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体系框架。

证书透明技术国际标准RFC9162为RFC6962升级版本，但因技术成熟原因并未实际应用，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，包括证书认证系统、日志系统、客户端消息接口的要求。

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 20518-2018 信息安全技术 公钥基础设施 数字证书格式

GB/T 25069-2022 信息安全技术 术语

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

GB/T 38636-2020 信息安全技术 传输层密码协议（TLCP）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 界定的以及下列术语和定义适用于本文件。

3.1

证书认证系统 certificate authentication system

对数字证书的签发、发布、更新、撤销等数字证书全生存周期进行管理的系统。

[来源：GB/T 25069-2022,3.787]

3.2

证书透明日志系统 certificate transparency log system

针对数字证书的签发行为公示系统，简称：证书备案系统。

注： 证书透明日志系统通过记录和公示所有已签发数字证书，使得第三方可验证证书的合法性和权威性，以监控和审计证书认证机构（CA），防止证书错误签发或恶意签发行为。

3.3

预签证书 precertificate

证书认证机构在签发订户证书之前提交到证书透明日志系统的数字证书，包含订户证书所有信息但不能用于建立安全连接（如HTTPS）。

注： 预签证书含有一个值为 NULL 的特定证书扩展项，使其不能被标准的 X.509v3 客户端验证。

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）
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 整体架构

证书透明（Certificate Transparency）是一个针对商用密码公共TLS/SSL服务器证书（简称SSL证书）的备案管理与公开监督体系，由证书透明日志系统、证书认证系统（CA）、浏览器、网站（SSL证书）、证书监督系统（由监测方和审计方参与运行）共同组成。证书透明系统整体架构如图1所示。

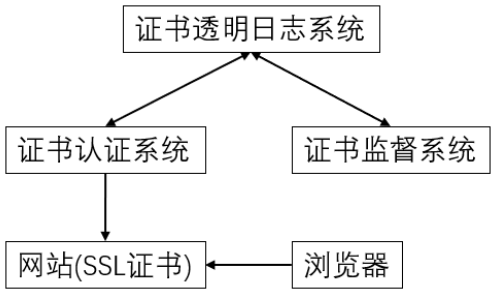

图 1 证书透明体系架构图

证书透明体系架构中各方的关系如下。

a) 日志运营：证书透明日志系统主要负责维护CA信任列表、存储所有信任的CA提交的SSL证书包括预签证书，维护仅可添加的默克尔树结构，生成证书签发时间戳（SCT）及签名树头（STH）。证书透明日志系统可分布式，防止单点故障。
b) 日志使用：

1) 证书认证系统：主要负责提交预签证书至证书透明日志系统，获取 SCT，并写入到 SSL 证书中，通过证书扩展项传递 SCT 至浏览器。
2) 证书监督系统：实时扫描证书透明日志系统，在检测异常证书时触发告警机制；通过验证默克尔树签名树头杂凑值，确保证书透明日志系统未被篡改，对日志系统进行一致性验证。

c) 日志应用：
1) 网站（SSL 证书）：网站部署包含了 SCT 的 SSL 证书，供浏览器验证。
2) 浏览器：验证 SCT 的合法性（签名来源、时间戳有效性），不信任无可信证书透明日志系统备案记录的 SSL 证书。

5.2 参与实体

证书透明系统涉及以下参与实体：

a) 日志服务运营商：提供证书透明日志服务、维护和管理证书透明日志系统的实体或组织，通常为浏览器厂商或证书认证机构（CA）。
b) 日志提交者：向日志系统提交证书或预签证书并获取 SCT 构建证书的组织，通常为证书认证机构（CA）。
c) 网站运营者（域名所有者）：负责网站日常运营管理以及向证书认证机构提交 SSL 证书申请并部署证书的实体或组织，通常为运营网站的企业、公司或其他实体。
d) SSL 客户端：支持商用密码算法 SSL 证书的浏览器或移动 APP，通常由浏览器厂商负责。
e) 监测方：监测并检查日志系统行为的正确性，包括检查每个新条目的独立实体或工具，可以是第三方监管机构、证书认证机构（CA）、域名所有者或安全研究人员。
f) 审计方：对日志系统进行审计的实体或组织，负责验证日志系统的完整性和正确性、检查日志的一致性、验证证书的合法性以及生成审计报告等，审计方可以是审计机构、浏览器厂商、域名所有者或安全研究人员。

上述实体中，监测方和审计方共同参与证书监督系统的运行。

5.3 运行流程

证书透明体系运行流程如图 2 所示。

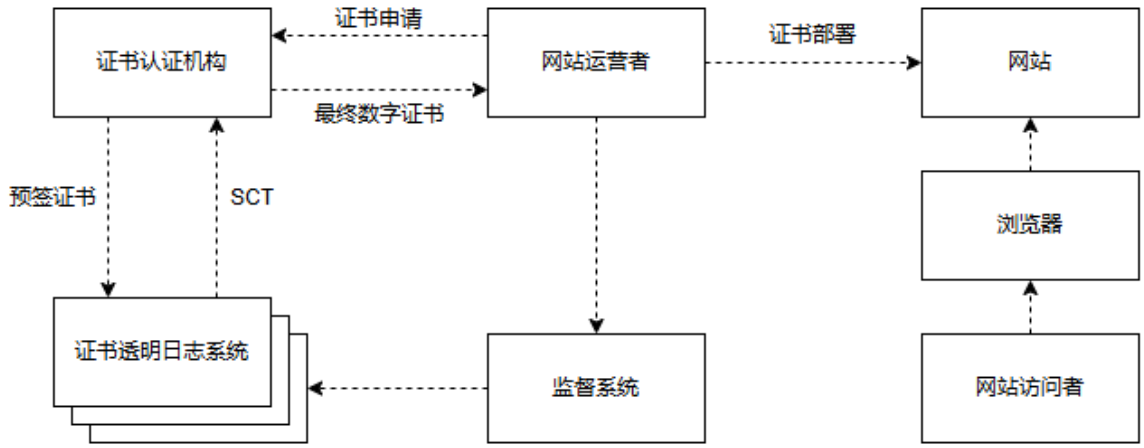

图 2 证书透明体系运行流程

证书透明体系的运行包含以下环节。

a) 网站运营者（域名所有者）作为证书申请人，向证书认证机构（CA）提交证书申请，同时完成网站域名控制权验证；
b) 证书认证机构在完成应审核证书申请人身份以及域名验证记录，审核通过后生成预签证书；
c) 证书认证机构将生成的预签证书提交至证书透明日志系统（可向多个证书透明日志系统提交，各日志系统由不同的日志服务运营商运营）；
d) 证书透明日志系统应验证收到的预签证书，验证通过后生成证书签发时间戳（SCT）并将 SCT 返回给证书认证机构，SCT 包含证书相关信息及证书透明日志系统签名，可证明证书已被证书透明日志系统记录备案；
e) 证书认证机构在收到足够数量的证书透明日志系统的 SCT 后，将 SCT 数据作为 X.509 扩展项写入到最终 SSL 证书中；
f) 证书认证机构将带有 SCT 的最终 SSL 证书分发给网站运营者，网站运营者将其部署到网站服务器上；
g) 当网站访问者通过浏览器访问网站时，浏览器验证网站服务器 SSL 证书的 SCT 有效性，如验证不通过，浏览器宜向用户发出警告提示证书可能存在风险，宜阻止用户访问该网站。

证书透明日志系统应对外公开备案信息，安全研究人员、监督系统等可通过分析证书透明日志系统数据及时发现证书签发异常行为，如有未经授权的证书签发，则相关方可采取必要措施（如撤销证书、追究责任等），以保障整体网络环境的安全性和证书体系安全。

证书透明体系运行流程涉及各实体之间的通信协议，证书透明日志系统对外提供客户端消息接口（接口定义规范见附录 A），日志提交者、SSL 客户端、监测系统、审计系统按规范调用证书透明日志系统接口完成数据交互。

6 证书认证系统要求

6.1 概述

证书认证系统接收 SSL 证书申请，对证书申请人身份及域名合法性进行审核鉴证。在正式签发用户证书之前，证书认证系统应生成一张预签证书并提交至证书透明日志系统。证书透明日志系统根据预签证书创建一个与用户证书等效的条目，签发证书签发时间戳(SCT)。

证书认证系统将 SCT 数据作为 X.509 扩展项写入到正式签发的用户证书中。数字证书格式应遵循 GB/T 20518-2018 标准，且包含的 SCT 数据格式应符合本文件第 7.3 的要求。

6.2 签发预签证书

预签证书的证书扩展项应添加一个特殊的关键扩展字段（OID 1.3.6.1.4.1.11129.2.4.3，其 extnValue 八位字节字符串包含 ASN.1 NULL 数据（0x05 0x00）），此字段确保预签证书不能被标准的 X.509 v3 客户端验证。

预签证书其他项与正式签发的用户证书保持一致，

证书认证系统可采用以下任一方式签发预签证书。

a) 使用专用的预签 CA 证书。预签 CA 证书的证书扩展应包含（CA:true, Extended Key Usage: Certificate Transparency, OID 1.3.6.1.4.1.11129.2.4.4），预签 CA 证书应直接由签发用户证书的 CA 证书（根证书或中级根证书）认证。
b) 使用签发用户证书的 CA 证书。

CA 提交预签证书时，应同时提交预签 CA 证书（如使用）及证书链中所有证书用于有效性验证。

预签证书中的CA签名具有约束力，如预签证书错误签发则等同于用户证书错误签发。

6.3 签发订户证书

订户证书是正式签发给证书申请人的SSL证书，该证书内嵌证书透明日志系统签发的SCT。证书认证系统可向多个证书透明日志系统提交预签证书，并将所获的所有SCT写入最终用户SSL证书。

证书认证系统是通过将SignedCertificateTimestampList结构编码为ASN.1 OCTET STRING，其结果数据作为X.509 v3证书扩展（OID 1.3.6.1.4.1.11129.2.4.2）插入TBSCertificate。

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：

SerializedSCT 为一个 opaque 类型的字节串，包含序列化的 SSL 结构。此编码确保 SSL 客户端可单独解码每个 SCT（例如，版本升级时则旧的客户端仍解析旧的 SCT，同时跳过升级版本的新 SCT）。

7 日志系统要求

7.1 概述

证书透明日志系统采用默克尔树模型作为基本的数据结构，并基于默克尔树生成审计路径并进行一致性验证，相关数据结构说明和审计路径、一致性验证示例见附录 B。

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。

监督方可通过查看 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 客户端宜拒绝使用无有效 SCT 数据的 SSL 证书。

SCT 数据发布后，日志系统应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（见 6.2）。

日志系统使用证书认证系统提供的中级根 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

证书透明日志系统存储结构为证书条目，每张成功申请到 SCT 的证书生成证书条目。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——证书条目类型，本协议版本的未来修订可能会添加新的 LogEntryType 值，客户端对新证书条目类型的处理方式见附录 A；

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,首张证书应为最终用户证书,后续证书应直接认证前序证书，最后一张证书应为证书透明日志系统接受的根证书

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，首张证书应为有效的预签 CA 证书且能认证预签证书，每张后续证书应直接认证前序证书。最后一张证书应为证书透明日志系统接受的根证书。

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。

如预签证书未使用将签发最终 SSL 证书的 CA 证书签名，则 TBS Certificate 可将其签发者改为将签发最终 SSL 证书的 CA 证书，也可通过从最终 SSL 证书中提取 TBSCertificate 并删除 SCT 扩展来重建此 TBSCertificate。

因 TBSCertificate 应包含匹配预签证书签名算法和最终用户证书签名算法的 AlgorithmIdentifier，故应使用相同算法和参数对其进行签名。如预签证书为使用预签 CA 证书签发，且 TBSCertificate 中存在授权密钥标识符扩展，则相应扩展应存在于预签 CA 证书中。在这种情况下，TBSCertificate 授权密钥标识符相应改为匹配的最终签发 CA。

```
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;
```

数字签名元素编码定义如下。

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——可从呈现 SCT 的上下文中隐含；

signed_entry——在 X509ChainEntry 的情况下为 leaf_certificate， 在 PrecertChainEntry 的情况下为 PreCert；

extensions——此协议版本(v1)的未来扩展，目前暂无指定。

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），对无法识别的叶节点类型的处理方式见附录 A；

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)

证书透明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，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，应与默克尔树中最新 SCT 时间戳一致，后续时间戳应晚于前序时间戳；

tree_size——默克尔树条目数；

sm3_root_has——默克尔树数根的杂凑值。

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

8 日志数据利用

8.1 概述

证书透明日志体系是一个保障SSL证书安全的生态体系，包括日志系统用户、日志数据验证方、检测方和审计方。

8.2 日志系统用户

日志系统用户包括日志系统运维方，CA 机构、监测方、审计方。CA 机构是主要日志提交者，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 构建用户证书。

8.3 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 客户端宜在同 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晚于当前时间的 SCT，不应信任对应的 SSL 证书。

8.4 监督系统

8.4.1 概述

监督系统涉及的参与实体包括监视方、审计方（见 5.2）。

8.4.2 监视方

监视方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 A.4）；
b) 验证 STH 签名；
c) 获取默克尔树中对应 STH 的所有条目（见 A.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 A.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 A.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 A.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。

8.4.3 审计方

审计方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 A.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 A.6）。

审计机构可自行获取 STH（见 A.4）。

9 风险检测

9.1 概述

CA、日志系统运营方和 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，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，同时宜提示此证书未透明公示。

提交到日志系统的证书，其 SCT 将在最大合并延迟时间 24 小时内合并至公共日志。如证书错误签发，在 SCT 合并前可能存在应用安全风险。

9.2 监督系统检测

日志系统自身不检测错误签发的证书，但可通过监督系统中的监测方和审计方（如域名所有者、监管机构等）实时监控和分析证书透明日志系统数据库，并在检测到错误签发时采取纠正措施。

域名所有者可以通过定期检测日志系统或订阅第三方监测服务来确保所有 CA 机构没有签发未授权的 SSL 证书，如有发现，则应立即联系 CA 机构申请撤销证书。

9.3 浏览器检测

浏览器验证 SSL 证书并披露验证结果，根据不同结果给予网站访问者以下提示。

a) 验证通过的 SSL 证书：浏览器宜提示证书透明信息；
b) 验证未通过的 SSL 证书：浏览器宜提示不安全或证书未完成证书透明备案等信息。

9.4 日志系统不当行为检测

日志系统可能出现以下不当行为。

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。

上述不当行为的检测路径如下。

——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。

——其他参与方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。

附 录 A
（规范性）
日志客户端消息接口定义

A.1 概述

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应为 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) 所有错误应作为 HTTP 4xx 或 5xx 响应返回，并带有错误消息。

A.2 添加证书链

请求方式：POST；

请求 URL：https://<log server>/ct/v1/add-chain。

接口定义如表 A.1 所示。

表A.1 添加证书链接口定义

输入参数	说明
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。
输出参数	说明
sct_version	SignedCertificateTimestamp 结构的版本，如V1
id	日志id，base64编码。
timestamp	SCT时间戳，十进制。
extensions	用于将来扩展信息。并非所有参与者都需要了解该领域的数据。日志系统应将其设置为空字符串。客户端应解码base64编码数据并将其包含在SCT中。
signature	为SCT签名，base64编码。
注：如sct_version非v1，则v1客户端可能无法验证签名，但不得因此解释为错误。 日志客户端无需验证此结构，仅SSL客户端需要。如此结构作为二进制blob提供，则无需升级到v1客户端。	

A.3 添加预签证书链

请求方式：POST；

请求 URL：https://<log server>/ct/v1/add-pre-chain。

接口定义如表 A.2 所示定义。

表A.2 添加预签证书链接口定义

输入参数	说明
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。
输出参数	说明
sct_version	SignedCertificateTimestamp 结构的版本，如V1
id	日志id，base64编码。
timestamp	SCT时间戳，十进制。
extensions	用于将来扩展信息。并非所有参与者都需要了解该领域的数据。日志系统应将其设置为空字符串。客户端应解码base64编码数据并将其包含在SCT中。
signature	为SCT签名，base64编码。
注：如sct_version非v1，则v1客户端可能无法验证签名，但不得因此解释为错误。 日志客户端无需验证此结构，仅SSL客户端需要。如此结构作为二进制blob提供，则无需升级到v1客户端。	

A.4 获取最新 STH

请求方式：GET；

请求 URL：https://<log server>/ct/v1/get-sth。

接口定义如表 A.3 所示。

表A.3 添加预签证书链接口定义

输出参数	说明
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编码。
tree_head_signature	上述数据的TreeHeadSignature。
tree_size	默克尔树的大小，以条目为单位，十进制。

A.5 获取一致性证明

请求方式：GET；

请求 URL：https://<log server>/ct/v1/get-sth-consistency。

接口定义如表 A.4 所示。

表A.4 获取一致性证明接口定义

输入参数	说明
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。
输出参数	说明
consistency	默克尔树节点数组，base64编码。
注：两种树的大小均须来自现有v1 STH；	

输入参数	说明
默克尔树节点数组数据用于验证已签名的STH，无需签名。	

A.6 获取审计证明

请求方式：GET；

请求URL：https://<log server>/ct/v1/get-proof-by-hash。

接口定义如表A.5所示。

表A.5 获取审计证明接口定义

输入参数	说明
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。
输出参数	说明
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。
注：hash应按7.5定义进行计算，tree_size应指定现有的v1 STH。。	

A.7 获取条目

请求方式：GET；

请求 URL：https://<log server>/ct/v1/get-entries。

接口定义如表 A.6 所示。

表A.6 获取条目接口定义

输入参数	说明
start	需检索的第一个条目的从0开始的索引，十进制。
输出参数	说明
entries	对象数组。每个对象数组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。
注：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客户端不得将无法识别的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无法识别的输入，验证数据的完整性。 start和end参数应在0 <= x < “tree_size”范围内，与本文件附录A.3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。	

A.8 获取根证书

请求方式：GET；

请求 URL：https://<log server>/ct/v1/get-roots。

接口定义如表 A.7 所示。

表A.7 获取根证书接口定义

输出参数	说明
certificates	日志可接受的一组base64编码的根证书。

A.9 获取条目审计证明

请求方式：GET；

请求URL：https://<log server>/ct/v1/get-entry-and-proof。

接口定义如表 A.8 所示。

表A.8 获取条目审计证明接口定义

输出参数	说明
leaf_index	所需条目的索引。
tree_size	需要证明的树的tree_size。树的大小必须指定现有的STH。
输出参数	说明
leaf_input	base64编码的MerkleTreeLeaf结构
extra_data	base64编码的无符号数据，同附录A.6。
audit_path	一组 base64 编码的默克尔树节点，证明包含所选证书

附 录 B
（资料性）
默克尔树及审计路径

B.1 数据结构

本文件定义的日志系统数据结构采用默克尔树模型，密码杂凑算法为SM3算法（SM3算法信息见GB/T 32905）。日志系统的记录数据仅可添加（append-only），其数据的完整性可通过相应审计路径予以验证。

默克尔树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。

默克尔树模型为基于SM3算法的二叉树形数据结构，包含叶节点、分支节点和根节点。其中，叶节点存储节点下数据的杂凑值，分支节点存储相邻两个叶节点的杂凑值。

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。根节点的杂凑值，可标识整个默克尔树的所有数据。

默克尔树模型具体构建方法如下：

输入数据条目列表，通过SM3算法计算出消息摘要（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）}，默克尔树杂凑值（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（也称为叶哈希），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）}。

叶节点和分支节点的杂凑值计算方式不同。这种域分离是实现抗二次原像性的必要条件。

在构建默克尔树模型时，输入数据条目列表长度不必为2的幂。

以7个叶节点的默克尔树为例，其数据结构如图6所示。

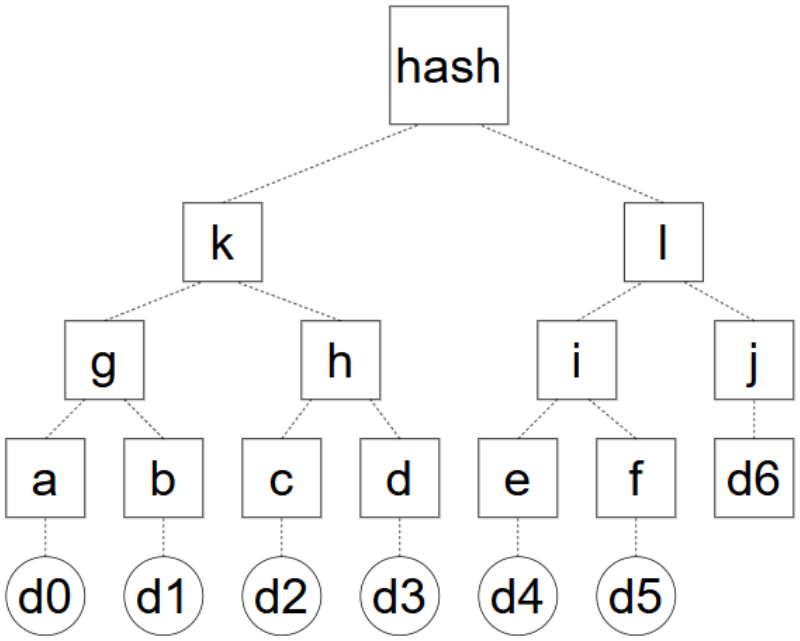

图 B.1　默克尔树结构示意图

B.2　审计路径

本文件定义的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。

其中：

：——列表串联，

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

以 7 个叶节点的默克尔树为例，其构建步骤如图 7 所示。

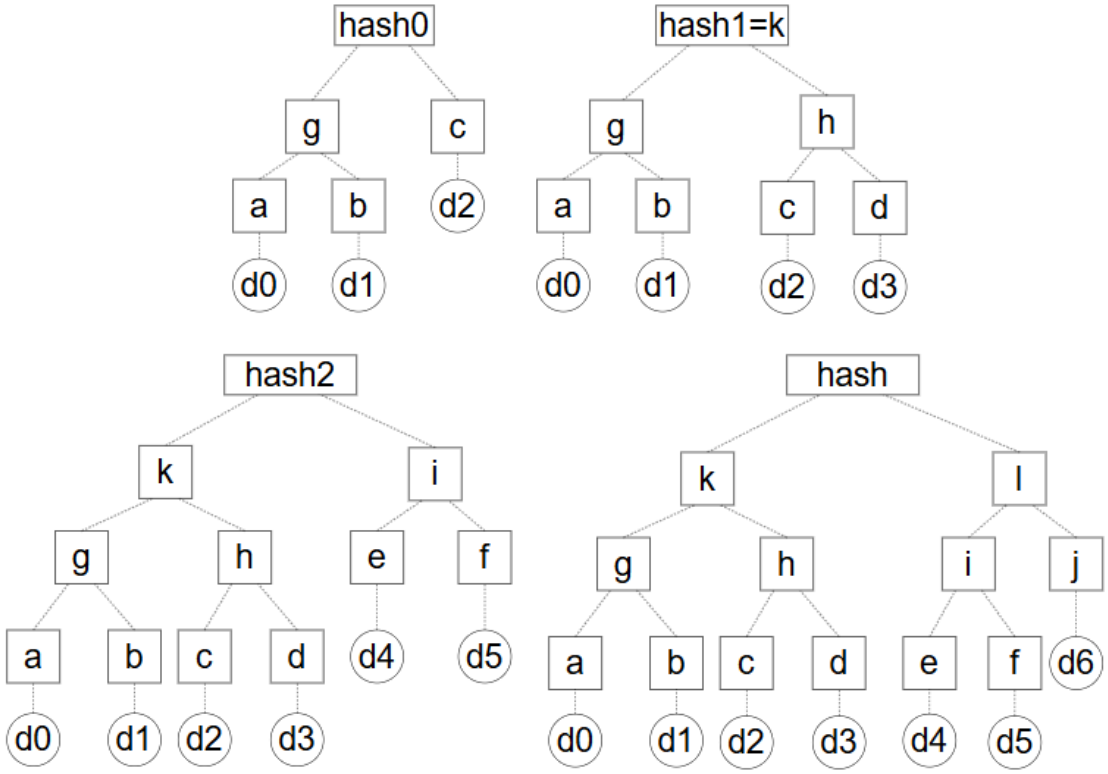

图 B.2　默克尔树构建步骤示意图

其中，各个叶节点审计路径如下：

d0 的审计路径为[b, h, l]，

d3 的审计路径为[c, g, l]，

d4 的审计路径为[f, j, k]，

d6 的审计路径为[i, k]。

B.3　一致性验证

证书透明日志系统所有数据记录“仅可添加（append-only）”，不可更改，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先前公布的前 m 个叶子的哈希 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，必须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。我们证明左子树条目 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。我们证明右子树条目 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。

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。

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