ICS 35.030
CCS L 80

中 华 人 民 共 和 国 密 码 行 业 标 准

GM/T XXXXX—XXXX

证书透明规范

Certificate transparency specification

（草案）

在提交反馈意见时，请将您知道的相关专利连同支持性文件一并附上。

XXXX－XX－XX 发布　　XXXX－XX－XX 实施

国家密码管理局　　发 布

目　　次

前　言

本文件按照GB/T 1.1—2020《标准化工作导则　第1部分：标准化文件的结构和起草规则》的规定起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由密码行业标准化技术委员会提出并归口。

本文件起草单位：

本文件主要起草人：

引　言

商用密码SSL证书可实现网络通信传输加密，保护传输中的数据安全及服务器身份可信，是网络安全及信息安全中的重要密码产品。为了保障我国商用密码SSL证书应用安全，建立公开、透明的商用密码SSL证书透明体系已成为首要任务。

本文件的起草基于我国商用密码体系的算法标准，参考了证书透明国际标准RFC6962技术规范，明确了证书透明系统架构和运行规范。

证书透明技术国际标准RFC9162为RFC6962升级版本，但因技术原因并未实际应用，包括分片日志挑战哈希计算的兼容性问题,MTH2算法节点类型前缀(0x00/0x01)与RFC6962哈希方式冲突问题均未解决。为保障商用密码SSL证书透明规范的可行性，本文件的起草重点考虑了商用密码合规和应用生态实际情况，暂未参考RFC9162。后续RFC9162成为主流应用标准时，商用密码SSL证书透明规范标准可同步修订。

证书透明规范

1 范围

本文件规定了商用密码SSL数字证书透明的基本原理和技术规范，包括密码组件、日志系统、客户端消息的要求。

本文件适用于商用密码SSL数字证书应用中证书透明体系的建立、适用和安全管理。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中,注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 16262.1-2006 信息技术 抽象语法记法一（ASN.1）第一部分：基本记法规范

GB/T 32905 信息安全技术 SM3密码杂凑算法

GB/T 32918 （所有部分）信息安全技术 SM2椭圆曲线公钥密码算法

GM/T 0015-2023 数字证书格式

GM/Z 4001 密码术语

RFC 6962 Certificate Transparency

3 术语和定义

GB/T 38636、GM/Z 4001和RFC 6962界定的术语和定义适用于本文件。

4 缩略语

下列缩略语适用于本文件。

ASN.1:抽象语法标记（Abstract Syntax Notation One）

CA:证书认证机构（Certificate Authority）

CRL:证书撤销列表（Certificate Revocation List）

CT:证书透明（Certificate Transparency）

MMD:最大合并延迟（Maximum Merge Delay）

MTH:默克尔树杂凑值（Merkle Tree Hash）

RFC:请求意见稿（Request for Comments）

OID:对象标识符（Object ID）

PKI:公钥基础设施（Public Key Infrastructure）

RFC:请求意见稿（Request for Comments）

SCT:证书签发时间戳（Signed Certificate Timestamp）

SSL:安全套接层（Secure Sockets Layer）

STH:已签名树头（Signed Tree Head）
TLS:传输层安全性协议（Transport Layer Security）

5 基本原理

5.1 概述

证书透明（Certificate Transparency）是针对商用密码公共 TLS/SSL 服务器证书（简称 SSL 证书）的备案管理与公开监督系统，涉及以下实体。

——CA：向日志系统提交预签证书备案；

——日志系统运营方：运营管理证书透明日志系统，接收 CA 提交的预签证书备案信息，并返回证书签发时间戳数据（SCT）；

——浏览器厂商：浏览器验证 SSL 证书中是否有可信的 SCT 数据，并依据验证情况提示网站访问者 SSL 证书是否可信；

——第三方监督机构和审计机构：实时监督 CA 签发证书行为，防范恶意签发风险。

5.2 系统架构

在证书透明机制下，只有完成透明公示和备案的SSL证书才被信任。证书用户、第三方监督机构和审计机构均可获得CA签发SSL证书的全部备案信息，以便用户能及时要求CA撤销错误签发或恶意签发的SSL证书，防范应用安全风险。证书透明系统整体架构如下。

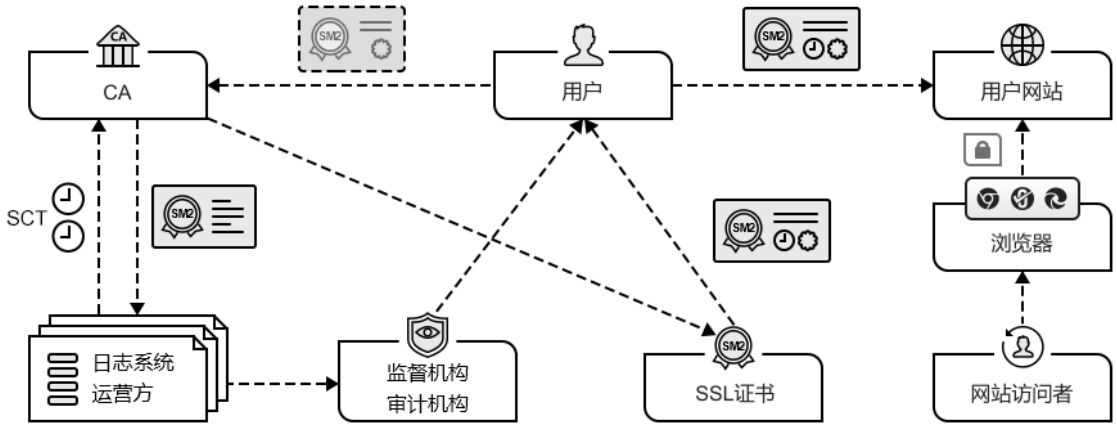

图1 证书透明系统架构图

5.3 运行流程

证书透明系统的运行包含以下环节。

a) 日志系统运营方公开信任的 CA 根证书列表；

b) 签发证书前，CA 向一个或多个日志系统提交预签证书到证书透明日志系统；

c) 针对信任列表内的 CA，日志系统运营方实时返回证书签发时间戳（SCT），作为已备案标识，并公示 CA 所有备案信息；

d) CA 将收到的 SCT 数据写入 SSL 证书，然后正式签发；

e) 网站部署CA签发的SSL证书；
f) 浏览器在用户访问网站时验证网站SSL证书，如无可信的SCT数据则提示“不安全”等警示信息。

6 密码组件要求

6.1 数据结构

6.1.1 默克尔树模型

日志系统数据结构采用二进制默克尔树模型，该模型为基于SM3算法（应符合GB/T 32905的要求）的二叉树形数据结构，包含叶节点、分支节点和根节点。

默克尔树模型数据运算方式为由下至上逐层进行杂凑运算，直至根节点。各节点数据规范如下。

a) 叶节点存储节点下数据的杂凑值；
b) 分支节点存储相邻两个叶节点的杂凑值；
c) 根节点存储相邻分支节点的杂凑值，可标识整个默克尔树的所有数据。

6.1.2 默克尔树构建方法

默克尔树模型构建方法为，输入数据条目列表，通过SM3算法计算消息摘要（杂凑值），生成叶节点（leaf）。输入n个有序列表，D[n] = {d（0），d（1），...，d（n-1）}。n不必为2的幂。

在默克尔树构建过程中，杂凑值（MTH）定义如下。

a) 列表为空时，默克尔树杂凑值为空字符串的杂凑值，定义为：

MTH（{}） = SM3（）

b) 列表为单条目时，默克尔树杂凑值定义为：

MTH（{d（0）}） = SM3（0x00 || d（0））

c) 列表为多条目时，n>1，令k为小于n的两个的最大幂（即，k<n <= 2k），n元素列表D[n]的默克尔树杂凑值递归定义为：

MTH（D[n]） = SM3（0x01 || MTH（D[0:k]） || MTH（D[k:n]））

式中：

||——串联；

D[k1:k2]——长度为（k2 - k1）的列表{d（k1），d（k1+1），...，d（k2-1）}。

6.2 审计路径

默克尔树叶节点审计路径为：计算从叶节点到根节点所需的节点列表，如计算出的根节点杂凑值与实际根节点杂凑值一致，则证明叶节点存在于默克尔树中。

输入n个有序列表，D[n] = {d（0），...，d（n-1）}，审计路径PATH（m，D[n]）对于第 （m+1）个输入d（m），0 <= m < n，定义如下：

a) 列表为单元素时，D[1] = {d（0）}的默克尔树叶节点审计路径为空：

PATH（0，{d（0）}） = {}

b) 列表为多元素时，n > 1，令k为小于n的两个中的最大幂。n > m元素列表中第（m+1）个元素d（m）的审计路径递归定义为：

PATH（m，D[n]） = PATH（m，D[0:k]） : MTH（D[k:n]） for m < k；

PATH（m，D[n]） = PATH（m - k，D[k:n]） : MTH（D[0:k]） for m >= k。

式中：

：——列表串联；

D[k1:k2]——长度（k2 - k1）列表{d（k1）, d（k1+1）,..., d（k2-1）}。

6.3 一致性验证

日志系统所有数据记录“仅可添加（append-only）”，不能更改，该属性可通过默克尔树的一致性予以验证。

本文件定义的默克尔树一致性验证方法如下：

杂凑值 MTH（D[n]）的默克尔一致性证明和前 m 个叶子的杂凑值 MTH（D[0:m]），m <= n，是默克尔树中的节点列表需要验证前 m 个输入 D[0:m] 在两棵树中是否相等。因此，应包含一组足以验证默克尔树杂凑值 MTH（D[n]）的中间节点（即对输入的承诺），这样的一个子集相同节点可用于验证 MTH（D[0 :m]）。输出唯一的最小一致性证明定义为：

输入 n 个有序列表，D[n] = {d（0）, ..., d（n-1）}，默克尔树一致性验证 PROOF（m, D[n]）对于先前的默克尔树杂凑值 MTH（D[0:m]），0 < m < n，定义为：

PROOF（m, D[n]） = SUBPROOF（m, D[n], true）

如 m 为最初请求 PROOF 的值，则 m = n 的子证明为空（表明默克尔树杂凑值 MTH（D[0:m]）已知）：

SUBPROOF（m, D[m], true） = {}

m = n 的子证明是默克尔树哈希提交输入 D[0:m]；否则：

SUBPROOF（m, D[m], false） = {MTH（D[m]）}

对于 m < n，令 k 为小于 n 的两个的最大幂。然后递归定义子证明。

如 m <= k，则右子树条目 D[k:n]仅存在于当前树中。证明左子树条目 D[0:k]是一致的，并向 D[k:n]添加承诺：

SUBPROOF（m, D[n], b） = SUBPROOF（m, D[0:k], b） : MTH（D[k:n]）

如 m > k，左子树条目 D[0:k]在两棵树中是相同的。证明右子树条目 D[k:n]是一致的，并向 D[0:k]添加承诺。

SUBPROOF（m, D[n], b） = SUBPROOF（m - k, D[k:n], false） : MTH（D[0:k]）

式中：

: ——列表串联；

D[k1:k2]——长度 （k2 - k1）列表{d（k1）, d（k1+1）,..., d（k2-1）}为前。结果证明中的节点数以 ceil（log2（n）） + 1 为界。

6.4 示例

6.4.1 数据结构示例

以 7 个叶节点的默克尔树为例，其数据结构如图 2 所示。

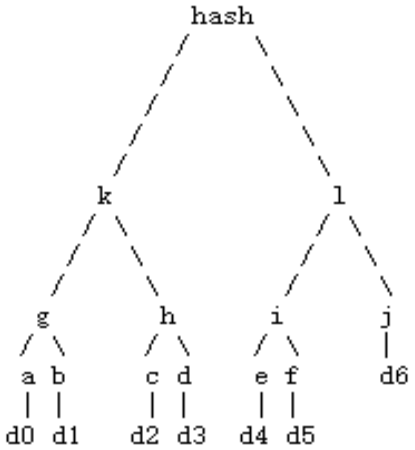

图2　默克尔树结构示意图

6.4.2　审计路径示例

以 7 个叶节点的默克尔树为例，其构建步骤如图 3 所示。

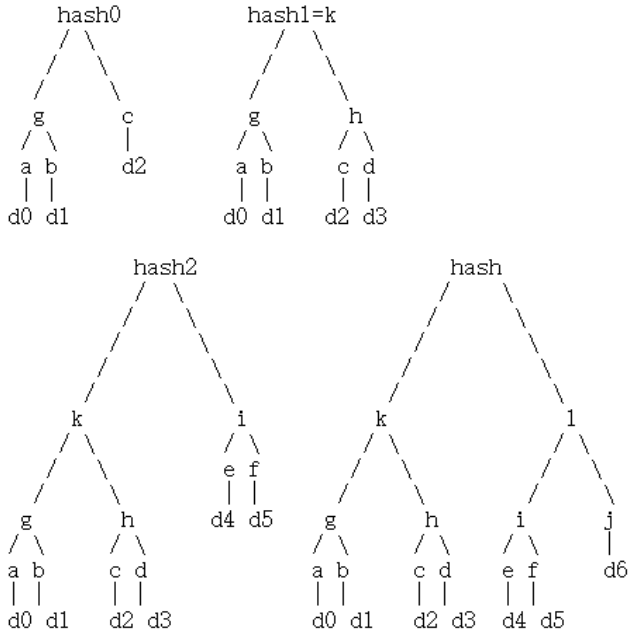

图3　默克尔树构建步骤示意图

其中，各叶节点审计路径如下。

a)　d0 的审计路径为[b, h, l]；

b)　d3 的审计路径为[c, g, l]；

c)　d4 的审计路径为[f, j, k]；

d)　d6 的审计路径为[i, k]。

6.4.3　一致性验证示例

以 7 个叶节点的默克尔树为例，其一致性验证方法如下。

a)　hash0 与 hash 的一致性验证方法为：

PROOF（3, D[7]） = [c, d, g, l]

其中，c、g 用于验证 hash0，d、l 用于验证 hash 与 hash0 一致。

b) hash1 和 hash 的一致性验证方法为：

PROOF（4, D[7]） = [l]

其中，hash 可使用 hash1=k 和 l 来验证。

c) hash2 和 hash 的一致性验证方法为：

PROOF（6, D[7]） = [i, j, k]

其中：

k、i——用于验证 hash2；

J——用于验证 hash 与 hash2 一致。

6.5 数字签名

日志系统中的所有数据结构，应使用 SM2 椭圆曲线公钥密码算法（应符合 GB/T 32918 的规定）完成数字签名。

7 日志系统要求

7.1 概述

日志系统对外公开接受CA提交的SSL证书数据记录,并实时返回该条记录的证书签发时间戳(SCT)。如已添加过相同记录，则返回相同的 SCT 值。

审计机构可通过查看 SCT 数据验证证书是否已在日志系统备案。

Web 服务器在向 SSL 客户端提供证书时，应同时提供所有日志系统的 SCT 数据。SSL 客户端必须拒绝使用无有效 SCT 数据的 SSL 证书。

SCT 数据发布后，日志系统须及时在默克尔树添加相应证书数据记录，并完成根节点数字签名。日志系统数据最大合并延迟（MMD）时间不应超过 24 小时。

7.2 日志条目

7.2.1 生成方法

日志系统公开发布其信任的 CA 根证书列表。CA 在签发证书前向日志系统提交证书备案，按以下环节生成日志条目。

a) CA 提交预签证书（Precertificate）；

b) 日志系统使用该预签证书创建日志条目；

c) 预签证书向用户证书添加关键特殊扩展字段 OID 1.3.6.1.4.1.11129.2.4.3，其 extnValue OCTET STRING 包含 ASN.1 NULL 数据（0x05 0x00 ）。

这个特殊扩展字段是为了确保预签证书无法通过标准的 X.509 v3 客户端验证，并采用以下两种方式生成 TBSCertificate：

——特殊用途（CA:true, Extended Key Usage: Certificate Transparency, OID 1.3.6.1.4.1.11129.2.4.4）预签证书签名证书。该预签证书签名证书由（根证书或中级根证书）CA 证书直接认证，将用于签名用户证书的 TBSCertificate，产生最终用户证书。

——用于 CA 证书签发用户证书。CA 提交预签证书时，须同时提交预签证书签名证书（如使用）及证书链所有证书，直至根证书。

日志系统使用 CA 提供的中级根 CA 证书链，验证用户证书或预签证书签名链的有效性，并在相应的

TBSCertificate 上签发证书签发时间戳（SCT）。

为验证证书链，日志系统应存储并完整证书链信息，包括用户证书或预签证书本身顶级根证书。

日志系统可备案已过期、尚未生效、已被撤销的证书，也可备案按 X.509 验证规则不完全有效的证书，但不应备案无有效证书链到已知根 CA 的证书。

7.2.2 组件要求

日志中的证书条目应包含以下组件：

```
enum { x509_entry（0）, precert_entry（1）, （65535） } LogEntryType;
struct {
          LogEntryType entry_type;
          select （entry_type） {
              case x509_entry: X509ChainEntry;
              case precert_entry: PrecertChainEntry;
          } entry;
      } LogEntry;

      opaque ASN.1Cert<1..2^24-1>;

      struct {
          ASN.1Cert leaf_certificate;
          ASN.1Cert certificate_chain<0..2^24-1>;
      } X509ChainEntry;

      struct {
          ASN.1Cert pre_certificate;
          ASN.1Cert precertificate_chain<0..2^24-1>;
      } PrecertChainEntry;
```

其中：

entry_type——证书条目的类型；

leaf_certificate——提交审计的最终用户证书；

certificate_chain——验证最终用户证书所需的附加证书链,首张证书须为最终用户证书,后续证书须直接证明前序证书，最末证书须为日志接受的顶级根证书；

pre_certificate——提交审计的预签证书；

precertificate_chain——验证预签证书所需的附加证书链。

证书链要求如下。

a) 首张证书应为有效的预签证书；

b) 后续证书应直接证明前序证书；

c) 最末证书应为日志接受的顶级根证书。

证书链长度可由日志系统自行约定。

7.3 证书签发时间戳

本文件定义的证书签发时间戳（SCT）结构如下：

```
enum { certificate_timestamp (0), tree_hash (1), (255) }
  SignatureType;

 enum { v1 (0), (255) }
  Version;

  struct {
      opaque key_id[32];
  } LogID;

  opaque TBSCertificate<1..2^24-1>;

  struct {
    opaque issuer_key_hash[32];
    TBSCertificate tbs_certificate;
  } PreCert;

  opaque CtExtensions<0..2^16-1>;
```

其中：

key_id——日志公钥的 SM3 杂凑值，由 SubjectPublicKeyInfo 的密钥的 DER 编码计算得出；

issuer_key_hash——证书签发者公钥的 SM3 杂凑值，由 SubjectPublicKeyInfo 的密钥的 DER 编码计算得出，用于将签发者绑定到最终用户证书；

tbs_certificate——预签证书的 DER 编码 TBSCertificate 组件，无签名和关键特殊扩展项。

```
struct {
    Version sct_version;
    LogID id;
    uint64 timestamp;
    CtExtensions extensions;
    digitally-signed struct {
        Version sct_version;
        SignatureType signature_type = certificate_timestamp;
        uint64 timestamp;
        LogEntryType entry_type;
        select(entry_type) {
            case x509_entry: ASN.1Cert;
            case precert_entry: PreCert;
        } signed_entry;
       CtExtensions extensions;
```

```
        };
    } SignedCertificateTimestamp;
```

其中：

sct_version——SCT 遵循的协议版本，v1；

timestamp——当前的 NTP 时间[RFC5905]，从纪元(1970 年 1 月 1 日，00:00)开始计量，忽略闰秒，以毫秒为单位；

entry_type——可以隐含；

signed_entry——leaf_certificate；

extensions——此协议版本(v1)的未来扩展，目前没有指定。

7.4 SSL 握手

CA 宜向多个日志系统提交预签证书信息备案，并将所获的所有 SCT 数据写入最终用户证书。本文件定义的 SSL 握手包含一个及以上最终用户证书 SCT 数据。具体方法如下。

a) 将 SignedCertificateTimestampList 结构编码为 ASN.1 OCTET STRING；

b) 将结果数据作为 TBSCertificate 插入 X.509 v3 证书扩展（OID 1.3.6.1.4.1.11129.2.4.2）；

c) 收到证书后，客户端通过重建原始的 TBSCertificate 验证 SCT 签名。

X509 v3 证书扩展中嵌入的 ASN.1 OCTET STRING 的内容如下：

```
opaque SerializedSCT<1..2^16-1>;
        struct {
              SerializedSCT sct_list <1..2^16-1>;
        } SignedCertificateTimestampList;
```

其中：

Serialized SCT——包含序列化 SSL 结构的不透明字节字符串。

SSL 客户端对 SCT 签名的验证要求如下。

——单独解码每个 SCT，如版本升级，则旧的客户端仍解析旧的 SCT，同时跳过升级版本的新 SCT；

——应验证 X.509 v3 扩展项，如有多个 SCT 数据，则应逐一验证。

SSL 服务端应同时发送多个 SCT，以防止部分日志不被客户端接受，如日志系统因不当行为被删除或密钥泄露。

7.5 默克尔树结构

默克尔树输入的结构如下：

```
enum { timestamped_entry (0), (255) }
   MerkleLeafType;

struct {
    uint64 timestamp;
    LogEntryType entry_type;
    select (entry_type) {
        case x509_entry: ASN.1Cert;
```

```
            case precert_entry: PreCert;
        } signed_entry;
        CtExtensions extensions;
    } TimestampedEntry;

    struct {
        Version version;
        MerkleLeafType leaf_type;
        select (leaf_type) {
            case timestamped_entry: TimestampedEntry;
        }
    } MerkleTreeLeaf;
```

其中：

Version——MerkleTreeLeaf 对应的协议版本 v1；

leaf_type——叶节点输入类型，目前仅定义了“timestamped_entry”（对应一个 SCT）；

Timestamp——此证书签发的对应 SCT 的时间戳；

signed_entry——相应 SCT 的“signed_entry”；

Extensions——相应 SCT 的“扩展”。

默克尔树的叶节点为相应“Merkle Tree Leaf”结构的叶节点杂凑值。

7.6 签名树头(STH)

日志系统添加新条目时，应对相应的默克尔树杂凑值和默克尔树信息进行签名（见 8.4），生成签名树头。该数据签名结构如下：

```
  digitally-signed struct {
            Version version;
            SignatureType signature_type = tree_hash;
            uint64 timestamp;
            uint64 tree_size;
            opaque sm3_root_hash[32];
        } TreeHeadSignature;
```

其中：

Version——TreeHeadSignature 所遵循的协议版本 v1；

Timestamp——当前时间的时间戳，须与默克尔树中最新的 SCT 时间戳一致。后续时间戳须晚于前序时间戳；

tree_size——默克尔树条目数；

sm3_root_hash——默克尔树数根的杂凑值。

日志系统应生成不早于最大合并延迟（MMD）的签名树头。如 MMD 期间系统无证书条目更新记录，则使用新的时间戳签名相同的默克尔树杂凑值。

8 客户端消息要求

8.1 概述

SSL 客户端消息以 HTTPS GET 或 POST 请求方式，按以下规范发送。

a) POST 的参数和所有响应编码为 JavaScript 对象表示法 （JSON）对象[RFC4627]；
b) GET 的参数编码为与顺序无关的键/值 URL 参数，使用“HTML 4.01 规范”[HTML401]中描述的“application/x-www-form-urlencoded”格式；
c) 二进制数据采用 base64 编码 [RFC 4648]；
d) JSON 对象和 URL 参数如包含上述字段之外的其他字段，则可忽略这些字段；
e) <log server>前缀可包含路径以及日志服务器名称和端口；
f) “version”通常为 v1，“id”为查询的日志服务器的 log id；
g) 所有错误都应作为 HTTP 4xx 或 5xx 响应返回，并带有错误消息。

8.2 添加证书链

日志添加证书链的方法如下：

POST https://<log server>/ct/v1/add-chain

输入： chain 一组 base64 编码的证书

首张证书为最终用户证书，第二张证书为中级根证书，依此类推，最末证书为顶级根证书或链接到已预置的根证书。

输出： sct_version SignedCertificateTimestamp 结构的版本 V1，十进制

其中：

Id——日志 ID，base64 编码，因请求 SCT 已包含在 SSL 握手中的日志客户端，无需验证，不假设日志 ID 已知；

Timestamp——SCT 时间戳，十进制；

extensions——一种不透明类型，用于未来扩展，日志系统应将其设置为空字符串，客户端应解码 base64 编码的数据，并纳入 SCT 中；

signature——SCT 签名，base64 编码。

如 sct_version 非 v1，则 v1 客户端可能无法验证签名，但不应因此解释为错误。日志客户端无需验证此结构，仅 SSL 客户端需要。

8.3 添加预签证书链

日志添加预签证书链（PreCertChain）的方法如下：

POST https://<log server>/ct/v1/add-pre-chain

输入： chain 一组 base64 编码的预签证书和根证书；

第一个元素是预签用户证书；第二个元素链接到第一个，依此类推，最后一个元素即顶级根证书或链接到已预置的根证书。

输出： 同 8.2。

8.4 获取最新 STH

获取最新 STH 的方法如下：

GET https://<log server>/ct/v1/get-sth

输入： 无

输出： tree_size 默克尔树的大小，以条目为单位，十进制
timestamp 时间戳，十进制
sm3_root_hash 根节点杂凑值，base64

tree_head_signature　上述数据的 TreeHeadSignature

8.5 获取一致性证明

获取两个 STH 之间的默克尔一致性证明方法如下：

GET https://<log server>/ct/v1/get-sth-consistency

输入：first　第一棵默克尔树的 tree_size，十进制

second　第二棵默克尔树的 tree_size，十进制

输出：consistency　默克尔树节点数组，base64 编码

其中：

——两种树的大小均须来自现有 v1 STH；

——默克尔树节点数组数据用于验证已签名的 STH，无需签名。

8.6 获取叶节点审计证明

通过叶节点杂凑值获取默克尔审计证明的方法如下：

GET https://<log server>/ct/v1/get-proof-by-hash

输入：hash　base64编码的v1叶节点杂凑值

tree_size　证明所基于的默克尔树的tree_size，十进制

输出：leaf_index　“hash”参数对应的是从0开始的索引

audit_path　一组base64编码的默克尔树节点，证明包含所选证书

其中：

——Hash应按照8.4定义进行计算；

——tree_size应指定现有的v1 STH。

8.7 获取条目

从日志中获取条目的方法如下：

GET https://<log server>/ct/v1/get-entries

输入：start　需检索的第一个条目的从 0 开始的索引，十进制

end　需检索的最后一个条目的从 0 开始的索引，十进制

输出：entries：　对象数组

其中，每个对象由下列两个数据组成：

——leaf_input：base64 编码的 MerkleTreeLeaf 结构。

——extra_data：与日志条目相关的 base64 编码的无符号数据。在 X509ChainEntry 情况下，为 certificate_chain；在 PrecertChainEntry 情况下为整个 PrecertChainEntry。

此消息未签名，可通过构造与检索到的 STH 对应的默克尔树杂凑值，来验证检索到的数据。所有叶节点均为 v1。v1 客户端不应将无法识别的 MerkleLeafType 或 LogEntryType 值解释为错误，但可将无法识别的叶节点视为默克尔树的不透明输入，验证数据的完整性。

start 和 end 参数应该在 0 <= x < “tree_size”范围内，与本文件 8.4 中 get-sth 返回值一致。

日志系统可通过返回仅涵盖指定范围内的有效条目的部分响应来符合 0 <= "start" < "tree_size" 和 "end" >= "tree_size"的要求。

日志系统可限制每个 get-entries 请求可检索的最大条目数。如客户端检索请求条目数超过限制，则日志系统应返回允许的最大条目数，并按顺序从“start”指定的条目开始。

8.8 获取根证书

获取根证书方法如下：

GET https://<log server>/ct/v1/get-roots

输入： 无

输出： certificates 日志可接受的一组 base64 编码的根证书

8.9 获取条目审计证明

获取默克尔树审计证明方法如下：

GET https://<log server>/ct/v1/get-entry-and-proof

输入： leaf_index 所需条目的索引

tree_size 需要证明的树的 tree_size

注：树的大小应指定现有的STH。

输出： leaf_input base64 编码的 MerkleTreeLeaf 结构

extra_data base64 编码的无符号数据，同 8.7

audit_path 一组 base64 编码的默克尔树节点，证明包含所选证书

注：上述API通常只对调试有用。

9 日志用户

9.1 概述

日志系统用户包括日志提交者、SSL 客户端、监督机构和审计机构。所有日志系统用户应交换 STH，确保信息同步。

9.2 日志提交者

日志提交者通常为 CA，也可以是其他方。

日志提交者应向日志系统提交证书或预签证书，并使用返回的 SCT 构建证书或直接在 SSL 握手中使用。

9.3 SSL 客户端

SSL 客户端通常指支持商用密码算法 SSL 证书的浏览器或移动 APP。

SSL 客户端应在同 Web 服务器握手获得的 SSL 证书中接收 SCT 数据，负责验证 SSL 证书及其证书链和 SCT。具体执行步骤为：计算从 SCT 数据输入的签名及证书，使用相应日志的公钥验证签名。

SSL 客户端不应信任时间戳时间是未来时间的 SCT，不应信任对应的 SSL 证书。

9.4 监督机构

监督机构通常指域名所有者和密码管理机构。

监督机构应监督并检查日志系统是否正确运行，包括检查每个日志中的每个新条目。具体执行步骤如下：

a) 获取当前 STH（见 8.4）；
b) 验证 STH 签名；
c) 获取默克尔树中对应 STH 的所有条目（见 8.7）；
d) 确认所获条目生成的默克尔树哈希与 STH 中的哈希相同；
e) 再次获取当前 STH（见 8.4），直至 STH 改变；

f） 验证 STH 签名；

g） 获取默克尔树中对应 STH 的所有新条目（见 8.7），如长时间无法获取，则视为日志不当行为。

或者：

1） 验证更新后的所有条目列表是否生成了一棵与新 STH 具有相同哈希的默克尔树；或者，如果它不保留所有日志条目：

2） 获取新 STH 与原 STH 的一致性证明（见 8.5）；

3） 验证一致性证明；

4） 验证新条目是否生成一致性证明中的相应元素；

5） 转到步骤 e。

9.5 审计机构

审计机构通常指 SSL 客户端的组成部分，某个独立服务，或监督机构的次要功能。

审计机构具体审计执行步骤如下：

a） 输入日志的部分信息，并验证此信息与已有的其他信息是否一致；

b） 来自同一日志的任何一对 STH，均可通过一致性证明来验证（见 8.5）；

c） 请求默克尔审计证明，通过与 SCT 时间戳+最大合并延迟之后的任一 STH 进行验证（见 8.6）。

审计机构可自行获取 STH（见 8.4）。

10 风险检测

10.1 概述

CA、日志系统运营机构和 Web 服务器共同参与证书透明的应用安全保障。

CA 将证书提交到日志系统运营机构，Web 服务器提供有效的证书签发时间戳，以证明该证书已在日志系统完成透明公示。如证书无透明公示记录，则 SSL 客户端不应接受该证书，同时应明确提示此证书未透明公示。

10.2 错误签发的证书

提交到日志系统的证书，其 SCT 将在最大合并延迟时间内合并至公共日志。如证书错误签发，则仅可能在最大合并延迟时间内因无法识别而存在风险。

10.3 错误签发证书检测

日志系统自身不检测错误签发的证书，但可通过监督机构检测错误并及时纠正。

10.4 不当行为检测

日志系统可能出现的不当行为包括：

a） 未在最大合并延迟时间内将证书与 SCT 合并到默克尔树中；

b） 在不同时间和/或向不同方呈现默克尔树的两个不同的、相互冲突的视图，违反“仅可添加”属性。

上述不当行为的监测路径如下。

——SSL 客户端通过检索 SCT 的默克尔审计证明，可监测到违反最大合并延迟的情形。检查可异步进行，每张证书只需执行一次。为保护用户隐私，检查无需向日志系统显示确切证书信息。用户可向受信任的审计方请求审计证明，或为 SCT 时间戳周围的一批证书请求默克尔证明。

——个人、机构和其他组织均可在审计日志中比较最新的签名树头版本。如同一日志监测到两个相冲突的 STH，则表明该日志系统存在不当行为。

11 效率考虑

默克尔树数据结构模型设计保持低通信开销，确保最优效率。审计日志的完整性无需第三方维护每个完整日志的副本，可在新条目可用时更新签名树头，而无需重新计算整棵树。第三方审计时，只需针对日志现有 STH 获取默克尔一致性证明，即可有效验证其默克尔树更新的仅可添加属性，而无需审计整个默克尔树。

参 考 文 献

[1] IETF.RFC 6962: Certificate Transparency [EB/OL] https://datatracker.ietf.org/doc/html/rfc6962, 2013年6月

[2] IETF.RFC 9162: Certificate Transparency Version 2.0 [EB/OL] https://datatracker.ietf.org/doc/html/rfc9162, 2021年12月