reasoning, or any additional commentary.
3. If a protocol is identified but you're unsure of the RFC details, omit that protocol entirely.
4. Do not include any protocols or RFCs that aren't directly evidenced by the function names.
5. Provide only the requested information in the specified format.
6. Do not show any intermediate outputs or processes, only the final consensus result.
7. Your response should only contain the final output in the format specified in the example output.

Now, proceed with the analysis of the input function list and provide only the final consensus output (protocols appearing in at least 3 out of 5 internal outputs) in the specified format.<ADD function list FILE>

Protocol: TLS (Transport Layer Security) RFC: 8446 Title: The Transport Layer Security (TLS) Protocol Version 1.3

Protocol: SSL (Secure Sockets Layer) RFC: 6101 Title: The Secure Sockets Layer (SSL) Protocol Version 3.0

Protocol: DTLS (Datagram Transport Layer Security) RFC: 6347 Title: Datagram Transport Layer Security Version 1.2

Protocol: OCSP (Online Certificate Status Protocol) RFC: 6960 Title: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP

Protocol: SRTP (Secure Real-time Transport Protocol) RFC: 3711 Title: The Secure Real-time Transport Protocol (SRTP)

方法流程：函数过滤（相关性筛选）

```rust
// 只看 tls1.3 版本的相关函数
fn is_useful_node(node: &str, node_labels: &HashMap<String, String>) -> bool {
    if let Ok(num) = node.parse::<u32>() {
        if num <= 1565 &&
           num > 347 &&
           num != 572 &&
           num != 847 &&
           num != 1299
           && num != 1423 {
            if let Some(label) = node_labels.get(node) {
                let lower_label = label.to_lowercase();
                lower_label.contains("tls13")
                    && (lower_label.contains("server")
                        || lower_label.contains("client")
                        || lower_label.contains("context"))
                    && !lower_label.contains("delete")
                    && !lower_label.contains("wpp")
                    && !lower_label.contains("initialize")
            } else {
                false
            }
        } else {
            false
        }
    } else {
        false
    }
}
```

过滤条件：TLS1.3，非 windows 系统表函数，标明身份（server,client），与语言特性无关（init, delete）

You are an expert in network protocols and Windows driver analysis, with specific knowledge of RFC 8446 (The Transport Layer Security (TLS) Protocol Version 1.3). Your task is to analyze a list of function names obtained through reverse engineering of Windows platform driver files and identify which functions are related to RFC 8446.

To assist you in this task, here's some relevant information retrieved from RFC 8446:

1. Key concepts: TLS 1.3, handshake protocol, key exchange, authentication, record protocol
2. Important operations: encryption, decryption, digital signatures, key derivation
3. Specific algorithms: ECDHE, PSK, HKDF, AEAD

Now, let's walk through the thought process for identifying relevant functions:

1. Look for functions that contain keywords related to TLS, handshake, or cryptographic operations.
2. Consider functions that might handle protocol-specific data structures or messages.
3. Identify functions that could be involved in implementing the TLS 1.3 state machine.

Here are a few examples to guide your analysis:

Example 1
Function name: InitializeTLS13Handshake
Thought process: This function likely initializes the TLS 1.3 handshake process, which is a core part of RFC 8446.
Result: Related to RFC 8446

Example 2:
Function name: ProcessTCP_IP_Packet
Thought process: While this function deals with network protocols, it's not specific to TLS 1.3 or the operations defined in RFC 8446.
Result: Not related to RFC 8446

Example 3:
Function name: PerformECDHE_KeyExchange
Thought process: ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) is a key exchange method used in TLS 1.3, as specified in RFC 8446.
Result: Related to RFC 8446

Now, given a list of function names, please analyze each one using a similar thought process. Only output the functions that you determine are related to RFC 8446, along with a brief explanation of why you believe they are related.

List of functions to analyze:

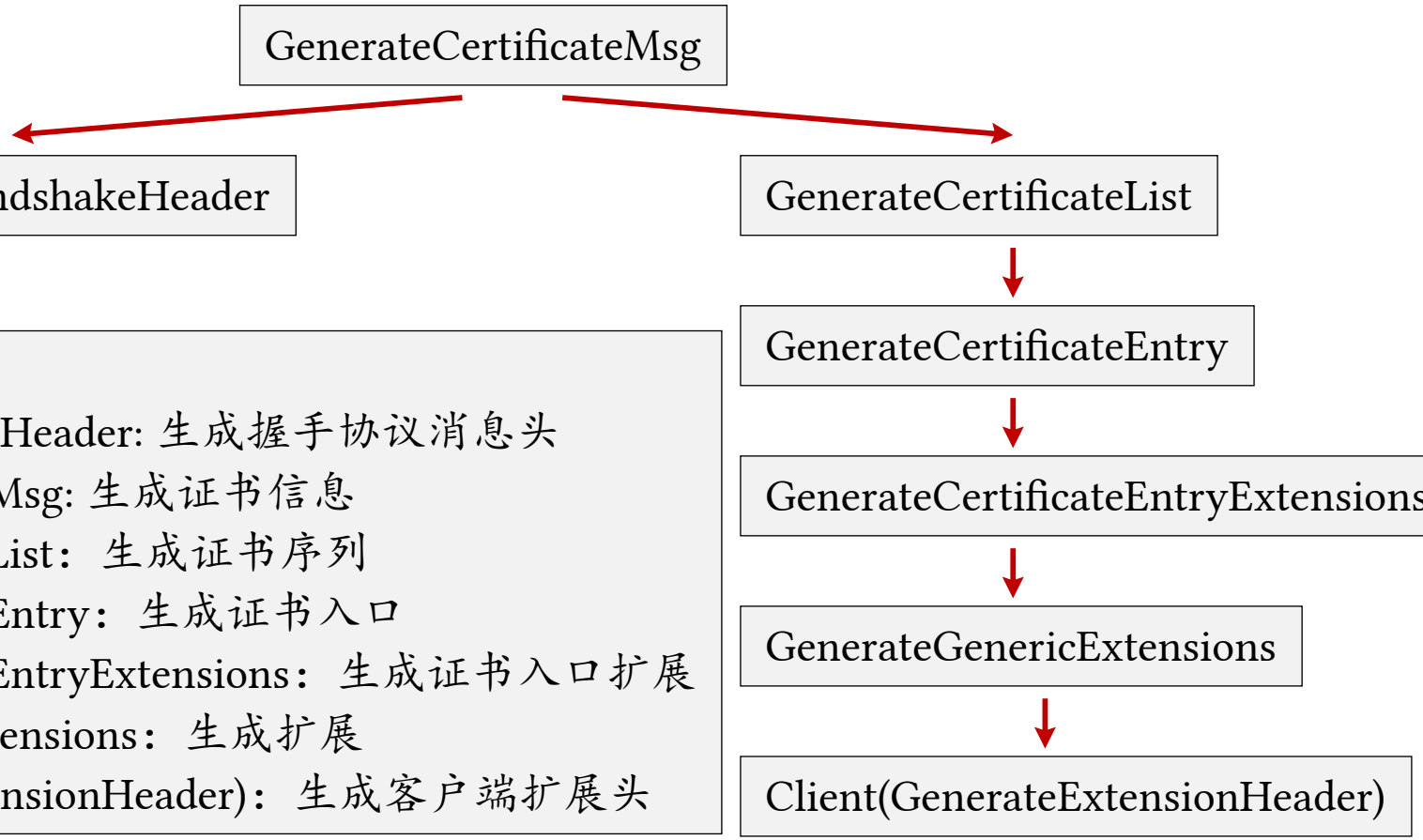
方法流程：函数过滤（家族分类）

Useful Call Graph for root node 1435 (filtered by blacklist criteria):

1435 (GenerateResponse_CTls13ServerContext_Buffer)
  1428 (GenerateClientHelloResponse_CTls13ServerContext_Buffer)
    1327 (ComputeCertificateMsgSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_slCredential)
      1325 (ComputeCertificateListSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)
        1323 (ComputeCertificateEntrySize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)
          1321 (ComputeCertificateEntryExtensionsSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)
            1337 (ComputeGenericExtensionsSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_eTlsHandshakeType)
    1329 (ComputeCertificateRequestSize_CTls13ServerHandshake)
      1328 (ComputeCertificateRequestExtensionsSize_CTls13ServerHandshake)
        1319 (ComputeCertificateAuthoritiesExtensionSize_CTls13ServerHandshake)
  1330 (ComputeCertificateVerifySize_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient)
    1318 (ComputeCertVerifySignatureSize_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient)
    1347 (DetermineCertVerifyCodePoint_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient)
      1348 (DetermineCertVerifySignatureAlgorithm_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)
  1333 (ComputeEncryptedExtensionsMsgSize_CTls13ServerHandshake)
    1334 (ComputeEncryptedExtensionsSize_CTls13ServerHandshake)
  1335 (ComputeFinishedSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)
  1343 (ComputeServerHelloOrHRRSize_CTls13ServerHandshake)
    1342 (ComputeServerHelloExtensionsSize_CTls13ServerHandshake)
      1338 (ComputeKeyShareEntrySize_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient)
  1361 (GenerateCertificateMsg_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)
    1359 (GenerateCertificateList_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)
      1355 (GenerateCertificateEntry_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)
        1357 (GenerateCertificateEntryExtensions_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)
          1374 (GenerateGenericExtensions_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_eTlsHandshakeType)
            1370 (GenerateExtensionHeader_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient_eTlsExtensions)
              1376 (GenerateHandshakeHeader_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_eTlsHandshakeType)

...

方法流程：函数过滤（家族分类）



函数名称简要说明

- GenerateHandshakeHeader: 生成握手协议消息头
- GenerateCertificateMsg: 生成证书信息
- GenerateCertificateList：生成证书序列
- GenerateCertificateEntry：生成证书入口
- GenerateCertificateEntryExtensions：生成证书入口扩展
- GenerateGenericExtensions：生成扩展
- Client(GenerateExtensionHeader)：生成客户端扩展头

相关 RFC8446 章节

- 4. Handshake Protocol：握手协议消息类型
- 4.4. Handshake Protocol_Authentication Messages 介绍握手协议中证书的作用
- 4.4.2. Authentication Messages_Certificate 介绍握手协议中证书消息
- 4.4.2.2. Certificate_Server Certificate Selection 介绍证书消息服务端格式
- Appendix C.2. Implementation Notes—Certificates and Authentication 证书完整性与安全

方法流程：函数优化

```
__int64 __fastcall CTls13Handshake<CTls13ServerContext,CTls13ExtServer>::GenerateCertificateList(__int64 a1)
{
  // ...var declaration

  v2 = *(_QWORD *)(a1 + 32);
  v14 = 0;
  result = (*(__int64 (__fastcall **)(__int64, __int64, char *))(*
  if ( !(_DWORD)result )
  {
    v4 = *(_QWORD *)(a1 + 32);
    v15 = __ROR2__(*(_WORD *)(a1 + 64), 8);
    result = (*(__int64 (__fastcall **)(__int64, __int64, __int16 *
    if ( !(_DWORD)result )
    {
      v5 = *(_QWORD *)(a1 + 72);
      if ( !v5 )
        return (*(_DWORD *)(*(_QWORD *)(a1 + 8) + 64i64) & 0x400515
      v6 = *(_QWORD *)(v5 + 64);
      if ( v6 )
      {
        v7 = *(__int64 **)(v6 + 16);
        if ( v7 )
        {
          // ...
}
```

```
/*
 * Naming Convention:
 * CTls13Handshake<CTls13ServerContext,CTls13ExtServer> - Template class for TLS 1.3 handsha
 * GenerateCertificateList - Function to generate a list of certificates
 * GenerateCertificateEntry - Helper function to generate a single certificate entry
 */

#include <stdint.h>
#include <stdbool.h>

int64_t __fastcall CTls13Handshake_GenerateCertificateList(int64_t context) {
    // ...var declaration

    // Get certificate type
    int64_t context_ptr = *(int64_t*)(context + 32);
    result = (*(int64_t (__fastcall **)(int64_t, int64_t, char*))(*(int64_t*)context_ptr + 3
    if (result) return result;

    // Get certificate request context
    context_ptr = *(int64_t*)(context + 32);
    certificate_request_context = __ROR2__(*(int16_t*)(context + 64), 8);
    result = (*(int64_t (__fastcall **)(int64_t, int64_t, int16_t*))(*(int64_t*)context_ptr
&certificate_request_context);
    if (result) return result;

    // ...
```

优化提示词：尝试还原一些函数中的信息

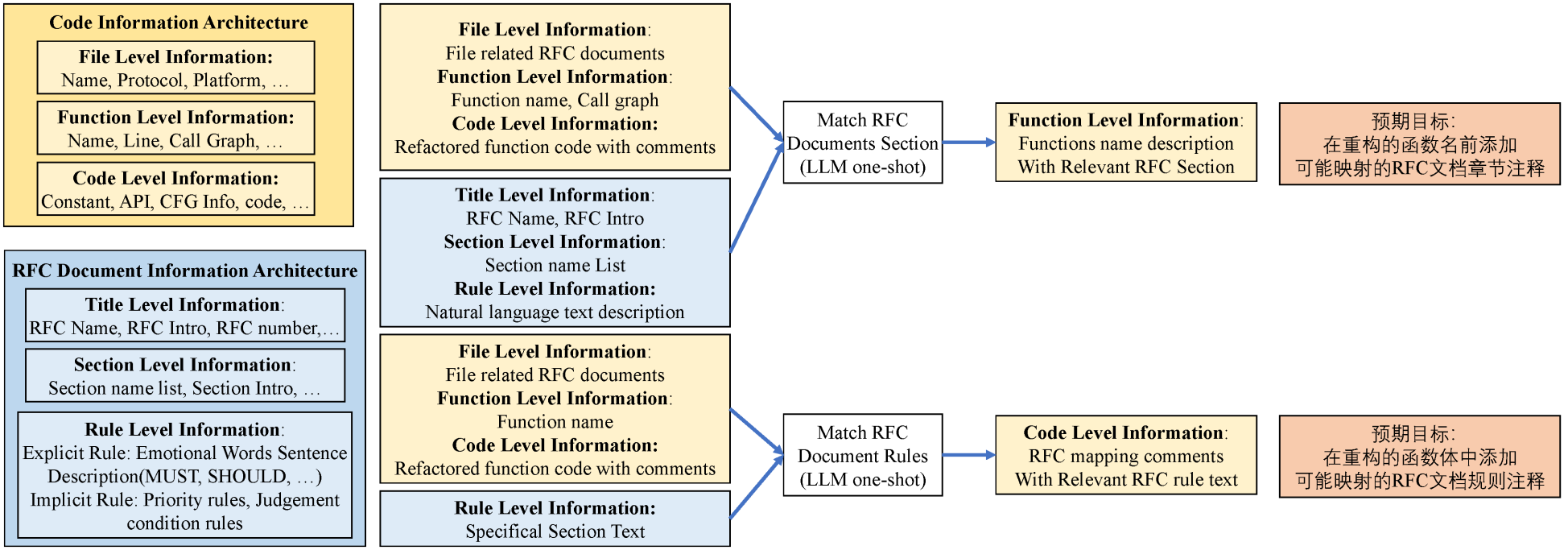
Refactor the provided C code to enhance readability and maintainability:

1. Improve formatting, indentation, and variable naming
2. Optimize for efficiency while preserving core logic
3. Add comments to explain complex sections
4. Include a header comment explaining the function naming convention（函数名称解释）

Constraints:

1. Do not split functions
2. Maintain one function for input and one for output
3. Preserve existing function names and parameter types
4. Strictly adhere to C language standards
5. Present only the refactored code in Markdown, with the naming convention explanation as a comment
6. Do not output any natural language descriptions other than comments in the code, including the opening (Here is ...)

方法流程：函数章节映射，函数规则映射



1. 代码信息架构
   - 文件层信息：文件名，涉及协议，所在 OS 平台
   - 函数层信息：函数名，函数大小，调用图
   - 代码层信息：函数体，常量，控制流信息
2. RFC 文档信息架构
   - 标题层信息：RFC 编号，标题，简介
   - 章节层信息：章节编号，标题
   - 规则层信息：显式规则，隐式规则，文本描述
3. RFC 规则映射
   - 函数 RFC 规则粗粒度映射：对应 RFC 文档章节编号和 title
   - 函数 RFC 规则细粒度映射：对应 RFC 文档指定章节内的规则

方法流程：函数规则粗粒度映射-提示词

章节匹配提示词：根据优化后代码和文档章节进行插入

You are an expert in C programming and RFC documentation analysis, specializing in TLS 1.3 (RFC 8446). Your task is to create a function that parses code and inserts detailed comments before each 'if' and 'while' statement.

1. Context Retrieval (RAG): Utilize your knowledge base to retrieve relevant sections from RFC 8446 for each 'if' and 'while' statement in the code.
2. Chain of Thought (CoT): For each retrieved RFC section, analyze its relevance to the code statement and formulate a clear explanation of the relationship.
3. Few-Shot Examples:

Here are three examples of properly formatted comments:

```
/* RFC 8446 Section 4.2.3 - Signature Algorithms*/
if (tls->signature_algorithms == NULL) {
    // Error handling code
}
/* RFC 8446 Section 4.4.1 - New Session Ticket Message*/
while (send_new_session_ticket && !finished_sent) {
    // Code to send NewSessionTicket
}
/* RFC 8446 Section 5.1 - Record Layer*/
if (record_type == HANDSHAKE || record_type == APPLICATION_DATA) {
    // Process recognized record types
}
```

4. Output Format: Provide only the complete code with inserted comments in Markdown format, without any additional natural language descriptions.
5. Consistency and Accuracy: Ensure that your analysis maintains consistency across all comments and accurately reflects both the code implementation and RFC 8446 specifications.

Each comment should:

1. Directly relate to RFC 8446 content
2. Include the relevant RFC 8446 section number

Constraints:

1. Ensure your analysis is thorough and accurate, leveraging your expertise in both C programming and RFC 8446 to provide insightful correlations between the code and the specification. Pay particular attention to areas where the implementation may differ from or extend beyond the RFC requirements.
2. Outputs only the complete code with comments inserted in Markdown.
3. Do not output any natural language descriptions other than comments in the code, including the opening (Here is ...).

// Code

1. file-1：RFC8446.txt
2. file-2: RFC8446-section-list.txt

方法流程：函数规则粗粒度映射

```
// ... same
    // ...var declaration

    /* RFC 8446 Section 4.4.2 - Certificate
       Determine the certificate type (X.509, RawPublicKey, etc.) */
    int64_t context_ptr = *(int64_t*)(context + 32);
    result = (*(int64_t (__fastcall **)(int64_t, int64_t, char*))
(*(int64_t*)context_ptr + 32))(context_ptr, 1, &certificate_type);
    if (result) return result;

    /* RFC 8446 Section 4.4.2 - Certificate
       Retrieve the certificate_request_context if this is in response
CertificateRequest */
    context_ptr = *(int64_t*)(context + 32);
    certificate_request_context = __ROR2__(*(int16_t*)(context + 64),
    result = (*(int64_t (__fastcall **)(int64_t, int64_t, int16_t*))
(*(int64_t*)context_ptr + 32))(context_ptr, 2, &certificate_request_co
    if (result) return result;

    /* RFC 8446 Section 4.4.2 - Certificate
       Check if a certificate chain exists */
    int64_t cert_chain = *(int64_t*)(context + 72);
    int64_t cert_chain_ptr = *(int64_t*)(cert_chain + 64);
    if (!cert_chain_ptr) return 87;

    certificate_chain = *(int64_t**)(cert_chain_ptr + 16);
    if (!certificate_chain || !*(uint32_t*)(cert_chain_ptr + 12)) retu
```

```
    /* RFC 8446 Section 4.4.2 - Certificate
       Iterate through the certificate chain */
    for (cert_index = 0; cert_index < *(uint32_t*)(cert_chain_entry +
{
        certificate = *(int64_t*)(*(int64_t*)(*(int64_t*)(cert_chain_
cert_index) + 8);
        if (!certificate) return 87;

        cert_blob = *(struct _CRYPTOAPI_BLOB**)(certificate + 24);
        if (!cert_blob) return 87;

        bool is_first_cert = (cert_index == 0);

        /* RFC 8446 Section 4.4.2.4 - Receiving a Certificate Message
           Skip non-matching certificates if not the first and flag is not set */
        if (!*(char*)(context + 80) && !is_first_cert) {
            if (!CertCompareCertificateName(*(uint32_t*)certificate, cert_blob + 3,
cert_blob + 5)) {
                continue;
```

```
    /* RFC 8446 Section 4.4.2 - Certificate
       Iterate through the certificate chain */
    for (cert_index = 0; cert_index < *(uint32_t*)(cert_chain_entry + 12); cert_index++)
{
        certificate = *(int64_t*)(*(int64_t*)(*(int64_t*)(cert_chain_entry + 16) + 8 *
cert_index) + 8);
        if (!certificate) return 87;

        cert_blob = *(struct _CRYPTOAPI_BLOB**)(certificate + 24);
        if (!cert_blob) return 87;

        bool is_first_cert = (cert_index == 0);

        /* RFC 8446 Section 4.4.2.4 - Receiving a Certificate Message
        Skip non-matching certificates if not the first and flag is not set */
        if (!*(char*)(context + 80) && !is_first_cert) {
            if (!CertCompareCertificateName(*(uint32_t*)certificate, cert_blob + 3,
cert_blob + 5)) {
                continue;
            }
        }
        result = CTls13Handshake_GenerateCertificateEntry(context, certificate,
is_first_cert);
        if (result) return result;
    }
    if (!cert_chain) {
        /* RFC 8446 Section 4.4.2.1 - OCSP Status and SCT Extensions
           Handle the case where no suitable certificate is available */
        return (*(uint32_t*)(*(int64_t*)(context + 8) + 64) & 0x40051555) != 0 ? 0x54F :
0;
    }
    int64_t cert_chain_ptr = *(int64_t*)(cert_chain + 64);
    if (!cert_chain_ptr) return 87;

    certificate_chain = *(int64_t**)(cert_chain_ptr + 16);
    if (!certificate_chain || !*(uint32_t*)(cert_chain_ptr + 12)) return 87;

    cert_chain_entry = *certificate_chain;
    if (!cert_chain_entry || !*(int64_t*)(cert_chain_entry + 16) || !*(uint32_t*)
(cert_chain_entry + 12)) return 87;
```

章节映射：LLM few-shot + rag + cot

方法流程：函数规则细粒度映射-提示词

You are an expert in C programming and RFC documentation analysis, specializing in TLS 1.3 (**RFC 8446 SECTION 4.4.2**). Your task is to create a function that parses C code and inserts detailed comments before each 'if' and 'while' statement according to existing comments. Use the following approaches:

1. RAG (Retrieval-Augmented Generation):
   - Retrieve relevant sections from RFC 8446, focusing on **SECTION 4.4.2.**
   - Use this information to augment your knowledge when creating comments.
2. COT (Chain of Thought):
   - For each 'if' and 'while' statement, follow this thought process:
     1. Identify the purpose of the statement in the context of TLS 1.3.
     2. Locate the corresponding section in RFC 8446.
     3. Analyze how the code implements the RFC specifications.
     4. Note any discrepancies or extensions beyond the RFC requirements.
3. Few-Shot Learning: Here are two examples of properly formatted comments:

   Example 1:

```
/* RFC 8446 Section 4.4.2: Certificate Verify
 * "The CertificateVerify message is used to provide explicit proof that an
 * endpoint possesses the private key corresponding to its certificate."
 * This code verifies the signature in the CertificateVerify message.
 */
if (ssl->handshake->certificate_verify == NULL) { /* Code to handle missing CertificateVerify message*/}
```

   Example 2:

```
/* RFC 8446 Section 4.4.2: Certificate Verify
 * "If the CertificateVerify message is sent by a server, the signature
 * algorithm MUST be one offered in the client's "signature_algorithms" extension"
 * This loop checks if the signature algorithm is supported by the client.
 */
while (p < end) {/*Code to iterate through supported signature algorithms*/}
```

4. Each comment should:
   1. Directly relate to RFC 8446 content
   2. Include the relevant RFC 8446 section number
   3. Provide a quote or paraphrase from the RFC text
   4. Explain how the code implements or relates to the RFC specifications
   5. Note any discrepancies between the code and RFC requirements, if applicable
5. Constraints:
   1. Ensure your analysis is thorough and accurate, leveraging your expertise in both C programming and RFC 8446 to provide insightful correlations between the code and the specification.
   2. Output only the complete code with comments inserted in Markdown.
   3. Do not output any natural language descriptions other than comments in the code.

方法流程：函数规则细粒度映射 Part-1

```
// same output
    // ...var declaration

    // Get certificate type
    int64_t context_ptr = *(int64_t*)(context + 32);
    result = (*(int64_t (__fastcall **)(int64_t, int64_t, char*))(*(int64_t*)context_ptr + 32))(context_ptr, 1, &certificate_type);

    /* RFC 8446 Section 4.4.2.2:
     * "The certificate type MUST be X.509v3 [RFC5280], unless explicitly
     * negotiated otherwise (e.g., [RFC7250])."
     * This check ensures the certificate type is valid as per RFC 8446.
     */
    if (result) return result;

    // Get certificate request context
    context_ptr = *(int64_t*)(context + 32);
    certificate_request_context = __ROR2__(*(int16_t*)(context + 64), 8);
    result = (*(int64_t (__fastcall **)(int64_t, int64_t, int16_t*))(*(int64_t*)context_p
&certificate_request_context);

    /* RFC 8446 Section 4.4.2:
     * "If this message is in response to a CertificateRequest, the value of
     * certificate_request_context in that message. Otherwise (in the case of server
     * authentication), this field SHALL be zero length."
     * This check ensures the certificate request context is properly set.
     */
    if (result) return result;

    // Check if certificate chain exists
    int64_t cert_chain = *(int64_t*)(context + 72);

    /* RFC 8446 Section 4.4.2:
     * "The server's certificate_list MUST always be non-empty."
     * This check ensures that the server's certificate chain is not empty,
     * as required by the RFC. If it is empty, it returns based on context flags.
     */
    if (!cert_chain) {
        // Return based on context flags
        return (*(uint32_t*)(*(int64_t*)(context + 8) + 64) & 0x40051555) != 0 ? 0x54F : 0;
    }
```

Section: 4.4.2.2.
Handshake Protocol_Authentication Messages_Certificate_Server Certificate Selection

- The certificate type MUST be X.509v3 [RFC5280], unless explicitly negotiated otherwise (e.g., [RFC7250]).

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

```
struct {
  select (certificate_type) {
    case RawPublicKey:
      /* From RFC 7250 ASN.1_subjectPublicKeyInfo */
      opaque ASN1_subjectPublicKeyInfo<1..2^24-1>;
    case X509:
      opaque cert_data<1..2^24-1>;
  };
  Extension extensions<0..2^16-1>;
} CertificateEntry;
```

certificate_request_context: If this message is in response to a CertificateRequest, the value of certificate_request_context in that message. Otherwise (in the case of server authentication), this field SHALL be zero length.

The server's certificate_list MUST always be non-empty.

方法流程：函数规则细粒度映射 Part-2

```
int64_t cert_chain_ptr = *(int64_t*)(cert_chain + 64);

/* RFC 8446 Section 4.4.2:
 * "Each following certificate SHOULD directly certify the one immediately preceding it."
 * This check ensures the certificate chain is properly structured.
 */
if (!cert_chain_ptr) return 87;

certificate_chain = *(int64_t**)(cert_chain_ptr + 16);

/* RFC 8446 Section 4.4.2:
 * "The sender's certificate MUST come in the first CertificateEntry in the list."
 * This check ensures the certificate chain is not empty and contains at least one entry.
 */
if (!certificate_chain || !*(uint32_t*)(cert_chain_ptr + 12)) return 87;

cert_chain_entry = *certificate_chain;

/* RFC 8446 Section 4.4.2:
 * "Each CertificateEntry contains a single certificate and set of extensions."
 * This check ensures the first certificate entry in the chain is valid.
 */
if (!cert_chain_entry || !*(int64_t*)(cert_chain_entry + 16) || !*(uint32_t*)(cert_chain_entry + 12)) return 87;

// Iterate through certificates
for (cert_index = 0; cert_index < *(uint32_t*)(cert_chain_entry + 12); cert_index++) {
    certificate = *(int64_t*)(*(int64_t*)(*(int64_t*)(cert_chain_entry + 16) + 8 * cert_index) + 8);

    /* RFC 8446 Section 4.4.2:
     * "The certificate_list MUST NOT be empty. If the server has no certificates
     * to send, it MUST send a Certificate message containing no certificates."
     * This check ensures each certificate in the chain is valid.
     */
    if (!certificate) return 87;

    cert_blob = *(struct _CRYPTOAPI_BLOB**)(certificate + 24);

    /* RFC 8446 Section 4.4.2:
     * "The certificate_list MUST NOT be empty. If the server has no certificates
     * to send, it MUST send a Certificate message containing no certificates."
     * This check ensures the certificate blob is valid.
     */
    if (!cert_blob) return 87;
```

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

If the corresponding certificate type extension (“server_certificate_type” or “client_certificate_type”) was not negotiated in EncryptedExtensions, or the X.509 certificate type was negotiated, then each CertificateEntry contains a DER-encoded X.509 certificate. **The sender’s certificate MUST come in the first CertificateEntry in the list. Each following certificate SHOULD directly certify the one immediately preceding it.** Because certificate validation requires that trust anchors be distributed independently, a certificate t anchor MAY be omitted from the chain, provided that supported peers are known to possess any omitted certificates.

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

certificate_list: A sequence (chain) of CertificateEntry structures, **each containing a single certificate and set of extensions.**

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

The server’s certificate_list MUST always be non-empty. A client will send an empty certificate_list if it does not have an appropriate certificate to send in response to the server’s authentication request.

方法流程：函数规则细粒度映射 Part-3

```
    bool is_first_cert = (cert_index == 0);

    /* RFC 8446 Section 4.4.2.2:
     * "The following rules apply to certificates sent by the server:"
     * This check implements the certificate selection rules specified in the RFC,
     * particularly for non-first certificates in the chain.
     */
    if (!*(char*)(context + 80) && !is_first_cert) {
        if (!CertCompareCertificateName(*(uint32_t*)certificate, cert_blob + 3, cer
            continue;
        }
    }

    result = CTls13Handshake_GenerateCertificateEntry(context, certificate, is_firs

    /* RFC 8446 Section 4.4.2:
     * "If the server cannot produce a certificate chain that is signed only
     * via the indicated supported algorithms, then it SHOULD continue the
     * handshake by sending the client a certificate chain of its choice
     * that may include algorithms that are not known to be supported by the
     * client."
     * This check ensures each certificate entry is generated successfully.
     */
    if (result) return result;
}

return 0;
}
```

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

If the server cannot produce a certificate chain that is signed only via the indicated supported algorithms, then it SHOULD continue the handshake by sending the client a certificate chain of its choice that may include algorithms that are not known to be supported by the client. This fallback chain SHOULD NOT use the deprecated SHA-1 hash algorithm in general, but MAY do so if the client's advertisement permits it, and MUST NOT do so otherwise.

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

Note: Prior to TLS 1.3, "certificate_list" ordering required each certificate to certify the one immediately preceding it; however, some implementations allowed some flexibility. Servers sometimes send both a current and deprecated intermediate for transitional purposes, and others are simply configured incorrectly, but these cases can nonetheless be validated properly. **For maximum compatibility, all implementations SHOULD be prepared to handle potentially extraneous certificates and arbitrary orderings from any TLS version, with the exception of the end-entity certificate which MUST be first.**

Section: 4.4.2.2.
Handshake Protocol_Authentication Messages_Certificate_Server Certificate Selection

The following rules apply to the certificates sent by the server:

- The certificate type MUST be X.509v3 [RFC5280], unless explicitly negotiated otherwise (e.g., [RFC7250]).
- The server's end-entity certificate's public key (and associated restrictions) MUST be compatible with the selected authentication algorithm from the client's "signature_algorithms" extension (currently RSA, ECDSA, or EdDSA).
- The certificate MUST allow the key to be used for signing (i.e., the digitalSignature bit MUST be set if the Key Usage extension is present) with a signature scheme indicated in the client's "signature_algorithms"/"signature_algorithms_cert" extensions (see Section 4.2.3).
- The "server_name" [RFC6066] and "certificate_authorities" extensions are used to guide certificate selection. As servers MAY require the presence of the "server_name" extension, clients SHOULD send this extension, when applicable.

5. 方法有效性统计

Partial Useful Call Graph for root node 1435 (filtered by blacklist criteria):

1435 (GenerateResponse_CTls13ServerContext_Buffer)<75>
  1428 (GenerateClientHelloResponse_CTls13ServerContext_Buffer)<390>
    1327 (ComputeCertificateMsgSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_slCredential)<38>
      1325 (ComputeCertificateListSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)<72>
        1323 (ComputeCertificateEntrySize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)<37>
          1321 (ComputeCertificateEntryExtensionsSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)<36>
            1337 (ComputeGenericExtensionsSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_eTlsHandshakeType)<46>
    1329 (ComputeCertificateRequestSize_CTls13ServerHandshake)<73>
      1328 (ComputeCertificateRequestExtensionsSize_CTls13ServerHandshake)<40>
        1319 (ComputeCertificateAuthoritiesExtensionSize_CTls13ServerHandshake)<19>
  1330 (ComputeCertificateVerifySize_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient)<40>
    1318 (ComputeCertVerifySignatureSize_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient)<76>
    1347 (DetermineCertVerifyCodePoint_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient)<47>
      1348 (DetermineCertVerifySignatureAlgorithm_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)<30>
  1333 (ComputeEncryptedExtensionsMsgSize_CTls13ServerHandshake)<27>
    1334 (ComputeEncryptedExtensionsSize_CTls13ServerHandshake)<38>
  1335 (ComputeFinishedSize_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)<26>
  1343 (ComputeServerHelloOrHRRSize_CTls13ServerHandshake)<46>
    1342 (ComputeServerHelloExtensionsSize_CTls13ServerHandshake)<50>
      1338 (ComputeKeyShareEntrySize_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient)<112>
  1361 (GenerateCertificateMsg_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)<36>
    1359 (GenerateCertificateList_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)<84>
      1355 (GenerateCertificateEntry_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)<67>
        1357 (GenerateCertificateEntryExtensions_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer)<28>
          1374 (GenerateGenericExtensions_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_eTlsHandshakeType)<50>
            1370 (GenerateExtensionHeader_CTls13Handshake_VCTls13ClientContext_Tls13ExtClient_eTlsExtensions)<23>
              1376 (GenerateHandshakeHeader_CTls13Handshake_VCTls13ServerContext_Tls13ExtServer_eTlsHandshakeType)<34>
...

6. 函数规则匹配可行性实验

GenerateCertificateMsg 函数映射

Code Summerization in Natural Language(自然语言代码总结):
This function **generates the Certificate message** as described in **Section 4.4.2 of RFC 8446.** It includes the **certificate_request_context** (which should be zero-length for server authentication), followed by the **certificate_list**. The certificate_list is generated in a separate function call to **GenerateCertificateList**.

The function also handles the case where a non-zero certificate_request_context is provided, which would be used for external client authentication mechanisms, though this is not typical for server certificates in TLS 1.3.

```
__int64 GenerateCertificateMsg(__int64 a1){
  // ...
  // Generate the handshake header for the Certificate message (type 11)
  result = GenerateHandshakeHeader(a1, 11, *(_WORD *)(a1 + 136));
  if (!(_DWORD)result ){
    // Determine if certificate_request_context should be included
    // As per Section 4.4.2: "The certificate_request_context MUST be zero length
    // unless an external client authentication mechanism is used"
    // ...
    // Write the certificate_request_context
    result = /**/;
    if (!(_DWORD)result ){
      if (!v6){goto LABEL_10;}
      // Check if a certificate is available when certificate_request_context is non-zero
      if ( !*(_QWORD *)(*(_QWORD *)(a1 + 8) + 1224i64) )
        return 2148074244i64; // Error if no certificate available
      // Write empty extension for the certificate_request_context
      result = /**/;
      if (!(_DWORD)result ){
LABEL_10:
        // Generate the certificate_list as described in Section 4.4.2
        result = GenerateCertificateList(a1);
        if (!(_DWORD)result )
          // Finalize the Certificate message
          return /**/;
      }
    }
  }
  return result; // error return result
}
// notes: all comments explaination generated by LLM(gpt4o/claude3.5-sonnet)
```

Section: 4. Handshake Protocol

The handshake protocol is used to negotiate the security parameters of a connection. Handshake messages are supplied to the TLS record layer, where they are encapsulated within one or more TLSPlaintext or TLSCiphertext structures which are processed and transmitted as specified by the current active connection state.

```
enum {
  // ...
  certificate(11),
  certificate_request(13),
  certificate_verify(15),
  // ...
  (255)
} HandshakeType;
```

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

This message conveys the endpoint's certificate chain to the peer.

...

```
struct {
  opaque certificate_request_context<0..2^8-1>;
  CertificateEntry certificate_list<0..2^24-1>;
} Certificate;
```

certificate_request_context: If this message is in response to a CertificateRequest, the value of certificate_request_context in that message. Otherwise (in the case of server authentication), this field SHALL be zero length.

certificate_list: A sequence (chain) of CertificateEntry structures, each containing a single certificate and set of extensions.

提示词框架:
Input File:
RFC8446 preprocessed doc and section info doc
Input Prompt:
orginal reverse engine Code +
Generating Summarization and Parsing function that inserts detailed comments into the code based on the contents of the provided RFC8446 documentation and RFC8446 documentation section info, requiring the comments and summarization to be strongly related to the contents of the RFC documentation. Make sure that each comment added has an RFC documentation section info reference.

GenerateHandshakeHeader 函数


```
__int64 GenerateHandshakeHeader(// Generate a TLS 1.3 handshake message header
  __int64 a1,          // this pointer
  char a2,             // HandshakeType (Section 4)
  unsigned __int16 a3 // length of the handshake message body
  ){
  // ...
  // Check if the length is at least 4 bytes (minimum handshake message size)
  // Section 4: "Protocol messages MUST be sent in the order defined in Section 4.4.1"
  if ( a3 < 4u ){return 87i64;} // ERROR_INVALID_PARAMETER
  v6 = *(_QWORD *)(a1 + 32);
  v10 = a2;

  // Write the HandshakeType (1 byte)
  // Section 4: "struct { HandshakeType msg_type; ... } Handshake;"
  result = (*(__int64 (__fastcall **)(__int64, __int64, char *))
              (*(_QWORD *)v6 + 32i64))(v6, 1i64, &v10);;
  if (!(_DWORD)result ){
    // Write the first byte of the length field (always 0 for TLS 1.3)
    // Section 4: "uint24 length; /* remaining bytes in message */"
    v7 = *(_QWORD *)(a1 + 32);
    v11 = 0;
    result = (*(__int64 (__fastcall **)(__int64, __int64, char *))
                (*(_QWORD *)v7 + 32i64))(v7, 1i64, &v11);
    if ( !(_DWORD)result ){
      // Write the remaining 2 bytes of the length field
      // Section 4: "uint24 length; /* remaining bytes in message */"
      v8 = *(_QWORD )(a1 + 32);
      v9[0] = ROR2(a3 - 4, 8);  // Convert to big-endian and subtract header size
      return /**/;
    }
  }
  return result;
}
// notes: all comments explaination generated by LLM(gpt4o/claude3.5-sonnet)
```

Section: 4. Handshake Protocol

```
struct {
  HandshakeType msg_type;    /* handshake type */
  uint24 length;             /* remaining bytes in mess*/
  select (Handshake.msg_type) {
      // ...
      case certificate:           Certificate;
      // ...
      case key_update:            KeyUpdate;
  };
} Handshake;
```

Protocol messages MUST be sent in the order defined in **Section 4.4.1** and shown in the diagrams in **Section 2**. **A peer which receives a handshake message unexpected order MUST abort the handshake with an "unexpected_message"**

New handshake message types are assigned by IANA as described in Section 11.

Section: 4.4.1. Handshake Protocol_Authentication Messages_The Transcript Hash

For concreteness, the transcript hash is always taken from the following sequence of handshake messages, **starting at the first ClientHello and including only those messages that were sent**: ClientHello, HelloRetryRequest, ClientHello, ServerHello, EncryptedExtensions, server CertificateRequest, server Certificate, server CertificateVerify, server Finished, EndOfEarlyData, client Certificate, client CertificateVerify, client Finished.

Section: 2.1. Incorrect DHE Share

If the client has not provided a sufficient "key_share" extension(e.g., it includes only DHE or ECDHE groups unacceptable to or unsupported by the server), the server corrects the mismatch with a HelloRetryRequest and the client needs to restart the handshake with an appropriate "key_share" extension, as shown in Figure 2. **If no common cryptographic parameters can be negotiated, the server MUST abort the handshake with an appropriate alert.**

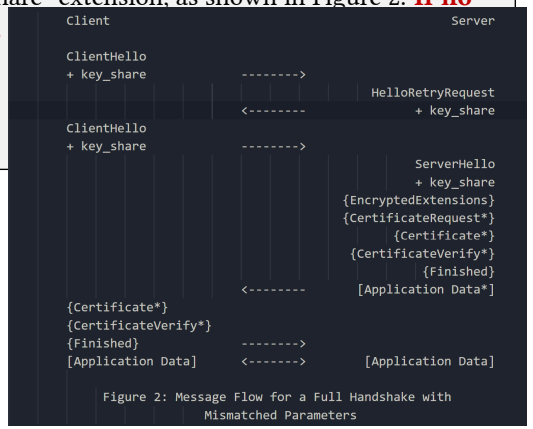


Figure 2: Message Flow for a Full Handshake with Mismatched Parameters

Code Summerization in Natural Language(自然语言代码总结):

This function **generates the header for a TLS 1.3 handshake message**.

The header consists of:

1. 1-byte message type
2. 3-byte length field (1 zero byte + 2 bytes for actual length)

The function follows the structure defined in RFC 8446 Section 4 for handshake messages. It ensures the minimum message length is respected and properly formats the type and length fields as per the specification.

7. 函数优化的必要性

GenerateCertificateList 函数映射


```
// This function generates the certificate list for a TLS 1.3 server handshake
// as described in RFC 8446 Section 4.4.2 (Certificate)
__int64 GenerateCertificateList(__int64 a1){
  // ... (variable declarations)
  // Write certificate_request_context (should be empty for server)
  // RFC 8446 Section 4.4.2: "In TLS 1.3,
  // the server's certificate_request_context MUST be zero length"
  v14 = 0;
  result = /**/;
  if ( !(_DWORD)result )
  {
    // Write certificate_list length
    v15 = __ROR2__(*(_WORD *)(a1 + 64), 8);
    result = /**/;
    if ( !(_DWORD)result )
    {
      // ... Server certificate selection logic
      // Process certificate chain RFC 8446 Section 4.4.2: "
      // The sender's certificate MUST come in the first CertificateEntry in the list"
      v6 = *(_QWORD *)(v5 + 64);
      if ( v6 ){
        // ...
        while ( 1 ){
          // Get certificate entry
          v10 = /**/;
          // ...
          // RFC 8446 Section 4.4.2.2: Check if we need to compare certificate names
          if ( *(_BYTE *)(a1 + 80) )
            goto LABEL_17;
          // ...
          // Generate certificate entry
          result = GenerateCertificateEntry(a1,v10, v12);
          // ...
        }
        // Additional certificate name checking (possibly related to SNI)
        if (CertCompareCertificateName(*(_DWORD *)v10, v11 +3, v11 + 5) )
          return 0i64;
      }
  // If no certificate chain, use single certificate
  // ...
  result = GenerateCertificateEntry(a1, v13, 1);
  // ...
}
// notes: all comments explaination generated by LLM(gpt4o/claude3.5-sonnet)
```

Code Summerization in Natural Language(自然语言代码总结):
This function implements the generation of the Certificate message for a TLS 1.3 server, as described in **RFC 8446 Section 4.4.2.** It writes **the certificate_request_context (which must be empty for a server) and the certificate_list length.** Then it processes the server's certificate chain, **ensuring that the server's end-entity certificate comes first in the list.** For each certificate in the chain, it calls GenerateCertificateEntry to create the corresponding CertificateEntry structure. If there's no certificate chain, it falls back to using a single certificate. The function adheres to the structure and requirements specified in the RFC for the Certificate message in TLS 1.3.

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

certificate_request_context: If this message is in response to a CertificateRequest, the value of certificate_request_context in that message. Otherwise (in the case of server authentication), this field SHALL be zero length.

If the corresponding certificate type extension ("server_certificate_type" or "client_certificate_type") was not negotiated in EncryptedExtensions, or the X.509 certificate type was negotiated, then each CertificateEntry contains a DER-encoded X.509 certificate. **The sender's certificate MUST come in the first CertificateEntry in the list.** Each following certificate SHOULD directly certify the one immediately preceding it. Because certificate validation requires that trust anchors be distributed independently, a certificate that specifies a trust anchor MAY be omitted from the chain, provided that supported peers are known to possess any omitted certificates.

Note: **Prior to TLS 1.3, "certificate_list" ordering required each certificate to certify the one immediately preceding it;** however, some implementations allowed some flexibility. Servers sometimes send both a current and deprecated intermediate for transitional purposes, and others are simply configured incorrectly, but these cases can nonetheless be validated properly. For maximum compatibility, **all implementations SHOULD be prepared to handle potentially extraneous certificates and arbitrary orderings from any TLS version, with the exception of the end-entity certificate which MUST be first.**

If the RawPublicKey certificate type was negotiated, then **the certificate_list MUST contain no more than one CertificateEntry, which contains an ASN1_subjectPublicKeyInfo value** as defined in [RFC7250], Section 3

The OpenPGP certificate type [RFC6091] MUST NOT be used with TLS 1.3.

The server's certificate_list MUST always be non-empty.

GenerateCertificateList 函数映射

```
// This function generates the certificate list for a TLS 1.3 server handshake
// as described in RFC 8446 Section 4.4.2 (Certificate)
__int64 GenerateCertificateList(__int64 a1){
  // ... (variable declarations)
  // Write certificate_request_context (should be empty for server)
  // RFC 8446 Section 4.4.2: "In TLS 1.3,
  // the server's certificate_request_context MUST be zero length"
  v14 = 0;
  result = /**/;
  if ( !(_DWORD)result )
  {
    // Write certificate_list length
    v15 = __ROR2__(*(_WORD *)(a1 + 64), 8);
    result = /**/;
    if ( !(_DWORD)result )
    {
      // ... Server certificate selection logic
      // Process certificate chain RFC 8446 Section 4.4.2: "
      // The sender's certificate MUST come in the first CertificateEntry in the list"
      v6 = *(_QWORD *)(v5 + 64);
      if ( v6 ){
        // ...
        while ( 1 ){
          // Get certificate entry
          v10 = /**/;
          // ...
          // RFC 8446 Section 4.4.2.2: Check if we need to compare certificate names
          if ( *(_BYTE *)(a1 + 80) )
            goto LABEL_17;
          // ...
          // Generate certificate entry
          result = GenerateCertificateEntry(a1,v10, v12);
          // ...
        }
        // Additional certificate name checking (possibly related to SNI)
        if (CertCompareCertificateName(*(_DWORD *)v10, v11 +3, v11 + 5) )
          return 0i64;
      }
  // If no certificate chain, use single certificate
  // ...
  result = GenerateCertificateEntry(a1, v13, 1);
  // ...
}
// notes: all comments explaination generated by LLM(gpt4o/claude3.5-sonnet)
```

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

certificate_request_context: If this message is in response to a CertificateRequest, the value of certificate_request_context in that message. Otherwise (in the case of server authentication), this field SHALL be zero length.

If the corresponding certificate type extension (“server_certificate_type” or “client _certificate_type”) was not negotiated in EncryptedExtensions, or the X.509 certificate type was negotiated, then each CertificateEntry contains a DER-encoded X.509 certificate. **The sender’s certificate MUST come in the first CertificateEntry in the list.** Each following certificate SHOULD directly certify the one immediately preceding it. Because certificate validation requires that trust anchors be distributed independently, a certificate that specifies a trust anchor MAY be omitted from the chain, provided that supported peers are known to possess any omitted certificates.

If the RawPublicKey certificate type was negotiated, then **the certificate_list MUST contain no more than one CertificateEntry, which contains an ASN1_ subjectPublicKeyInfo value** as defined in [RFC7250], Section 3.

The OpenPGP certificate type [RFC6091] MUST NOT be used with TLS 1.3.

The server’s certificate_list MUST always be non-empty.

Section: 4.4.2.2.
Handshake Protocol_Authentication Messages_Certificate_Server Certificate Selection

- The certificate type MUST be X.509v3 [RFC5280], unless explicitly negotiated otherwise (e.g., [RFC7250]).
- **The “server_name” [RFC6066] and “certificate_authorities” extensions are used to guide certificate selection.** As servers MAY require the presence of the “server_name” extension, clients SHOULD send this extension, when applicable.

If the server cannot produce a certificate chain that is signed only， via the indicated su client a certificate chain of its choice that may include algorithms that are not known to SHA-1 hash algorithm in general, but MAY do so if the client’s advertisement permits it,

If the server has multiple certificates, it chooses one of them based on the above-mentio configuration, and preferences).

COMPLECATED， NOT IN DETAIL AND NOT ENOUGH !!!
ONLY ENSURE RELATED TO SECTION4.4.2 AND SECTION4.4.2.2

GenerateCertificateEntry 函数映射

```
// This function generates a CertificateEntry as described in RFC 8446 Section 4.4.2
__int64 GenerateCertificateEntry(__int64 a1, __int64 a2, char a3){
  // ... (variable declarations)
  // Initialize certificate data length and pointer
  // RFC 8446 Section 4.4.2:
  // "CertificateEntry contains a single certificate and set of extensions"
  v5 = 0;
  v6 = 0i64;
  v7 = 65530i64;
  if ( a2 ){
    // Check if certificate data length is within allowed range
    // RFC 8446 Section 4.4.2: "opaque cert_data<1..2^24-1>;"
    if ( *(_DWORD *)(a2 + 16) > 0xFFFAu ) return 87i64;
    // ...
  }
  // Handle extensions if required
  // RFC 8446 Section 4.4.2: "Extension extensions<0..2^16-1>;"
  if ( a3 ){
    v7 = 65530 - (unsigned int)v5;
    if ( (int)v7 < *(unsigned __int16 *)(a1 + 154) ) return 87i64;
  }

  // Write certificate_type (always 0 for X.509) RFC 8446 Section 4.4.2:
  // "enum { X509(0), RawPublicKey(2), (255) } CertificateType;"
  v9 = *(_QWORD *)(a1 + 32);
  v13 = 0;
  result = /**/;
  if ( !(_DWORD)result ){
    // Write certificate data length
    // ...
    result = /**/;
    if ( !(_DWORD)result ){
      // Write certificate data
      if ( /**/ ){
        if ( a3 ){
          // Generate certificate extensions RFC 8446 Section 4.4.2
          return GenerateCertificateEntryExtensions(a1);
        }
        else{// Write empty extensions field
          // ...
          return /**/;
    // ...
}
```

Code Summerization in Natural Language(自然语言代码总结):
This function generates a CertificateEntry as defined in **RFC 8446 Section 4.4.2.** It handles the writing of **the certificate type (always X.509), certificate data length, and the certificate data itself.** If extensions are required, it calls a separate function to generate them. The function includes checks to ensure that the certificate data length is within the allowed range and that there's enough space for extensions if needed. This implementation closely follows the structure defined in the RFC, including **the handling of extensions and the format of the CertificateEntry.**

Section: 4.4.2. Handshake Protocol_Authentication Messages_Certificate

...

```
enum {X509(0),RawPublicKey(2),(255)} CertificateType;
struct {
  select (certificate_type) {
    case RawPublicKey:
      /* From RFC 7250 ASN1_subjectPublicKeyInfo */
      opaque ASN1_subjectPublicKeyInfo\<1.2^24-1>;
    case X509:
      opaque cert_data\<1..2^24-1>;
  };
  Extension extensions\<0..2^16-1>;
} CertificateEntry;

struct {
  opaque certificate_request_context\<0..2^8-1>;
  CertificateEntry certificate_list\<0..2^24-1>;
} Certificate;
```

certificate_list: A sequence (chain) of CertificateEntry structures, each containing a single certificate and set of extensions.

Section: 4.4.2.2. Handshake Protocol_Authentication Messages_Certificate_Server Certificate Selection

- The certificate type MUST be X.509v3 [RFC5280], unless explicitly negotiated otherwise (e.g., [RFC7250]).