ICS 35.030
CCS L 80

中 华 人 民 共 和 国 密 码 行 业 标 准

GM/T XXXXX—XXXX

证书透明体系框架

Framework For Certificate Transparency System

（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 III
引　　言 IV
1 范围 1
2 规范性引用文件 1
3 术语和定义 1
4 缩略语 3
5 基本原理 3
5.1 整体架构 3
5.2 参与实体 4
5.3 运行流程 4
6 证书认证系统要求 5
6.1 概述 5
6.2 签发预签证书 6
6.3 签发服务器证书 6
7 日志系统要求 7
7.1 概述 7
7.2 日志条目 7
7.3 证书签发时间戳 8
7.4 TLS 握手中的 SCT 传递 9
7.5 默克尔树结构 10
7.6 签名树头(STH) 11
8 日志数据利用 11
8.1 概述 11
8.2 日志系统用户 11
8.3 TLS 客户端 11
8.4 监督系统 12
9 安全风险监测与响应 12
9.1 概述 12
9.2 监督系统异常识别 13
9.3 浏览器验证与提示 13
9.4 日志系统不当行为检测 13
9.5 各参与方操作要求 13
附　录　A　（规范性）　日志客户端消息接口定义 14
A.1 概述 14
A.2 添加证书链 14
A.3 添加预签证书链 14
A.4 获取最新 STH 15
A.5 获取一致性证明 15
A.6 获取审计证明 16
A.7 获取条目 16

A.8 获取根证书 17
A.9 获取条目审计证明 17
附 录 B （资料性） 默克尔树及审计路径 19
B.1 数据结构 19
B.2 审计路径 20
B.3 一致性验证 21
参 考 文 献 23

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。

本标准起草单位：

本标准主要起草人：

引　言

商用密码服务器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服务器证书应用安全，建立公开、透明的商用密码服务器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大范围应用。本文件的目标是在商用密码体系下建立证书透明框架，而非简单照搬国际标准的最新版本。以已形成广泛共识的RFC 6962为基础，有利于国内CA、日志运营商和浏览器厂商在过渡期内平稳实施。

证书透明体系框架

1 范围

本文件规定了商用密码服务器证书透明体系总体架构、参与实体、运行流程，以及证书认证系统、证书透明日志系统、客户端验证、监测审计和日志客户端消息接口的要求。

本文件适用于商用密码服务器证书透明体系（CT）的建设、运行、应用和安全管理。

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

证书透明 certificate transparency

一种以防止证书错误签发或恶意签发为目标，通过公开可验证的日志系统，对数字证书的签发、记录和披露进行透明化管理，使得第三方可监测和审计证书认证机构（CA）行为的体系。简称为CT。

3.2

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]

3.3

证书透明日志 certificate transparency log

以默克尔树为基础数据结构，记录证书透明信息、仅可追加写入（append-only），并能生成用于一致性验证和审计验证的可验证密码学证明的数据记录。

3.4

证书透明日志系统 certificate transparency log system

提供证书透明日志服务的系统，负责接收证书提交、生成证书签发时间戳（SCT）、发布签名树头（STH），并提供一致性证明和审计证明查询服务。

3.5

日志服务器 log server

提供证书透明日志系统对外接口服务的服务器实体，负责处理证书提交请求并返回相应响应。

3.6

服务器证书

用于在TLS、TLCP等传输层安全协议中标识服务器身份，并支持密钥协商或身份鉴别的终端实体数字证书。本文件中也称TLS/SSL证书。

3.7

预签证书 precertificate

证书认证机构在签发服务器证书之前，为提前向证书透明日志系统提交并锁定待签发证书核心数据而构造的数字证书。预签证书包含服务器证书的全部信息，但须包含一个特定的关键毒化扩展字段（Poison Extension，OID 1.3.6.1.4.1.11129.2.4.3），以确保其不能被标准 X.509 v3 客户端用作正式证书建立连接。

3.8

证书签发时间戳 signed certificate timestamp; SCT

证书透明日志系统在接收到有效数字证书或预签证书后返回的、带有数字签名的凭证，用于证明该证书已被日志系统接收并将在最大合并延迟（MMD）规定时间内被纳入日志的默克尔树中。

3.9

签名树头 signed tree head; STH

证书透明日志系统对默克尔树（Merkle Tree）在某一时刻的整体状态（包括当前树的规模及根节点杂凑值）连同时间戳一起进行数字签名所形成的数据结构。

3.10

默克尔树 Merkle tree

一种基于密码杂凑算法构建的二叉树形数据结构。其叶节点包含基础数据条目的杂凑值，非叶节点包含其直接子节点杂凑值串联后的杂凑值，具有防篡改和快速验证数据存在性与一致性的特性。本文件所用密码杂凑算法为SM3算法，应符合GB/T 32905的规定。

3.11

最大合并延迟 maximum merge delay; MMD

证书透明日志系统自签发证书签发时间戳（SCT）之日起，将对应证书条目实际合并至默克尔树并对公众公开可用所允许的最长绝对时间。

3.12

监测方 monitor

具体参与实体（如域名所有者、CA机构、安全研究人员），负责持续检索并解析证书透明日志系统中的新增条目，以识别未经授权的数字证书签发行为或日志系统状态异常。

3.13

审计方 auditor

具体参与实体（如浏览器厂商、审计机构），负责通过请求和验证一致性证明及审计证明，核验日志系统的仅可添加属性，确保签名树头（STH）在时间轴上未被分叉或回滚。

3.14

证书监督系统 monitor system

由监测方和审计方共同构成的逻辑体系（而非具体的软件系统或管理机构），负责对证书透明日志系统的数据一致性、仅可添加属性以及证书签发行为的合规性进行持续观察、验证与异常上报。监督系统本身不直接干预证书签发流程，但通过密码学验证和数据分析为决策者（如浏览器厂商、监管机构）提供安全依据。

3.15

审计证明 audit proof

一组默克尔树中间节点的杂凑值列表，用于向客户端或审计方在数学上证明：某张特定的数字证书（或预签证书）的条目，确实被包含在当前具有特定签名树头（STH）的默克尔树中。

3.16

一致性证明 consistency proof

一组默克尔树中间节点的杂凑值列表，用于向监测方或审计方在数学上证明：当前包含 n 个条目的新默克尔树，完整且不变地包含了之前公布的只有 m 个条目（$m \leq n$）的旧默克尔树。

3.17

日志服务运营商 log service operator

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。运营商负责部署和维护日志服务器（3.5），确保其服务可用性，并发布信任锚列表。

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）

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）

RFC：请求意见稿（Request for Comments）

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

STH：已签名树头（Signed Tree Head）

TLS：传输层安全性协议（Transport Layer Security）

TLCP：传输层密码协议（Transport Layer Cryptographic Protocol）

5 基本原理

5.1 整体架构

证书透明体系(CT)由证书透明日志系统、证书认证系统、证书监督系统和网站服务器等共同组成，涉及TLS客户端（含浏览器）、监测方和审计方等参与实体。证书透明系统整体架构如图1所示。

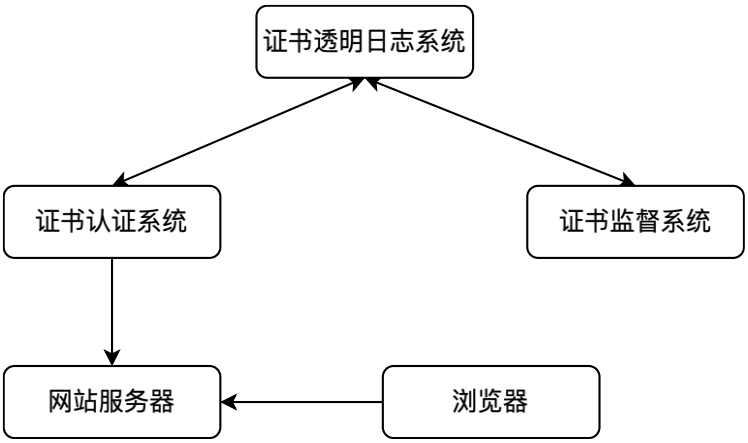

图1　证书透明体系架构图

CT体系架构中各方按照“提交—记录—嵌入—验证—监督”的闭环关系协作如下:

a) 日志服务层（核心存储）：证书透明日志系统维护仅可添加的默克尔树。日志服务运营商应至少部署物理或逻辑上独立的多个日志实例，以防止单点故障。
b) 证书签发层（数据提交方）：证书认证系统（CA）作为日志提交者，在签发最终证书前，应将预签证书提交至日志系统以获得SCT，并将SCT嵌入最终证书中。
c) 终端应用层（数据消费方）：（1）网站服务器：部署嵌入了有效SCT的服务器证书；（2）TLS客户端（浏览器）：在握手时提取并验证证书中的SCT签名及时效性，验证失败应拒绝连接。
d) 监督审计层（外部校验方）：由监测方和审计方组成证书监督系统。监测方持续拉取日志新增条目以发现异常证书；审计方通过请求一致性证明来验证日志系统的仅可添加属性是否被破坏。

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）。日志服务运营商通过部署日志服务器（见 3.5）对外提供接口服务。
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 TLS 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) TLS 客户端（含浏览器）：支持商用密码算法 TLS 证书的浏览器或移动 APP，负责验证服务器证书及其 SCT 的有效性，通常由浏览器厂商负责。
e) 监测方（见 3.12）：在 CT 体系中负责持续监测日志系统新增条目，可以是行业监管机构（如国家密码管理局）、证书认证机构（CA）、域名所有者或安全研究人员。
f) 审计方（见 3.13）：在 CT 体系中负责通过验证一致性证明及审计证明核验日志完整性，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。监督系统并非独立的物理实体或管理机构，而是由监测方和审计方这些具体参与实体所构成的逻辑协作体系。

5.3　运行流程

证书透明体系运行流程如图2所示。

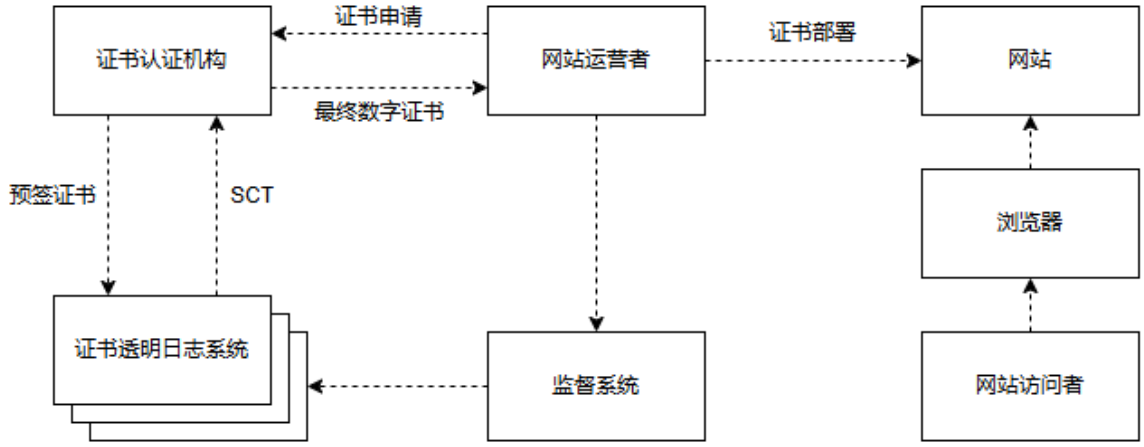

图2　证书透明体系运行流程

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，为获取 SCT，证书认证机构构造一张与最终实体证书内容等同但包含特殊限制扩展（Poison）的预签证书；

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 用于证明该证书已被证书透明日志系统接收，并将在最大合并延迟（MMD）规定时间内被纳入日志的默克尔树中（见 3.8）；

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，将获取到的 SCT 作为扩展项数据嵌入，并最终签署服务器证书；

f) 证书认证机构将带有 SCT 的最终 TLS 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 TLS 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透明记录信息，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。

CT 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、TLS 客户端、监测方、审计方按规范调用证书透明日志系统接口完成数据交互。监测方及审计方（共同构成证书监督系统）。

6　证书认证系统要求

6.1　概述

证书认证系统接收 TLS 证书申请，对证书申请人身份及域名合法性进行审核鉴证。

当采用 X.509 v3 证书扩展项方式向客户端传递 SCT 时，在正式签发服务器证书之前，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服务器证书等效的条目，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服务器证书中。数字证书格式应遵循 GB/T 20518 标准，且包含的 SCT 数据格式应符合本文件第 7.3 的要求。

若采用 TLS 握手扩展或 OCSP 装订（Stapling）方式传递 SCT，证书认证系统或网站运营者可直接将服务器证书提交至日志服务器（见 3.5）进行透明记录，以获取 SCT。

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，其扩展值（extnValue）的数据结构按照 GB/T 16262.1 的规定进行编码，具体为一个包含 ASN.1 NULL 数据（0x05 0x00））的八位字节字符串。该关键扩展字段用于确保证书路径验证引擎在处理预签证书时将其视为无效，从而防止预签证书被误用为最终实体证书建立连接。

预签证书其他项与正式签发的服务器证书保持一致，本文件支持以下两种预签证书签发方式，由证书认证系统根据实际情况自行选择：

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 证书应直接由签发服务器证书的 CA 证书（根证书或中级根证书）认证。

b) 使用签发服务器证书的 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服务器证书错误签发。

注： 上述两种预签证书签发方式（专用预签 CA 证书与直接使用签发服务器证书的 CA 证书）均为本文件支持的合规方式。若使用专用预签 CA 证书（方式 a），日志系统在验证证书链时需能检索到该专用 CA 证书；若使用方式 b），则无需额外构造证书链。证书认证系统可根据自身密钥管理策略和证书生命周期管理现状自行选择，但无论采用何种方式，CA 均应对预签证书内容的正确性承担与正式证书同等的法律责任。

6.3 签发服务器证书

服务器证书是正式签发给证书申请人的 TLS 证书，该证书内嵌证书透明日志系统签发的 SCT。证书认证系统应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 TLS 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 TLS 结构。此编码确保 TLS 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）。

对于面向公共网络服务的服务器证书，证书认证系统应至少从两个相互独立、且宜由不同日志服务运营商运行的证书透明日志系统获取有效 SCT，并将其嵌入服务器证书，或通过本文件规定的其他机制

向客户端提供。宜优先选择由不同日志服务运营商运营的日志系统。对于专用网络、测试环境或封闭业务场景，SCT 数量可由相关管理策略规定。

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TLS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。

监测方可通过查看 SCT 数据验证证书是否已在日志系统透明记录。

网站服务器在向 TLS 客户端提供证书时，应同时提供所有日志系统的 SCT 数据。TLS 客户端宜拒绝使用无有效 SCT 数据的 TLS 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）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服务器（见 3.5）提交证书进行透明记录（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，验证服务器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，日志系统应存储完整的证书链信息，包括从服务器证书或预签证书至顶级根证书的全部节点。

日志系统可记录已过期、尚未生效、已被撤销的证书，也可记录按 X.509 验证规则不完全有效的证书，但不应记录无有效证书链到已知根 CA 的证书。

7.2.2 组件要求

证书透明日志系统的存储结构以“证书条目”（LogEntry）为基本单位。日志系统为每一张成功获取 SCT 的证书创建一个对应的证书条目，该条目包含证书本身及其验证链信息，并作为默克尔树的叶节点数据被持久化存储。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：提交审计的服务器证书；

certificate_chain：验证服务器证书所需的附加证书链,首张证书应为签发服务器证书的 CA 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（SM3 算法应符合 GB/T 32905 的规定），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服务器证书；

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

若预签证书并非由最终签发 TLS 服务器证书的 CA 进行数字签名，则可在构建 TBSCertificate 时将其签发者替换为最终发证的 CA；或者，也可通过从最终 TLS 证书中提取 TBSCertificate 并删除 SCT 扩展来重构此 TBSCertificate。

因 TBSCertificate 应包含匹配预签证书签名算法和服务器证书签名算法的 AlgorithmIdentifier，故应使用相同算法和参数对其进行签名。在此过程中，所采用的公钥密码算法应符合 GB/T 32918（所有部分）的规定。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[2]，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 TLS 握手中的 SCT 传递

CA 宜向多个日志系统提交预签证书信息进行透明记录，并将所获的所有 SCT 数据写入服务器证书。本文件规定 TLS 握手场景下 SCT 的传递方式。采用 X.509 v3 证书扩展方式传递 SCT 时，服务器证书中应包含一个及以上服务器证书 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) 客户端在收到包含 SCT 的证书后，应移除该证书中的 SCT 扩展项（OID 1.3.6.1.4.1.11129.2.4.2）以重构原始的 TBSCertificate 结构，并以此作为输入参数验证 SCT 的数字签名。

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 TLS 结构的不透明字节字符串。

TLS 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。

TLS 服务端应同时发送多个 SCT，以防单一日志系统因发生密钥泄露或不当行为被客户端取消信任，从而导致服务器证书被拒绝。

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_hash：当前默克尔树根节点的 SM3 密码杂凑值。该值按照附录 B.1 中定义的默克尔树杂凑值（MTH）计算方法得出，长度为 32 字节。日志系统在计算该根节点值时，必须严格区分叶节点与分支节点的杂凑输入前缀（即叶节点前缀为 0x00，分支节点前缀为 0x01），以防范长度扩展攻击。算法应符合 GB/T 32905 的规定。
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。若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 TLS 证书安全的生态体系，包括日志系统用户、监测方和审计方。

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 构建服务器证书。

8.3 TLS 客户端

TLS 客户端通常指支持商用密码算法 TLS 证书的浏览器或移动 APP。
TLS 客户端应在与网站服务器握手获得的 TLS 证书中接收 SCT 数据，负责验证 TLS 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。
TLS 客户端不应信任签发时间戳晚于当前系统时间的 SCT，不应信任对应的 TLS 证书。

客户端应验证服务器证书所携带 SCT 的数量、签名有效性、时间戳合理性以及日志系统可信状态。公共信任场景下，有效 SCT 数量不满足本文件要求时，客户端应按安全策略拒绝连接或提示风险。

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测方、审计方（见 5.2）。

8.4.2 监测方

监测方应监测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) 确认计算所得的根节点杂凑值与 STH 中包含的杂凑值相一致；
e) 再次获取当前 STH（见 A.4），直至 STH 改变；
f) 验证 STH 签名；
g) 获取默克尔树中与该 STH 对应的新增条目（见 A.7），若超过最大合并延迟（MMD）时间仍未获取到，则判定该日志系统存在不当行为。
h) 验证日志系统的一致性，具体可执行以下任一操作：
   1) 验证更新后的所有条目列表是否生成了一棵与新 STH 具有相同杂凑值的默克尔树；
   或者，如果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。
i) 转到步骤 e）继续监测。

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 安全风险监测与响应

9.1 概述

CA、日志系统运营方和网站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，网站服务器向客户端提供有效的 SCT，以证明该证书已被日志系统纳入透明记录。公共信任场景下，客户端应验证服务器证书所携带或关联的 SCT。SCT 缺失、签名验证失败、时间戳不合理、日志系统不可信或有效 SCT 数量不满足本文件要求时，客户端应按安全策略拒绝连接或提示风险。

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。若证书错误签发，在 SCT 合并前可能存在应用安全风险。

9.2 监督系统异常识别

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）持续检索并分析证书透明日志系统数据，并在监测到异常签发行为时采取纠正措施。

域名所有者可以通过定期验证日志系统一致性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 TLS 证书，如有发现，则应立即联系 CA 机构申请撤销证书。

9.3 浏览器验证与提示

浏览器验证 TLS 证书及其 SCT，并根据验证结果向网站访问者提示以下信息：

a) 验证通过的 TLS 证书：浏览器宜提示证书透明信息；
b) 验证未通过的 TLS 证书：浏览器宜提示不安全或证书未完成透明记录等信息。

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。

对上述不当行为的检测路径如下。

a) 客户端异步审计：客户端（或其委托的受信任审计方）可使用 SCT 获取默克尔审计证明，以监测日志系统是否违反最大合并延迟（MMD）的承诺。为保护用户隐私，此类检查宜采用异步机制，且对于同一证书仅需执行一次。
b) 视图一致性比对：证书透明生态中的各参与方（监测方、审计方）宜定期交换并比对各自获取的最新签名树头（STH）。若针对同一日志系统监测到两个冲突的 STH，则可作为该日志系统存在不当行为的密码学证据。

9.5 各参与方操作要求

a) 证书认证系统运营方：（1）应至少向两个相互独立（宜由不同日志服务运营商运行）的日志系统提交预签证书并获得有效 SCT；（2）应将获取的所有有效 SCT 嵌入服务器证书的 X.509 v3 扩展项中；（3）若因技术原因无法嵌入（如老旧系统兼容性），应采用 TLS 握手扩展或 OCSP 装订方式传递 SCT，但不得降低安全性要求。
b) 日志服务运营商：（1）应保证日志条目的最大合并延迟（MMD）不超过 24 小时；（2）应公开发布并定期更新本日志系统信任的根证书列表；（3）应保障 STH 的发布时间间隔不超过 MMD，即使在无新增条目的情况下，也应以新时间戳重新签署原 STH。
c) 网站运营者（域名所有者）：（1）应部署包含有效 SCT 的服务器证书；（2）在证书到期续期或更换证书时，应重新向日志系统提交并获取新的 SCT，不得重复使用旧 SCT。
d) 监测方和审计方：（1）监测方应持续拉取日志条目，并对新条目与最新 STH 进行根哈希一致性核算；（2）审计方应定期（建议每周至少一次）向日志系统请求新旧 STH 之间的一致性证明，若验证失败，应立即向行业监管机构或浏览器根程序上报。

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[3] ；

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”[5] 中描述的“application/x-www-form-urlencoded”格式；

c) 二进制数据采用 Base64 编码[4]；

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>（日志服务器，见 3.5）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唯一标识符（log id），该标识符为日志系统公钥的 SM3 杂凑值（32 字节），以 Base64 编码形式表示；

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，首张证书应为服务器证书，第二张证书为中级根证书，以此类推，最后一张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。

客户端对于无法识别的 sct_version（非 v1 版本），应忽略该 SCT 但继续处理其他 SCT，不应将其视为错误。提交证书的日志客户端（即 CA 端）不要求验证此结构，SCT 的签名验证由 TLS 客户端（浏览器）在握手阶段执行。

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，首张证书应为预签证书， 第二张证书为用于验证该预签证书的证书链。使用专用预签CA证书的，第二张证书应为预签CA证书；最后一张证书应为日志系统接受的根证书，或能够链接至日志系统预置信任锚的证书。
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。

客户端对于无法识别的 sct_version（非 v1 版本），不应将其视为错误。日志客户端不要求验证此结构，仅 TLS 客户端进行验证。

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 获取最新 STH 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	当前默克尔树根节点的 SM3 杂凑值（32字节），Base64 编码。
tree_head_signature	上述数据的TreeHeadSignature。

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。

输入参数	说明
注：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。	

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	整数，十进制，指定本次审计证明所依据的默克尔树的当前大小（以条目数为单位）。该值应取自日志系统已发布的STH中的tree_size字段。
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应按附录 B.1 定义的叶节点杂凑值计算方式进行计算，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。	

输入参数	说明
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

输入参数	说明
leaf_index	所需条目的索引。
tree_size	整数，十进制，指定审计证明所针对的默克尔树的大小（以条目数为单位）。该值应对应于一个已由日志系统签名并发布的STH。
输出参数	说明
leaf_input	Base64编码的MerkleTreeLeaf结构
extra_data	Base64编码的附加数据，其内容与日志条目类型相关：对于X509ChainEntry类型，为证书链（certificate_chain）；对于PrecertChainEntry类型，为完整的PrecertChainEntry结构。该字段的数据结构与附录A.6中entries数组内对应条目的extra_data字段一致。
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

本附录为资料性附录，提供默克尔树数据结构、审计路径及一致性验证方法的示例性说明，旨在帮助读者理解第7章“日志系统要求”中相关技术原理。附录内容不构成强制性要求，但日志系统实现者若需与CT生态中其他参与方（如TLS客户端、审计方）保持数据格式兼容，建议参考本附录所描述的计算方法。

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 = 2^{⌊log₂(n-1)⌋}（即 k 为小于或等于 n 的最大 2 的整数次幂，且满足 k ＜ n ≤ 2k）。长度为 n 的列表 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 B.1 所示。

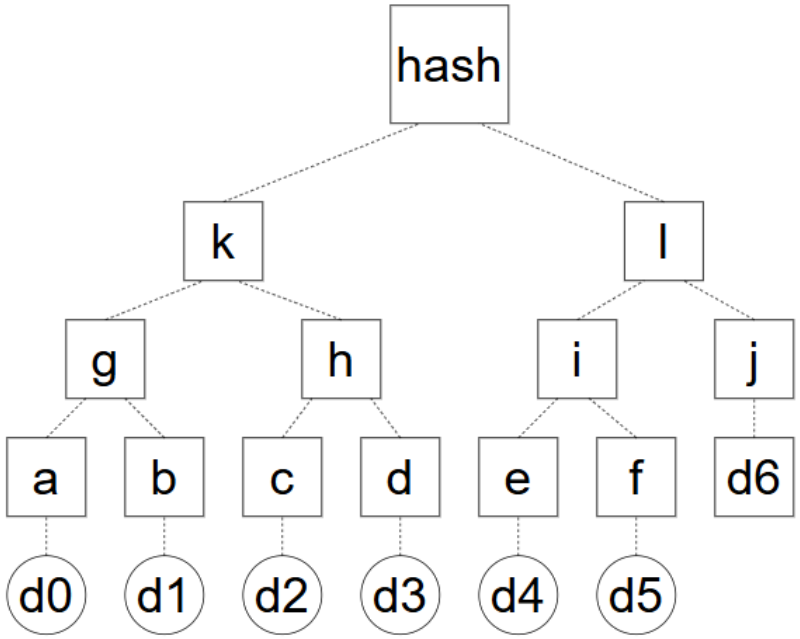

图 B.1　默克尔树结构示意图

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 为满足 k < n 的最大 2 的整数次幂（即 k = 2^⌊log₂(n-1)⌋）。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 B.2 所示。

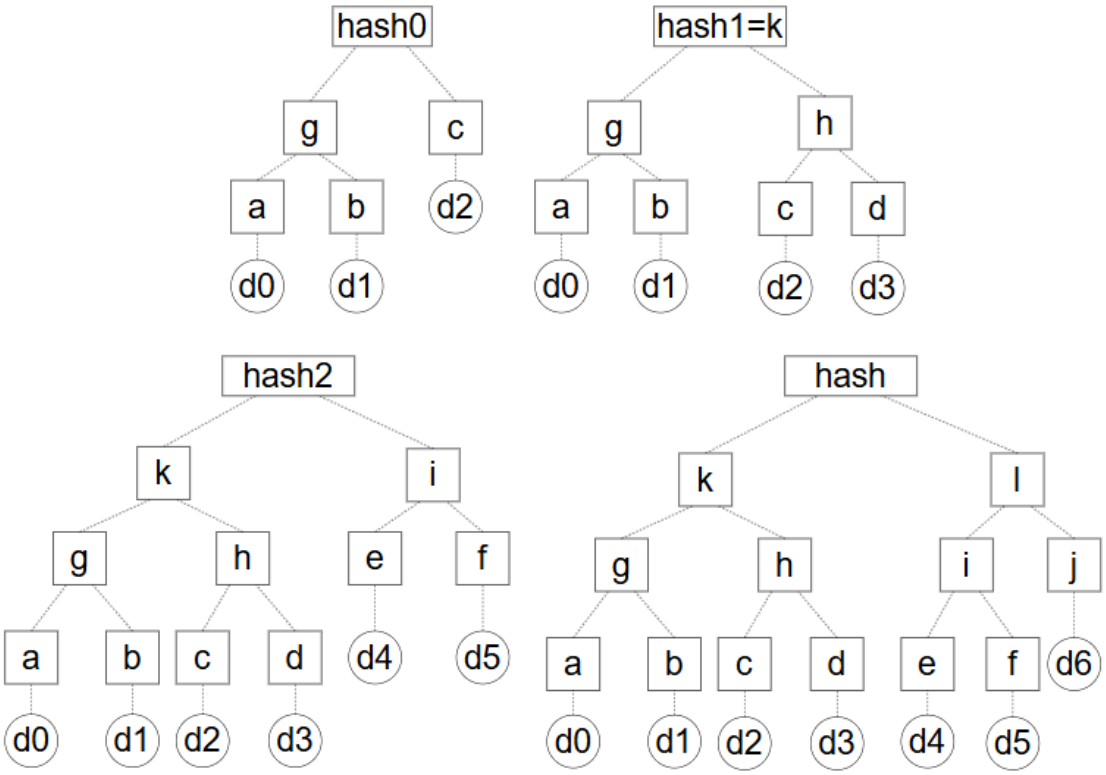

图 B.2　默克尔树构建步骤示意图

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：

对于当前包含 n 个条目的默克尔树杂凑值 MTH(D[n])，以及先前公布的包含前 m 个条目（m≤n）的默克尔树杂凑值 MTH(D[0:m])，二者之间的一致性证明是指默克尔树中的一组节点列表，该列表用于验证前 m 个输入 D[0:m]在新旧两棵树中保持完全等同。因此，一致性证明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），同时该节点集合的特定子集也可用于重构并验证旧的默克尔树杂凑值 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 = 2^⌊$\log_2$(n-1)⌋（即 k 为小于或等于 n 的最大 2 的整数次幂，且满足 k < n ≤ 2k）。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。需证明左子树条目 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。需证明右子树条目 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. Certificate Transparency Version 2.0: RFC 9162 [S/OL]. (2021-12). https://datatracker.ietf.org/doc/html/rfc9162.

[2] IETF. Network Time Protocol Version 4: RFC 5905 [S/OL]. (2010-06). https://datatracker.ietf.org/doc/html/rfc5905.

[3] Notation IETF. The application/json Media Type for JavaScript Object Notation (JSON): RFC 4627 [S/OL]. (2006-07). https://datatracker.ietf.org/doc/html/rfc4627.

[4] IETF. The Base16, Base32, and Base64 Data Encodings: RFC 4648 [S/OL]. (2006-10). https://datatracker.ietf.org/doc/html/rfc4648.

[5] W3C. HTML 4.01 Specification [S/OL]. (1999-12). https://www.w3.org/TR/html401/.