Provable Security for PKI Schemes

Sara Wrótniak
School of Computing
University of Connecticut
Storrs, CT
sara.wrotniak@uconn.edu

Hemi Leibowitz[*]
Faculty of Computer Science
The College of Management Academic Studies
Rishon LeZion, Israel
menahemle@colman.ac.il

Ewa Syta[†]
Department of Computer Science
Trinity College
Hartford, CT
ewa.syta@trincoll.edu

Amir Herzberg
School of Computing
University of Connecticut
Storrs, CT
amir.herzberg@uconn.edu

ABSTRACT

PKI schemes provide a critical foundation for applied cryptographic protocols. However, there are no rigorous security specifications for realistic PKI schemes, and therefore, no PKI schemes were proven secure. Cryptographic systems that use PKI are analyzed by adopting overly simplified models of PKI, often simply assuming securely-distributed public keys. This is problematic given the extensive reliance on PKI, the multiple failures of PKI systems, and the complexity of both proposed and deployed systems, which involve complex requirements and models.

We present game-based security specifications for PKI schemes and analyze important and widely deployed PKIs: PKIX and two variants of Certificate Transparency (CT). These PKIs are based on the X.509v3 standard and its CRL revocation mechanism. Our analysis identified a few subtle vulnerabilities and provides reduction-based proofs showing that the PKIs ensure specific requirements under specific models (assumptions). To our knowledge, this is the first reduction-based proof of security for a realistic PKI scheme, e.g., supporting certificate chains.

CCS CONCEPTS

• **Security and privacy** → **Security requirements**; **Formal security models**; **Key management**.

KEYWORDS

PKI, provable-security

[*]The work was partially completed during the author's PhD studies at the Dept. of Computer Science, Bar-Ilan University, Israel

[†]The work was partially completed during a visiting position at the School of Computing, University of Connecticut, Storrs, CT

ACM Reference Format:
Sara Wrótniak, Hemi Leibowitz, Ewa Syta, and Amir Herzberg. 2024. Provable Security for PKI Schemes. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3670374

1 INTRODUCTION

Public Key Infrastructure (PKI) provides an essential foundation for applications that rely on public key cryptography, which is crucial to ensure security in open networks and systems. Early PKI ideas were proposed in 1978 [26], and the first version of the X.509 standard [8] was published in 1988. Since then, the deployment of PKI has been dominated by X.509, specifically, by the IETF PKIX standard, which adopts version 3 of X.509 (X.509v3) for Internet protocols, most notably, TLS/SSL [45]. *Certificate Transparency (CT)* [43, 29] is a recent, widely-deployed extension to PKIX, motivated by multiple PKI failures, mainly, rogue certificates issued by corrupt or negligent Certificate Authorities (CAs). A significant number of other PKI schemes were recently proposed, with different goals and properties, and different, non-trivial designs, including [48, 13, 35, 25, 53, 2, 58, 51, 52, 33, 15, 1, 54, 27].

Considering the importance, variety and complexity of (some) PKI schemes, it is essential to ensure their security. Currently, there are no rigorous security specifications for realistic PKI schemes, and therefore, no PKI schemes were proven secure. This situation stands in sharp contrast to the accepted norms in (applied and theoretical) modern cryptography, which require well-defined security requirements and reduction-based proofs of security. These norms began in the 1980s with the seminal papers defining secure encryption and secure signature schemes. We present the first complete[1] definitions and analysis for (certificate-based) PKI schemes and their security.

The lack of rigorous specifications and analysis for PKI schemes is especially alarming, since PKI provides a critical infrastructure to applied cryptography, i.e., security of many applied cryptographic systems depend on the security of the underlying PKI. Extensive efforts to prove cryptographic protocols may be moot when these protocols depend on an insecure PKI scheme. The concerns are

[1]Basic PKI schemes, without even revocation and certificate chains, were presented in [7, 3, 16]; the long version of [3] describes how revocation could be added.

even greater, considering that attacks against PKI are not only a theoretical threat, but also major concerns in practice [50, 12].

Rigorous security specifications are relevant to practical, emerging developments and applications of PKIs, too. Version 2.0 of the European Union eIDAS (Electronic Identification, Authentication, and Trust Services) regulation [10], approved in 2024, aims to establish a comprehensive digital identity framework across the EU. As noted in [37, 20], eIDAS 2.0 sets out specific technical requirements and constraints for the existing Web PKI. It essentially mandates that web browsers accept and trust CAs controlled by EU member states, and may limit or prohibit security enhancements such as CT.

Rigorous definitions and analysis often allow the identification of subtle yet significant issues that otherwise may go unnoticed. We identified a few of these. First, the CRL design does not achieve the intuitively expected, guaranteed notion of revocation, but only a weaker notion, *Accountable* Δ*-Revocation.* Second, in a similar fashion, CT does not achieve *Guaranteed* Δ*-Transparency* and only ensures a weaker version of transparency, *HL* Δ*-Transparency*, assuming that a certificate has been logged by at least one honest logger. Third, the non-standard CTwAudit extension [24] avoids this assumption, but still ensures only a weaker notion of *Audited* Δ*-Transparency.* Finally, as also noted in [31], in CT, ensuring awarness of issued certificates (transparency) requires one or more honest monitors that collectively cover *all* the logs that relying parties trust in validating transparency of certificates.

To address these concerns, we present the *first rigorous (game-based) definitions of security requirements for PKI schemes*, and then the *first reduction-based proofs of security* for practical PKI schemes. Table 1 summarizes our results.

Defining and proving security for PKI schemes is challenging, especially for post-X.509 schemes, whose requirements (goals) and designs are more advanced and complex. Historically, PKI schemes evolved without adhering to the traditional process of defining cryptographic primitives and goals before protocol design. This deviation stemmed from the simplicity of early PKIs (e.g., PKIX v1), which used basic signature schemes for certificate issuance, eliminating the perceived need for further definitions or formalizations. While PKI properties may appear simple, formally defining them is more challenging due to their rapid evolution and increasing complexity in addressing real-world demands. PKI schemes vary in communication, synchronization, and adversary models, and may even involve different entities; for instance, CT introduces *loggers* and *monitors* alongside certificate authorities. Further, the distributed nature of PKI systems introduces critical aspects that require careful consideration in their definition and analysis, such as using Δ to represent a 'propagation delay' to reflect real-world constraints of, for example, communication delays. This evolution has resulted in numerous PKI scheme variations, each with unique properties and subtleties. As a result, existing works often use informal security requirements and models for PKI schemes.

To illustrate the challenge of properly defining goals of PKI systems, consider *accountability*, a basic goal of PKI systems with surprising subtleties. Intuitively, accountability is the ability to *identify* the CA that is *responsible* for the issuing of an unauthorized certificate ψ. However, *who is the responsible CA* for ψ? Instinctively, we may expect this to be the *issuer* of ψ, i.e., in X.509 certificates, the CA identified in the issuer field of ψ, denoted as $\psi.issuer$. However, surely $\psi.issuer$ should be held accountable for ψ only if it actually issued (signed) ψ. On the contrary, $\psi.issuer$ should not be held accountable if the public verification key pk used to validate ψ is *not* a correct public-key of $\psi.issuer$. Also, obviously, $\psi.issuer$ can only be considered accountable if it is a real, supposedly trustworthy CA. For example, consider a scenario where a rogue CA issues a certificate ψ' which fraudulently specifies pk as a public verification key of $\psi.issuer$; this rogue CA should be held accountable, rather than the (benign or even non-existing) $\psi.issuer$.

Revocation is another basic goal of PKI systems which is not trivial to define. In fact, we found it necessary to define two variants of revocation: *Guaranteed* and *Accountable.* The Guaranteed variant is more intuitive; basically, a PKI scheme ensures *Guaranteed* Δ*-Revocation* if a certificate ψ revoked by a benign CA at some time t, will not be considered as valid by any benign party after time $t + \Delta$. However, during our analysis, we realized that PKIX does not ensure Guaranteed Δ-Revocation (Claim 1). Instead, PKIX ensures a weaker notion which we call *Accountable* Δ*-Revocation* (§3.4). Intuitively, Accountable Δ-Revocation means that if ψ was revoked at time t by its benign issuing CA yet considered valid after $t + \Delta$, then we can identify, and hold accountable, a rogue CA responsible for this failure to revoke ψ.

Transparency is another important PKI goal, underlying *Certificate Transparency (CT)* [28, 46]. Intuitively, transparency aims to ensure that certificates are available for scrutiny, in the form of a *public log* where certificates must appear within a specified time-frame after being issued. Transparency allows detection of rogue certificates (and applying accountability), before their use by attackers, and does not rely on victims detecting their use. CT was motivated by the detection of the issuance of over 530 fraudulent certificates by DigiNotar CA [56, 28]; the detection was only a month after the breach of the DigiNotar private key. This incident motivated the creation of public logs of certificates to enable swift detection of rogue certificates, aiming to avoid reliance on a set of trusted third parties [28].

However, our analysis shows that the CT specification [28, 46] ensures only a weak notion of transparency (HL Δ-Transparency), which requires that a certificate has been logged by at least one honest logger. This is not due to a vulnerability of the cryptographic mechanisms used by CT, which were shown to be secure by Dowling et al. [11] and Chase and Meiklejohn [9]. While the underlying CT cryptography is sound, its deployment in CT is secure only if at least one logger is honest; e.g., as noted in [11, 9], a rogue logger can simply ignore some requests. The CT specifications[2] [43, 46] also do not specify or require gossip and audit, implicitly assumed by [11, 9].

We have also analyzed CTwAudit, which is a variant of CT, supported as an option by Google's Chrome, where the browser performs auditing of the loggers [24]. We show that CTwAudit ensures another weak variant of the Δ-Transparency requirement, *Audited* Δ*-Transparency.*

We use the *Modular Security Specifications (MoSS) framework* [17] to define PKI requirements and the adversary, synchronization,

[2]Version 1 of the CT specifications [43] incorrectly states that the public logs can be *untrusted.*

Table 1: PKI requirements defined in this work, and properties we prove for prominent PKI systems.

	Requirements				
	Unforgeability	**Accountability**	**Δ-Revocation (Definition 3 and Algorithm 4)**		**Δ-Transparency**
PKI scheme	**(Algorithm 2)**	**(Algorithm 3)**	**Accountable**	**Guaranteed**	**(Definition 4 and Algorithm 5)**
PKIX (X.509 version 3 with CRL)					✗ (n/a)
CT	✓ (Theorem 3)			✗ (Claim 1)	HL-Transparency (Theorem 4)
CTwAudit					Audited-Transparency (Theorem 6)

network and other models assumed by different PKIs. The use of MoSS provides essential modularity, as different PKIs support different requirements and assume different models. MoSS enabled us to separate requirements from models, significantly simplifying the definitions. This modularity is crucial given the complexity of PKI definitions. However, MoSS has drawbacks, such as its lack of protocol composition support. We hope that future work will result in a framework that combines necessary modularity with composition support, potentially through a new or adapted MoSS variant, Adapting our approach to a composable framework could enable the application of our PKI requirements and security proofs for PKIX, CT, and CTwAudit to analyze the security of other PKI-dependent protocols or schemes.

We present pseudocode to rigorously define and analyze (minimally simplified[3] versions of) the most well-known and widely-deployed PKI schemes: PKIX (X.509 version 3 with CRL as defined in [39]), CT [43, 46] and CTwAudit [24]. Table 1 summarizes the results of our analysis of these schemes.

Out of scope. We do not address how relying parties select their *trust anchors*, i.e., the identities of the 'root CAs'; and we mostly ignore constraints on the allowed certificate-chains, such as the *name, length* and *policy constraints* (defined in X.509v3 and PKIX). A model of such *trust decisions* for PKI systems was proposed by Maurer [34], extended by [32, 5], and others [18, 30, 49, 4, 22]. These constraints and solutions are complementary and orthogonal to our results; additional constraints can only prevent an adversary from 'breaking' the PKI.

Contributions. The contributions of this work are as follows.

(1) Presents the first definition of a (non-trivial) PKI scheme, e.g., supporting certificate chains.
(2) Presents the first rigorous security requirements for PKI schemes, including the *unforgeability*, *accountability*, *revocation*, and *transparency* requirements.
(3) Presents the first analysis and proofs of security for the most widely-deployed PKI standards, PKIX, CT and CTwAudit.
(4) Identifies subtle aspects of these PKIs, especially their failure to meet the stronger (and simpler, natural) revocation and transparency requirements; defines weaker notions and proves they are achieved.
(5) Introduces and constructs a *certificate scheme* (Section 4.1), an abstraction for applying signatures to structured information. Certificate schemes simplify definition and analysis of PKI schemes, and may have additional applications.

Organization. The paper is organized as follows. We define *PKI* schemes in §2 and define security requirements for PKI in §3. We present the specifications of PKIX in §4 and CT in [57] and we analyze their security in §5 and [57]. Finally, we conclude and discuss future work in §6.

2 PKI SCHEMES

PKI schemes define how to issue, manage and use *certificates*. Usually, e.g., in X.509, a certificate is a signed object, containing some *certified information*. In §2.1, we describe common certificate fields and different certificate types. In §2.2, we discuss *basic PKI entities*. Then, we discuss basic *PKI functions*: certifying (issuing) and revoking certificates (§2.3), and evaluating the *validity* of a certificate (§2.4). Finally, in §2.5, we define *PKI schemes* and *transparent PKI schemes*, which are PKI schemes with additional entities and functions used to ensure transparency.

2.1 Certificate Fields and Types

A *certificate* is a string which encodes the value of multiple fields, as well as a signature, or other cryptographic mechanism, to validate the authenticity of the certificate. Different PKIs may certify different information (fields), use different encodings or different signature algorithms, and, in principle, may even use a different design (i.e., not a signature over an object). For example, in X.509 and PKIs based on it, certain certified information is encoded as a 'field', while other information is encoded as an 'extension'. That said, all deployed PKI schemes have similar designs.

PKI standards, including X.509, PKIX and CT, define and refer to specific named values in certificates. We refer to all of these named values as 'fields'. We list in Table 2 some of the important fields[4]. We use the *dot notation* to refer to a specific field in a certificate, e.g., $\psi.issuer$ refers to the value of the *issuer* field in certificate ψ. The exceptions are the *type* and *PKIadded* fields; they are not defined in X.509 or other existing PKIs, yet we found them important.

The *type* field is used to distinguish between *different types of certificates*. X.509 defines two types of certificates: *public key*

[3]We included every aspect which appeared to possibly impact the requirements. Our most significant simplification is that we focus on the standard CRL revocation mechanism, while the specifications also allow OCSP and proprietary revocation mechanisms. We also simplify by assuming that each logger keeps only one log, that the public key used for the log is the same as the logger's public key, and that loggers' self-certified keys and issued SCTs do not have a validity period, i.e., never expire. We expect that automation will be necessary to extend our analysis to cover all aspects of the specifications.

[4]We use the term 'field' for all of these named values, and have abstracted away the encoding. However, PKI specifications often use other terms to refer to these 'fields', and may use different encodings for different fields. For example, PKIX uses the terms dnsName *component* (in the *subjectAltName* extension) and *cA Boolean* (in the BasicConstraints extension). For simplicity, assume that each cert has one or no value for each field, ignoring, e.g., that a certificate may contain both dnsName and email.

certificates and *attribute certificates*, which we identify by $\psi.type =$ *'PubKey'* and $\psi.type =$ *'ATTR'*, respectively. Public key certificates associate a public key with a particular subject ('owner' of the public key), whereas *attribute certificates* do not contain a public key (i.e., if $\psi.type =$ *'ATTR'*, then $\psi.pk = \bot$). Further, the *type* field can mark other types of certificates issued by the PKI, e.g., *pre-certificates*, which can be submitted to loggers in CT, or *Route Origin Authorizations (ROAs)*, defined in the Resource Public Key Infrastructure (RPKI) [42], used to specify allowed announcements for IP prefixes. In the case of ROAs, the *issuer* certifying a ROA would have a public-key certificate proving its ownership of a relevant IP prefix. Certificates can have additional fields indicating specific constraints of the certificates.

We use the *PKIadded* field to distinguish between fields that were submitted prior to the creation of the certificate and fields that are added during its creation. For example, the $\psi.pk$ field is the public key provided by the subject, i.e., before the creation of the certificate, while the $\psi.issuer$ field in PKIX is added by the CA when it issues the certificate. Hence, we use the $\psi.PKIadded$ field to identify such fields, e.g., in the previous example: 'issuer' $\in \psi.PKIadded$. To refer to the entire set of (name, value) fields in certificate ψ, including the *PKIadded* field, we use the notation $\psi.tbc$, e.g.: ('pk', pk) $\in \psi.tbc$ or ('PKIadded', {'issuer', 'type', 'PKIadded'}) $\in \psi.tbc$.

2.2 PKI Entities

A PKI scheme $\mathcal{P}$ is defined by a set of functions, some stateful and some stateless, and a set N of stateful entities. Entities in N can perform the stateful functions, e.g., issue certificates. We refer to the entities in N that issue public key certificates as *certificate authorities (CAs)*. There could be other entities in N, e.g., Certificate Transparency (CT) uses *loggers* and *monitors*. Entities in N may be *honest (benign)* or *corrupt*, i.e., controlled by an adversary. *Relying parties* are entities which use only stateless PKI functions, in particular, the certificate validation function, which allows the relying parties to decide whether to *rely on the certificate* (i.e., use the certified public key) or not.

The state of the entities in N is initialized using a dedicated *initialization* operation $\mathcal{P}.\mathsf{Init}$. Typically, $\mathcal{P}.\mathsf{Init}$ outputs a *self-certified public key certificate*, i.e., a certificate ψ which certifies a key for its issuer ($\psi.issuer = \psi.subject$), and is validated (successfully) using the certified public key $\psi.pk$. Self-certified public key certificates can be outputted by any entity: a CA, logger or a monitor. Typically, to validate a certificate ψ, we use a set *store.CAs* of *trusted root certificates* which are self-certified by trusted CAs.

2.3 Certifying and Revoking Certificates

A certificate ψ is issued using the private certification key of a CA, which the CA maintains as part of its state st. To issue a certificate, the CA uses the PKI's $\mathcal{P}$.Certify operation, namely, $\mathcal{P}.\mathsf{Certify}(st, clk, tbc, aux) \rightarrow (st, \psi, aux)$, which takes as input the entity's local state st, local clock clk and the set of (name, value) fields to be certified tbc, and outputs an updated state st, and, if successful, a signed certificate ψ s.t. $\psi.tbc = tbc$.

Since certificates are typically issued for a specific time period, most PKI schemes provide a way to revoke certificates before their expiration date, for example, if a certificate is found to be fraudulently issued or the corresponding private key exposed. Revocation is done by the issuer using a dedicated revoke operation, denoted as $\mathcal{P}$.Revoke. The $\mathcal{P}$.Revoke operation takes as input a certificate ψ and outputs whether the revocation was successful or not, i.e.: $\mathcal{P}.\mathsf{Revoke}(st, clk, \psi) \rightarrow (st, \top/\bot)$. For example, $\mathcal{P}$.Revoke may fail if attempting to revoke an already expired or revoked certificate, or a certificate not issued by this issuer.

Most PKIs use a *non-revocation* mechanism to allow relying parties to verify that a certificate was not revoked at a given time t. X.509 defines two non-revocation mechanisms, *certificate revocation lists (CRLs)* and the *online certificate status protocol (OCSP)* allowing relying parties to verify non-revocation status of a certificate by obtaining a CRL or OCSP response valid at time t.

2.4 Certificate Validity

Each PKI has a criteria to determine whether a given certificate is valid or not. As an example of such criteria, consider a PKI where a certificate ψ is *valid* at time t, if (1) t is between $\psi.from$ and $\psi.to$, (2) ψ was certified by $\psi.issuer$. But, even such straightforward and intuitive criteria has some important subtleties.

In particular, how can we determine if ψ was *really* certified by $\psi.issuer$? In a simple setting, the validating party knows the validation key of $\psi.issuer$, typically by having the self-signed key of $\psi.issuer$ in *store*, the set of trusted certificates at the validating party, which includes *trusted root CAs'* self-certificates. The set *store*, used to establish the trust, is an input to the validation function; *store* is often referred to as the *root store* or *trust anchor*. The sequence of certificates $\xi \equiv \psi_1 - \psi_2 - \ldots - \psi_r$ used to validate ψ, terminating in a certificate $\psi_r \in store$, is called a *certificate chain* or a *chain of trust*.

However, in a more practical setting, $\psi.issuer$ is not a trusted root CA, and therefore, will not exist in *store*. Instead, the trust in the public key of $\psi.issuer$ is established using a certificate for $\psi.issuer$, which should also be: (1) valid, and in particular, (2) signed by a trusted CA. This trusted CA "gains trust" either because its self-certified certificate appears in *store*, or because it is certified by a different trusted CA (and so on).

A PKI may have additional requirements for considering a certificate to be valid. PKIs often require some form of certification that indicates non-revocation, e.g., certificate ψ is not included in a CRL ψ_{CRL} valid at time t. To facilitate such additional requirements, the validation function accepts also an *auxiliary input aux*. A PKI implementation would define the structure and content of *aux*, based on its validity criteria. For example, in PKIX (§4.2) a certificate is considered valid only together with *aux* containing a valid certificate chain and certificate(s) of non-revocation.

Formally, the validity of a certificate ψ is determined using the stateless *certificate validation predicate* of the PKI, namely, $\mathcal{P}.\mathsf{Valid}(\psi, t, store, aux)$, where the inputs include the certificate ψ, the time t, the trust-anchor *store* and the auxiliary information *aux*. When $\mathcal{P}.\mathsf{Valid}(\psi, t, store, aux) = \top$, this means that $\mathcal{P}$.Valid managed to establish trust from ψ to an entity with a self-signed certificate in *store*. This trust is typically in the form of a sequence of certificates which are chained together. To obtain the certificates used to validate ψ, the stateless operation $\mathcal{P}$.VCerts can be used; in

Table 2: Common certificate fields.

Field	Description	Encoding in X.509
$\psi.issuer$	The entity that issued the certificate.	*distinguished name* in the *Issuer* certificate field. X.509 certificates may also include an *Issuer Alternative Name (IAN)* extension that may include additional identifiers for the issuer, e.g., as a DNS name.
$\psi.from$	The date and time at which the certificate becomes valid.	*notBefore* entry of the *Validity* certificate field.
$\psi.to$	The date and time at which the certificate should expire (become invalid).	*notAfter* entry of the *Validity* certificate field.
$\psi.serial$	A number that uniquely identifies the certificate.	*serialNumber* certificate field. In X.509, it must be positive and unique among the rest of the certificates issued by the issuing CA.
$\psi.subject$	An identifier for the subject of ψ.	*distinguished name* in the *Subject* certificate field. X.509 certificates may also include a *Subject Alternative Name (SAN)* extension, that may include additional identifiers for the subject, e.g., as a DNS name.
$\psi.pk$	A public key certified as the public key of the subject.	Part of the *Subject Public Key Info* certificate-field.
$\psi.is\_CA$	Whether the subject is a CA, i.e., authorized to issue public key certificates.	Part of the *Basic Constraints* extension.
$\psi.type$, $\psi.PKIadded$	The specific type of certificate, e.g., public key, and a set of fields added to the certificate by the PKI.	These fields are not defined in X.509; see discussion in §2.1.
$\psi.tbc$	The set of all (name, value) fields in ψ	This is a notation, not a field.

particular, if $\mathcal{P}$.VCerts is executed on the same inputs as $\mathcal{P}$.Valid, then the output of $\mathcal{P}$.VCerts should include information sufficient to validate ψ using $\mathcal{P}$.Valid.

2.5 Definition of a PKI Scheme

Definition 1 (PKI scheme). *A PKI scheme $\mathcal{P}$ is a set containing (at least) the following PPT algorithms:*

- $\mathcal{P}$.Init$(st, clk, params) \rightarrow (st, \psi)$: *Takes as input the state*[5] *st, local clock clk, and parameters $params$, and returns the initialized local state st and a self-certified certificate ψ after performing initialization based on the input parameters $params$ and time clk. A returned value $\psi = \perp$ indicates failure to initialize.*
- $\mathcal{P}$.Certify$(st, clk, tbc, aux) \rightarrow (st, \psi, aux)$: *Takes as input the state st, local clock clk, a set of (name, value) fields to be certified tbc, and auxiliary information*[6] *aux, and returns an updated state st, a string ψ, and auxiliary information aux. The string ψ may be a valid certificate, information which may be used to create a valid certificate, or a failure indicator $\perp$.*
- $\mathcal{P}$.Revoke$(st, clk, \psi) \rightarrow (st, \top/\perp)$: *Takes as input the state st, local clock clk, and a certificate ψ, and returns an updated state st and $\top$ if ψ was revoked successfully or $\perp$ if the revocation failed.*
- $\mathcal{P}$.Valid$(\psi, t, store, aux) \rightarrow (\top/\perp)$: *This (stateless) algorithm takes as input a certificate ψ, time t, a root store $store$, and auxiliary information aux, and returns either $\top$ or $\perp$.*
- $\mathcal{P}$.VCerts$(\psi, t, store, aux) \rightarrow (\{\psi_i\})$: *This (stateless) algorithm takes as input the same input as in* $\mathcal{P}$.Valid. *If ψ is valid w.r.t. t, $store$, and aux, then the algorithm returns a set containing information sufficient to validate ψ, typically certificates $\{\psi_i\}$; otherwise, the algorithm returns the empty set $\emptyset$.*

PKI schemes might include additional inputs or operations. In particular, we next define *transparent* PKI schemes; such schemes have additional operations used to ensure that valid certificates are publicly available (i.e., "transparent"), allowing to detect discrepancies and suspect certificates. Specifically, transparent PKIs require the ability to instruct monitors to start the monitoring process of a given log ($\mathcal{P}$.Monitor), the ability to retrieve what is known to a monitor regarding a given subject ($\mathcal{P}$.Lookup), and the ability to identify logs which, based on the local knowledge of a monitor, should include a given certificate but do not include it ($\mathcal{P}$.Audit). In addition, in a transparent PKI scheme, the output of $\mathcal{P}$.VCerts for a given certificate ψ should also include the log certificates of the logs which should include ψ.

Definition 2 (Transparent PKI scheme). *A transparent PKI scheme $\mathcal{P}$ is a PKI scheme with the following additional PPT algorithms:*

- $\mathcal{P}$.Monitor$(st, clk, \psi_L) \rightarrow st$: *Takes as input state st, local clock clk and certificate ψ_L, and returns an updated state st, and starts to monitor (certificates* logged *in) the log corresponding to certificate ψ_L.*
- $\mathcal{P}$.Lookup$(st, clk, subject) \rightarrow (st, \Psi)$: *Takes as input state st, local clock clk, and an identifier $subject$, and returns an updated state st and a set $\Psi = \{(\psi_1, aux_1), (\psi_2, aux_2), ...\}$ of all pairs (ψ_i, aux_i) known to the entity s.t. ψ_i is the certificate of the given $subject$, i.e., $\psi_i.subject = subject$, and aux_i is the corresponding auxiliary information for ψ_i.*
- $\mathcal{P}$.Audit$(st, \psi, aux) \rightarrow \{\psi_L\}$: *Takes as input state st, certificate ψ, and auxiliary information aux, and returns a set containing log certificates known to the monitor for logs, which, to the monitor's knowledge, should include ψ but do not.*

3 PKI REQUIREMENTS

Formally defined requirements are necessary to prove whether a given PKI implementation meets specific requirements, and if so, under what assumptions. In §3.1, we briefly discuss the MoSS framework [17], which we use to define the PKI requirements. Then, we define the following requirements: Existential Unforgeability (§3.2), Accountability (§3.3), Guaranteed and Accountable Δ-Revocation (§3.4), and HL, Audited and Guaranteed Δ-Transparency (§3.5).

3.1 Modular Security Specifications

PKI schemes, deployed and proposed, vary greatly in their designs, operate under different models (assumptions) and aim to satisfy different requirements (goals). To define requirements which apply to different PKI schemes, even if they assume different models, we

[5]The state st is given in the input to Init (and other stateful operations) for a technical reason. We use the *Modular Security Specifications (MoSS) Framework* [17] to define PKI specifications (see §3.1). In MoSS, operations receive the state st as input. The Init operation is invoked during the MoSS execution, and, therefore, receives the state st as input, like other operations.

[6]The auxiliary information may include e.g. certificate chains and CRLs or other information used for certificate validation.

use the *Modular Security Specifications (MoSS) Framework* [17] for our specifications. MoSS separates the definition of *requirements* from the *models*, simplifying the definition of requirements and allowing for the evaluation of requirements satisfied by PKI schemes under different models. For example, we show that CT does not provide Guaranteed Δ-Transparency and only a weaker notion, HL Δ-Transparency (Table 1).

To illustrate the importance of separating models from requirements, consider this *simplified* Δ-Revocation requirement: a certificate revoked at time t by an honest CA would be considered invalid at any time after $t + \Delta$ (Definition 3). The delay Δ can be a function of network delays, clock bias and design decisions (e.g., periodicity of issuing CRLs). Satisfaction of the Δ-Revocation requirement depends on the clock synchronization and network/communication delay models.

This work follows the MoSS framework [17]; we briefly recall here concepts required for this work. MoSS defines an execution process $\mathbf{Exec}_{\mathcal{A},\mathcal{P}}(params)$ for a given adversary algorithm $\mathcal{A}$ and protocol $\mathcal{P}$. The execution consists of a series of events. In each event e, the adversary decides on the entity invoked $ent[e]$, the operation $opr[e]$, the input $inp[e]$, the real-time clock $\tau[e]$ and the local clock $clk[e]$ of entity $ent[e]$; and the adversary receives the output $out[e]$ of the protocol.[7] When the execution loop of protocol $\mathcal{P}$ with parameters $params$ and adversary $\mathcal{A}$ is terminated (by the adversary), the execution outputs a *transcript* (trace), denoted $T \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(params)$. The transcript contains the set of entities $T.\mathsf{N}$, the set of faulty entities $T.\mathsf{F}$ as identified by the adversary, the adversary's output $T.out_{\mathcal{A}}$, and the values of all inputs and outputs for each event, e.g., $T.ent[e]$.

We use the extended execution process[8] from [17], which allows three entity-corruption operations: *Get-state* (exposing the state of the entity), *Set-state* (setting the state of the entity) and *Set-output* (causing the entity to output a specified value).

To define assumptions and restrict the adversary, MoSS uses *model predicates*, which are computed over the execution transcript. For example, Algorithm 1 shows the $\pi^{\text{Drift}}_{\Delta_{clk}}$ model predicate, which ensures bounded clock drift (and may help to understand why we use both $clk[e]$ and $\tau[e]$). This is one of the models (assumptions) from [17] that we use in our security analysis (§5).

Algorithm 1 $\pi^{\text{Drift}}_{\Delta_{clk}}$ Model Predicate (from [17])

```
1: return $\forall \hat{e} \in \{1, \ldots, T.e\}$:                          ▷ For each event
2:     $|T.clk[\hat{e}] - T.\tau[\hat{e}]| \leq \Delta_{clk} \wedge$     ▷ Local clock within $\Delta_{clk}$ drift from real time
3:     if $\hat{e} \geq 2$ then $T.\tau[\hat{e}] \geq T.\tau[\hat{e} - 1]$   ▷ Monotonically increasing real time
```

An adversary $\mathcal{A}$ *satisfies* model predicate[9] π, if for every protocol $\mathcal{P}$ it holds that $\Pr[\pi(\mathbf{Exec}_{\mathcal{A},\mathcal{P}}(params)) = \bot] \in Negl(|params|)$, i.e., there is a negligible probability that the transcript[10] of a random execution of protocol $\mathcal{P}$ with adversary $\mathcal{A}$ will fail to satisfy π.

Similarly, MoSS uses *requirement predicates*, also computed over the execution transcript T, to define the requirements for a PKI scheme. We define several requirements in the rest of this section, e.g., the Accountability requirement (Algorithm 3) and the three Δ-Transparency requirements (Definition 4). A PKI scheme $\mathcal{P}$ *satisfies requirement predicate* $\pi_{\mathcal{R}}$ *assuming model predicate* $\pi_{\mathcal{M}}$, if the probability that $\pi_{\mathcal{R}}(\mathbf{Exec}_{\mathcal{A},\mathcal{P}}(params)) = \bot$ is negligible, where $\mathbf{Exec}_{\mathcal{A},\mathcal{P}}(params)$ is the transcript of a random execution of $\mathcal{P}$ interacting with any PPT adversary $\mathcal{A}$ that satisfies $\pi_{\mathcal{M}}$.

3.2 Existential Unforgeability Requirement

The first requirement for PKI we define is *Existential Unforgeability* (π_{EUF}, Algorithm 2), which is slightly more complex than Existential Unforgeability for signature schemes. We might think of Existential Unforgeability for PKI as the requirement that a valid certificate can only be generated through a proper use of the 'Certify' operation. That is, an adversary cannot generate (with non-negligible probability) a valid certificate with respect to a public key pk without having access to the corresponding private key sk of a subject. However, this does not capture the primary purpose of PKI schemes, which is to establish trust in public keys, in particular, in the corresponding public key pk of the subject. Consider an attacker that generates a *fraudulent yet valid* certificate, i.e., a certificate for a public key $pk' \neq pk$, i.e., is not the public key pk generated and self-certified by the subject. Such attacker does not need access to the 'real' subject's private key sk, because it can simply falsely claim that pk' is a public key of the subject. This means that an attacker may be able to issue a valid certificate without proper use of the 'Certify' operation by the subject.

Therefore, intuitively, for a PKI scheme to satisfy Existential Unforgeability, it means that for every certificate ψ_0, either:

(1) ψ_0 is invalid (Line 3) for the given time t, root store *store* and auxiliary information *aux*, or
(2) the 'Certify' operation of the PKI scheme was used at the entity $\psi_0.issuer$ to certify the non-PKIadded fields of ψ_0 (Line 4), which implies that the adversary may have gotten either ψ_0 or another certificate with the same non-PKIadded fields as ψ_0 correctly from $\psi_0.issuer$ using the 'Certify' operation, or
(3) $\psi_0.issuer$ is not a benign entity, e.g., it is a rogue entity (Line 5), which implies that the issuer may have generated certificates without correctly using the 'Certify' operation, or
(4) the validation of ψ_0 uses a valid fraudulent certificate ψ_1 for the issuer of ψ_0 (Line 6), which implies that the private key corresponding to $\psi_1.pk$ may be known to or controlled by the adversary.

To be able to identify such a fraudulent public key certificate ψ_1, we require PKI schemes to follow two conventions[11]:

[7] Since the adversary and protocol may be invoked many times during the execution, there is the issue of whether their total runtimes will be polynomial if they are PPT algorithms. [17] discusses this issue and provides an approach to ensure that the runtimes are polynomial.

[8] In [17], these operations are included in the set X, which is specified as part of the parameters of the execution process.

[9] MoSS [17] defines more general models and requirements, which supports non-negligible probability β of failures for some predicates; this is not required for our models and requirements.

[10] We consider params to be part of the execution transcript instead of giving it as a separate input parameter to predicates, as in [17].

[11] The conventions do not require changes to existing PKI implementations.

(1) Whenever a benign entity ι generates a public key pk, it would output a pair (*'SelfCert'*, ψ) where the issuer and subject fields are both ι, i.e., $\psi.issuer = \psi.subject = \iota$, and $\psi.pk = pk$. We refer to ψ as a *self-certificate*.
(2) The PKI supports a function $\mathcal{P}$.VCerts which takes the same input as $\mathcal{P}$.Valid, and when a certificate ψ_0 is found valid by $\mathcal{P}$.Valid, then $\mathcal{P}$.VCerts should return the set of certificates used by $\mathcal{P}$.Valid to validate ψ_0. These certificates may include certificates from *aux* and from *store*.

We use the first convention in the SelfCert function (Lines 22-27). We utilize the second convention and SelfCert in CanFindFraudIssuerCert (Lines 15-21) to find, in the output of $\mathcal{P}$.VCerts, a certificate ψ_1 which certifies a fraudulent public key for $\psi_0.issuer$.

Algorithm 2 π_{EUF}: Existential Unforgeability Requirement

```
    Input: execution transcript $T$.
    Output: $\bot$ if the adversary wins and $\top$ if the adversary failed. To win, the adversary needs to
    present a valid yet forged certificate $\psi_0$, i.e., a valid certificate where $\psi_0.issuer$ is benign, $\psi_0$
    was not certified by $\psi_0.issuer$, and the validation of $\psi_0$ w.r.t a valid fraudulent certificate for
    the issuer of $\psi_0$.
 1: procedure $\pi_{\mathsf{EUF}}(T)$
 2:     $(\psi_0, t, store, aux) \leftarrow T.out_{\mathcal{A}}$                       ▷ Adversary's output
 3:     if $\neg\mathcal{P}$.Valid$(\psi_0, t, store, aux) \lor$                       ▷ If $\psi_0$ invalid, or
 4:         FieldsCertified$(T, \psi_0, \psi_0.issuer) \lor$                          ▷ fields were certified at issuer, or
 5:         $\psi_0.issuer \notin T.\mathsf{N} - T.\mathsf{F} \lor$                   ▷ issuer is rogue, or
 6:         CanFindFraudIssuerCert$(T, \psi_0, t, store, aux)$                        ▷ there is a fraudulent issuer cert,
 7:         then return $\top$                                                        ▷ then adversary failed (EU holds)
 8:         else return $\bot$                                                        ▷ otherwise, adversary wins
 9: end procedure
10: procedure FieldsCertified$(T, \psi, ent)$
11:     return $\exists e$ s.t. $T.opr[e] =$ 'Certify' $\land$                      ▷ If 'Certify' operation was executed
12:             $T.inp[e] = \left\{ (field, val) \;\middle|\; \begin{matrix} \exists (field, val) \in \psi.tbc \land \\ field \notin \psi.PKIadded \end{matrix} \right\} \land$
                                                                                      ▷ on non-PKIadded fields of $\psi.tbc$
13:             $T.ent[e] = ent$                                                      ▷ by entity $ent$
14: end procedure
15: procedure CanFindFraudIssuerCert$(T, \psi_0, t, store, aux)$
16:     $vcerts \leftarrow \mathcal{P}$.VCerts$(\psi_0, t, store, aux)$              ▷ Certificates used for validating $\psi_0$
17:     return $\exists \psi_1 \in vcerts$ s.t.                                      ▷ Contain a certificate $\psi_1$ such that
18:             $\left[ \begin{matrix} \mathcal{P}.\mathsf{Valid}(\psi_1, t, store, aux) \lor \\ \left[ \begin{matrix} \psi_1 \in store \land \\ \psi_1.subject = \psi_1.issuer \end{matrix} \right] \end{matrix} \right] \land$
                                                                                      ▷ $\psi_1$ is a valid certificate, or
                                                                                      ▷ $\psi_1$ is in store and
                                                                                      ▷ has same subject and issuer
19:             $\psi_1.subject = \psi_0.issuer \land$                                ▷ The subject of $\psi_1$ is the issuer of $\psi_0$
20:             $\neg$SelfCert$(T, \psi_0.issuer, \psi_1.pk)$                         ▷ $\psi_1$ is fraudulent
21: end procedure
22: procedure SelfCert$(T, \iota, pk)$
23:     return $\exists e, \psi$ s.t. $T.ent[e] = \iota \land$                       ▷ Entity $\iota$
24:             $T.out[e] =$ ('SelfCert', $\psi$) $\land$                            ▷ outputted certificate $\psi$
25:             $\psi.pk = pk \land$                                                  ▷ certifying given pk
26:             $\psi.issuer = \psi.subject = \iota$                                  ▷ using the same identity
27: end procedure
```

3.3 Accountability Requirement

We now define the *Accountability* requirement (Algorithm 3). Defining accountability turned out to be more complex than we initially anticipated. Intuitively, for a PKI scheme to satisfy accountability, an accountable root CA should be identifiable for every valid certificate ψ_0. Let us explain what we mean by an accountable root CA and why identifying such a root CA is necessary.

Intuitively, if $\psi.issuer$ has issued (certified) the certificate ψ, it should be considered accountable for ψ. While this is typically the case, recall from §3.2 that ψ may have been certified by an attacker using a *fraudulent yet valid* certificate $\hat{\psi}$, typically where $\hat{\psi}.subject = \psi.issuer$, where $\hat{\psi}.pk$ may not be the self-certified key of $\hat{\psi}.subject$. In fact, there may even be no 'real entity' called $\psi.issuer$ if ψ if fraudulently issued; in particular, certificates may contain an empty $\psi.issuer$[12].

We say that a valid certificate ψ' is *accountable* for a certificate ψ if the 'Certify' operation of the PKI scheme was used at $\psi'.subject$ to certify the non-PKIadded fields of ψ, or to certify those fields *of another certificate* $\hat{\psi}$ which is accountable for ψ. Note that we exclude the PKIadded fields; this is since they are added by the Certify operation itself rather than part of the input fields being certified.

The Accountability requirement requires that for any valid certificate ψ_0, the output of $\mathcal{P}$.VCerts for ψ_0 contains at least one root certificate; and for every root certificate ψ_r in the output of $\mathcal{P}$.VCerts for ψ_0, either:

(1) ψ_r is accountable for ψ_0 (Line 7), or
(2) ψ_r is a bad certificate (Line 8), where a *bad* certificate is a certificate which has a rogue subject or whose public key was not self-certified by its subject, or
(3) ψ_r is accountable for a valid bad certificate ψ' (Line 9),

By identifying an accountable root certificate, ψ_r, a relying party can respond to a problematic-yet-valid certificate, ψ. For example, a relying party may hold the root CA, ι_R, which issued ψ_r, responsible for damages caused by treating ψ as valid. If ι_R does not take corrective action, such as providing compensation, the relying party may stop trusting ι_R and remove ι_R from the *root CA store*.

Note that we do not require the identification of non-root accountable CAs, as a relying party may not be able to take action upon identifying a non-root accountable CA, and it may be impossible to reliably identify an accountable non-root CA.

3.4 Revocation Requirements

Certificates sometimes need to be revoked prior to their expiration date. Revocation is initiated by the issuer for various reasons, including the loss or compromise of the private key corresponding to a certified public key, or when the certificate contains incorrect or outdated information.

It takes time to communicate the revocation of a certificate to the relying parties; we use Δ to denote the maximum allowed time. The 'grace period' Δ is typically required for three reasons: (1) the bias between the clock of $\psi.issuer$ and the clock of the relying party validating ψ, (2) the communication delay from $\psi.issuer$ to the relying parties, and (3) the time for any proof that ψ is non-revoked which was issued prior to its revocation to become stale, i.e., expire.

Guaranteed Δ-Revocation and the Zombie certificate attack. Preferably, we would like revocation to be fully enforced, i.e., if the issuer

[12]The identification of the corresponding issuer's certificate may be done using the authority and subject key identifier extensions, see [23, 41].

Algorithm 3 π_{ACC}: Accountability Requirement

Input: execution transcript T.

Output: $\bot$ if the adversary wins and $\top$ if the adversary failed. To win, the adversary needs to present a valid certificate ψ_0 where the output of $\mathcal{P}$.VCerts does not include a bad root certificate, nor a root certificate which is accountable for a valid bad certificate among the certificates in the output of $\mathcal{P}$.VCerts, nor a root certificate which is accountable for ψ_0.

```
1: procedure $\pi_{\text{ACC}}(T)$
2:     $(\psi_0, t, store, aux) \leftarrow T.out_{\mathcal{A}}$                ▷ Adversary's output
3:     if $\neg\mathcal{P}$.Valid$(\psi_0, t, store, aux)$ then return $\top$   ▷ If $\psi_0$ is invalid, then adversary failed (Acc. holds)
4:     $vcerts \leftarrow \mathcal{P}$.VCerts$(\psi_0, t, store, aux)$
5:     if $store \cap vcerts \neq \emptyset \wedge$                     ▷ If vcerts contains at least one root certificate, and
6:         $\forall \psi_r \in store \cap vcerts$ :                         ▷ for every root cert $\psi_r$ in vcerts:
7:             Accountable$(T, \psi_0, \psi_r, vcerts) \vee$             ▷ $\psi_r$ is accountable for $\psi_0$ (see Line 13), or
8:             BadCert$(T, \psi_r) \vee$                                ▷ $\psi_r$ is bad (see Line 17), or
9:             [ $\exists \psi' \in vcerts$ s.t.                          ▷ a certificate $\psi'$ in vcerts exists s.t.
                 $\mathcal{P}$.Valid$(\psi', t, store, aux) \wedge$       ▷ $\psi'$ is valid, and
                 BadCert$(T, \psi') \wedge$                             ▷ $\psi'$ is bad (see Line 17), and
                 Accountable$(T, \psi', \psi_r, vcerts)$ ]              ▷ $\psi_r$ is accountable for $\psi'$ (see Line 13)
10:        then return $\top$                                         ▷ then adversary failed (Acc. holds)
11:        else return $\bot$                                         ▷ otherwise, adversary wins
12: end procedure
13: procedure Accountable$(T, \psi, \psi', vcerts)$
14:     return FieldsCertified$(T, \psi, \psi'.subject) \vee$          ▷ Fields of $\psi$ were certified at $\psi'.subject$ (see Alg. 2), or
15:         [ $\exists \psi'' \in vcerts$ s.t.                           ▷ a certificate $\psi''$ in vcerts exists s.t.
              Accountable$(T, \psi, \psi'', vcerts) \wedge$              ▷ $\psi''$ is accountable for $\psi$, and
              FieldsCertified$(T, \psi'', \psi'.subject)$ ]              ▷ Fields of $\psi''$ were certified at $\psi'.subject$ (see Alg. 2)
16: end procedure
17: procedure BadCert$(T, \psi)$
18:     return $\psi.subject \notin T.\text{N} - T.\text{F} \vee$            ▷ $\psi.subject$ is not benign, or
19:         $\neg$SelfCert$(T, \psi.subject, \psi.pk)$                  ▷ $\psi$ is fraudulent (see Line 22 in Alg. 2)
20: end procedure
```

revokes a certificate at time t, then no relying party should accept the certificate after $t+\Delta$. We refer to this property as *Guaranteed Δ-Revocation.* This ensures that revocation by a benign CA is always effective, with at most Δ delay; a relying party cannot be misled to rely on a revoked certificate after $t + \Delta$.

The most well-known revocation mechanism is the X.509 standard *Certificate Revocation List (CRL)* mechanism, the 'basic' revocation mechanism of PKIX and CT; see §4.2 or [41] for details. The CRL is a timestamped list of revoked certificates signed by their issuer. CRLs are issued periodically, say once every Δ seconds. A relying party R considers certificate ψ as non-revoked at time t, if R received a CRL from the issuer of ψ, issued at $t - \Delta$ or later, which does not list ψ as a revoked certificate.

The PKIX specifications [41] require the CRL to be validated using a valid public key certificate ψ' issued to the issuer of ψ, i.e., $\psi'.subject = \psi.issuer$. Additionally, they require that the trust anchor for the certification path of ψ' be the same as the trust anchor used to validate ψ. However, as we next show, the CRL mechanism *fails to ensure Guaranteed Δ-Revocation*; we later define the weaker *Accountable Δ-Revocation* property, which the CRL mechanism ensures.

Claim 1 (The Zombie certificate attack). *PKIX and CT, using the CRL revocation mechanism, fail to ensure the Guaranteed Δ-Revocation requirement.*

Proof: Let ψ_0 be a certificate issued by a benign CA $\psi_0.issuer$, with a validity period $[\psi_0.from, \psi_0.to]$, and revoked by $\psi_0.issuer$ at time t_R, where $\psi_0.from < t_R < \psi_0.to - \Delta$. Assume that $\psi_0.issuer$ is not a root CA; therefore, for ψ_0 to be valid, it must have come with a certificate ψ_1 for $\psi_0.issuer$. Consider the case where $\psi_1.issuer$ is a rogue CA. Then, it can issue another certificate ψ_1' s.t. $\psi_1'.subject = \psi_1.subject = \psi_0.issuer$; in fact, ψ_1' is identical to ψ_1, except for having a different public key. Specifically, $\psi_1'.pk$ is chosen by the adversary, i.e., the adversary knows the corresponding private certification key. The adversary now uses this key to sign a (fake) CRL where ψ_0 appears as non-revoked at time $t_R + \Delta$. Using this CRL, ψ_0 appears still valid at $t_R + \Delta < \psi_0.to$. Hence, the Guaranteed Δ-Revocation requirement is not satisfied. □

Accountable Δ-Revocation. This requirement *allows* ψ to be considered valid even after being revoked by its benign issuer CA $\psi.issuer$, provided that the PKI can identify a different certificate ψ_R for a benign subject, which is either a fraudulent root certificate or a fraudulent-yet-valid certificate. That is, the benign subject $\psi_R.subject$ did not output a corresponding self-signed certificate for the public key $\psi_R.pk$. For example, in the Zombie certificate attack of Claim 1, we will have $\psi_R = \psi_1'$. To validate ψ after its revocation, using the fake CRL, the attacker must include $\psi_1' = \psi_R$; therefore, we can identify the rogue issuer ι_R. If the PKI can identify such a the rogue certificate ψ_R, and assuming that the PKI ensures *Accountability*, then we can identify a root CA which is either accountable for ψ_R, is a bad certificate, or is accountable for a valid bad certificate (see Section 3.3).

The Guaranteed Δ-Revocation requirement is stronger, as it prevents the use of revoked certificates. In contrast, Accountable Δ-Revocation only provides accountability after an attack has occurred. Notably, the stipulation in [41] that the trust anchor for the certification path of ψ' must be the same as the trust anchor used to validate ψ underscores this point. This requirement would not be necessary if the designers were only aiming for Accountable Δ-Revocation. We believe this indicates a clear intention to *prevent* the use of revoked certificates (Guaranteed Δ-Revocation), rather than merely offering post-attack accountability (Accountable Δ-Revocation). Importantly, ensuring Guaranteed Δ-Revocation is indeed feasible for PKI. For instance, PKIX could have mandated the validation of the CRL using the same public key that is used for validating the certificate itself. More generally, a PKI could require that a 'certificate of non-revocation' (such as a CRL or an OCSP response) must be validated using the same public key as that is used to validate the certificate itself. Note that using the same key to validate the certificate of non-revocation has the disadvantage that if this key is revoked, then the certificate of non-revocation can no longer be validated.

We define both the Guaranteed and Accountable Δ-Revocation requirement predicates in Definition 3. In §5, we show that PKIX and CT, with the CRL revocation mechanism, ensure Accountable Δ-Revocation.

Definition 3 (Δ-Revocation requirements). *We define the Guaranteed and Accountable Δ-Revocation predicates as:*

$$\pi_{\Delta}^{GtdRev}(T) = \left\{ \begin{array}{l} \top \text{ iff } f_{\Delta}^{\mathsf{Rev}}(T) = \text{'}G\text{'}, \\ \bot \text{ otherwise} \end{array} \right\}$$

$$\pi_{\Delta}^{AccRev}(T) = \left\{ \begin{array}{l} \top \text{ iff } f_{\Delta}^{\mathsf{Rev}}(T) \in \{\text{'}G\text{'}, \text{'}A\text{'}\}, \\ \bot \text{ otherwise} \end{array} \right\}$$

Algorithm 4 defines the function $f_{\Delta}^{\mathsf{Rev}}$. □

Algorithm 4 $f_{\Delta}^{\mathsf{Rev}}$: Δ-Revocation Function

```
Input: transcript $T$.
Output: 'G' if the adversary fails and $\Delta$-Revocation is guaranteed, 'A' if $\Delta$-Revocation is only accountable, and $\bot$ if the adversary wins.
1: procedure $f_{\Delta}^{\mathsf{Rev}}(T)$
2:     $(\psi, t, store, aux) \leftarrow T.out_{\mathcal{A}}$                ▷ Extract adversary's output
3:     if [ $\mathcal{P}$.Valid$(\psi, t, store, aux) \wedge$                 ▷ $\psi$ is valid
            $\psi.issuer \in T.\mathsf{N} - T.\mathsf{F} \wedge$               ▷ $\psi$'s issuer is benign
            CertifyAndRevokeRequested$(T, \psi, t, \Delta)$ ]        ▷ See [57]
4:         if CanFindFraudulentCert$(T, \psi, t, store, aux)$        ▷ See [57]
5:             then return 'A'   ▷ $\psi$ is 'revoked-yet-valid', but fraudulent cert was detected
6:             else return $\bot$   ▷ Adversary wins
7:     else return 'G'   ▷ Adversary fails (Guaranteed revocation)
8: end procedure
```

3.5 Transparency Requirements

Accountability, as described in §3.3, is a reactive defense, whose goal is to *deter* rogue or negligent behavior by root CAs. For many years, this reactive measure was seen as sufficient under the assumption that root CAs, and CAs certified by them, are respectable and trustworthy entities who would not risk being implicated in issuing rogue certificates, intentionally or otherwise. However, repeated cases of rogue certificates issued by compromised or dishonest CAs, have proven this assumption to be overly optimistic. Punishing root CAs is non-trivial: beyond negative publicity, punishment has often been ineffective [47, 21]. Furthermore, punishing a CA requires that a rogue certificate is *discovered*. An attacker could reduce the risk of discovery by minimizing the exposure of the rogue certificate. Efforts such as Perspectives Project [55] and the EFF SSL Observatory [14] provide some assistance in discovering rogue certificates but are not sufficient to address this concern.

This motivated *transparent* PKI designs, most notably, the standardized and deployed *Certificate Transparency (CT)* [28, 46]. To support transparency, a certificate must be logged at one or more *loggers*. Loggers are parties committed to including certificates in a *public log* they maintain, and to making this log available to third parties called *monitors*. Each monitor keeps tabs on the certificates logged by one or, usually, more loggers; in addition, monitors may detect suspicious certificates and inform interested parties, such as domain owners and relying parties.

In other words, transparency aims to prevent a CA from stealthily generating a rogue certificate ψ to attack select victims. Transparency facilitates early detection of rogue certificates issued by a corrupt, compromised or negligent CA. By demanding that a valid certificate must be transparent, we ensure the detection of rogue and suspicious certificates. There is some unavoidable delay from the time a certificate is submitted to a log until the relevant monitors are aware of it. This delay is primarily due to the significant time allowed between receiving a certificate and including it in a new version of the log.

The Δ-Transparency Requirements. Similarly to the case with revocation, we found that the CT standard [46] does not satisfy the natural, strong and *guaranteed* transparency. *Guaranteed-Transparency* means that an honest monitor which is monitoring the relevant logs is always aware of a valid certificate[13] after a specific delay.

CT, however, satisfies only a weaker notion, *HL Δ-Transparency*, where transparency only holds if a certificate is logged by at least one benign logger. The Chrome implementation of CT ([24], [57]) also ensures another weak notion of transparency, *Audited Δ-Transparency*. In this case, an honest monitor which is monitoring the relevant (possibly all corrupt) logs might be unaware of a valid certificate[14], but when presented with such a certificate during an *audit*, the monitor outputs a log certificate of a corrupt logger responsible for this lack of transparency.

Definition 4 (Δ-Transparency requirements). *We define the Guaranteed, HL and Audited Δ-Transparency predicates as:*

$$\pi_{\Delta}^{GtdTra}(T) = \left\{ \begin{array}{l} \top \text{ iff } f_{\Delta}^{\mathsf{Tra}}(T, \bot) = \text{'}G\text{'} \\ \bot \text{ otherwise} \end{array} \right\}$$

$$\pi_{\Delta}^{HLTra}(T) = \left\{ \begin{array}{l} \top \text{ iff } f_{\Delta}^{\mathsf{Tra}}(T, \top) = \text{'}G\text{'} \\ \bot \text{ otherwise} \end{array} \right\}$$

$$\pi_{\Delta}^{AudTra}(T) = \left\{ \begin{array}{l} \top \text{ iff } f_{\Delta}^{\mathsf{Tra}}(T, \bot) \in \{\text{'}G\text{'}, \text{'}A\text{'}\}, \\ \bot \text{ otherwise} \end{array} \right\}$$

Algorithm 5 defines the function $f_{\Delta}^{\mathsf{Tra}}$. □

Algorithm 5 $f_{\Delta}^{\mathsf{Tra}}$: Δ-Transparency Function

```
Input: transcript $T$ and the HL flag.
Output: 'G', if $\Delta$-Transparency is guaranteed; if the HL input flag is set, then we require that at least one honest log should include $\psi$. 'A', if $\Delta$-Transparency is (only) audited. $\bot$ if neither guaranteed nor audited $\Delta$-Transparency holds (adversary wins).
1: procedure $f_{\Delta}^{\mathsf{Tra}}(T, \text{HL})$
2:     $(\psi, t, store, aux, \iota_M) \leftarrow T.out_{\mathcal{A}}$        ▷ Extract adversary's output
3:     $LogCerts \leftarrow \mathcal{P}.\mathsf{VCerts}(\psi, t, store, aux) \cap store.logs$   ▷ Certs of logs for $\psi$
4:     if HL $\wedge$ NoHonestLog$(T, LogCerts)$ return 'G'        ▷ HL w/o honest log (See [57])
5:     if [ $\psi.from \le t - \Delta \wedge$
            $\mathcal{P}$.Valid$(\psi, t, store, aux) \wedge$               ▷ $\psi$ is valid
            $\iota_M \in T.\mathsf{N} - T.\mathsf{F} \wedge$                 ▷ $\iota_M$ is benign
            MonitoringLogs$(T, t - \Delta, \iota_M, LogCerts) \wedge$   ▷ See [57]
            MonitorIsUnaware$(T, \psi, t, \iota_M)$ ]           ▷ See [57]
6:         if Audited$(T, \psi, t, aux, \iota_M)$                 ▷ See [57]
7:             if FailedIdentifyCorrupted$(T, \psi, t, aux, \iota_M, LogCerts)$   ▷ See [57]
8:                 then return $\bot$        ▷ Adversary wins
9:                 else return 'A'         ▷ Audited transparency
10:        return 'G'                      ▷ Adversary fails (Guaranteed transparency)
11: end procedure
```

[13]The monitor may not be aware of *all* the fields of the certificate. In particular, a certificate ψ may include a field $\psi.PKIadded$ listing fields which were added to the certificate by the PKI, possibly after the other fields were logged. The monitor should be aware, however, of the other fields in the certificate (those which are not in $\psi.PKIadded$).

[14]This is a simplification. More precisely, in addition to the fields in $\psi.PKIadded$ of which the monitor may be unaware, the monitor may also be unaware of the other fields in the certificate (those which are not in $\psi.PKIadded$).

4 PROVABLY-SECURE PKI SCHEMES

In this work, we present three implementation of PKI schemes based on two PKIs used in practice: PKIX following [41] (see §4.2) and CT following the CT 2.0 specification [46] (see [57]), using PKIX for aspects not covered in [46]. In addition, we provide a specification of CTwAudit, a variant of CT augmented by an auditing mechanism [24] (see [57]). While this variant is not standardized, we found it important to include, as it is supported as an option by Google Chrome, the most popular browser.

In all implementations, we use the CRL revocation mechanism[15]. The schemes cannot, realistically, cover all aspects of the corresponding (lengthy and not fully specified) RFCs, but we have done our best to retain all aspects related to the security requirements. For example, we include the details of certificate chains and basic constraints, but omit the (less deployed and less relevant) length, name and policy constraints.

4.1 Certificate Scheme

Typical PKI schemes use an encoding scheme Θ to encode the certificate fields (Table 2), and a signature scheme $\mathcal{S}$ to sign the encoded fields. In practice, the PKI schemes use one or more encoding schemes, e.g., BER, DER, PEM, and SPKI [41, 44, 19, 38], and one or more signature schemes, e.g., RSA and ECDSA [40]. We could have specified a PKI implementation with a specific encoding scheme Θ and signature scheme $\mathcal{S}$, and then prove security by reduction to the unforgeability of $\mathcal{S}$; however, this would be inconvenient and inefficient, given the many possible encodings and signature schemes.

Instead, we define an abstract *certificate scheme* C and its existential unforgeability requirement (Definition 5). We then present a simple, generic design of a certificate scheme from a signature scheme $\mathcal{S}$ and an encoding scheme Θ (Definition 6), and prove that it satisfies unforgeability by a reduction to the unforgeability of the signature scheme. Then, we present the implementations of the PKI schemes (PKIX and CT) using any certificate scheme C.

The new certificate scheme abstraction offers several advantages. Firstly, it simplifies the description of PKI schemes. Secondly, it eases the analysis and reductions. Thirdly, analyzing multiple PKI schemes, each using different encoding and signature schemes, would be impractical without this abstraction. Fourthly, it may enable PKI schemes that utilize different constructions than those in Definition 6, such as signatures over digests of accumulators or Merkle trees, allowing validation with only parts of the certified data. Finally, certificate schemes appear useful for applications beyond PKI schemes.

DEFINITION 5 (Certificate scheme). *A* certificate scheme C *is defined as a tuple of PPT algorithms:*

$$C = (\mathsf{KeyGen}, \mathsf{Certify}, \mathsf{Verify}, \mathsf{Decode})$$

where:

- $C.\mathsf{KeyGen}(1^n) \rightarrow (\{0,1\}^*, \{0,1\}^*)$*: Takes as input security parameter* 1^n*, and returns a pair* (sk, pk) *of keys, where* sk *is a private (certification) key and* pk *is a corresponding public verification key.*
- $C.\mathsf{Certify}_{sk}(tbc) \rightarrow \psi$*: Takes as input a set* $tbc = \{(n_i, v_i)\}$ *(*to be certified*) of name-value pairs, where* n_i *is an alphanumeric* field name *and* $v_i \in \{0,1\}^*$ *is a* field value*, and using the private certifying key* sk*, returns* $\psi \in \{0,1\}^*$*. Field names are unique, i.e.,* $(\forall i \neq j)(n_i \neq n_j)$*. We say that* ψ *is a* certificate *of* tbc*.*
- $C.\mathsf{Verify}_{pk}(\psi) \rightarrow \{\top, \bot\}$*: Takes as input a certificate* $\psi \in \{0,1\}^*$*, and using the public verification key* pk*, returns* $\top$ *if the certificate is valid, i.e., was certified using the private cerifying key corresponding to the public key* pk*, and* $\bot$ *otherwise.*
- $C.\mathsf{Decode}(\psi) \rightarrow tbc \cup \{\bot\}$*: Takes as input a certificate* $\psi \in \{0,1\}^*$*, and returns a set* tbc*, of name-value pairs, as defined for the* $C.\mathsf{Certify}$ *algorithm above, or* $\bot$ *(when decoding fails).*

Dot notation. We use dot notation to extract the value of a field from a tbc set, e.g., $tbc.issuer$ denotes the value of the $issuer$ field in tbc. We also use dot notation to denote the value of a field in a certificate's tbc set, e.g., $\psi.issuer$ denotes the value of the $issuer$ field of $tbc \leftarrow C.\mathsf{Decode}(\psi)$.

We say that certificate scheme C ensures *correctness* if for every $(sk, pk) \leftarrow C.\mathsf{KeyGen}(1^n)$ and for every set of name-value pairs tbc, with unique names, the following holds:

(1) $C.\mathsf{Verify}_{pk}(C.\mathsf{Certify}_{sk}(tbc)) = \top$, and
(2) $C.\mathsf{Decode}(C.\mathsf{Certify}_{sk}(tbc)) = tbc$

We say that C ensures *Existential Unforgeability* if for every PPT $\mathcal{A}$, the probability of $\mathcal{A}$ to win in the *Existential Certificate Forgery* game (Algorithm 6), $\Pr[ECF(\kappa, C, \mathcal{A})]$, is negligible in κ. Similarly, we say that C ensures *Multi-key Existential Unforgeability* if for every PPT $\mathcal{A}$, the probability of $\mathcal{A}$ to win in the *Multi-key Existential Certificate Forgery* game (Algorithm 7), $\Pr[MECF(\kappa, C, \mathcal{A})]$, is negligible in κ.

Algorithm 6 Existential Certificate Forgery (*ECF*) Game

```
   $\underline{ECF(\kappa, C, \mathcal{A})}$:
1: $(sk, pk) \leftarrow C.\mathsf{KeyGen}(1^\kappa)$                                  ▷ Generate keys
2: $\psi \leftarrow \mathcal{A}^{C.\mathsf{Certify}_{sk}(\cdot)}(1^\kappa, pk)$       ▷ Give $\mathcal{A}$ oracle access and $pk$
3: $tbc \leftarrow C.\mathsf{Decode}(\psi)$                                          ▷ Decode $tbc$ from $\psi$
4: if $C.\mathsf{Verify}_{pk}(\psi) \wedge$                                          ▷ $\psi$ is valid
5:     $tbc \neq \bot \wedge$                                                        ▷ $tbc$ is not empty
6:     $tbc \notin \{\mathcal{A}$'s inputs to $C.\mathsf{Certify}_{sk}$ oracle$\}$   ▷ $\mathcal{A}$ did not cheat
7:         then return 1                                                             ▷ $\mathcal{A}$ won
8:         else return 0                                                             ▷ $\mathcal{A}$ failed
```

We now define a generic construction $C^{\mathcal{S},\Theta}$ of a certificate scheme from a public-key signature $\mathcal{S}$ and a pair of invertible encoding functions $\Theta = (\Theta_C, \Theta_S)$, both mapping sets of name-value pairs to binary strings. PKIX, CT and other deployed PKI schemes use certificate schemes following this construction. The invertible encoding schemes (Θ_C, Θ_S) are detailed, and composed of different encodings for specific certificate types, making the use of the certificate scheme abstraction and this generic construction essential to understand, analyze and prove security of PKIs.

[15]We chose CRL over other existing revocation mechanisms for its simplicity and the fact that it is not very different from other mechanisms. For example, OCSP introduces the issues of OCSP responders as distinct parties, stapled OCSP and more, which introduces more complexity. While the implementation and analysis can be extended to support additional revocation mechanisms such as OCSP, we leave it for future work.

Algorithm 7 Multi-key Existential Certificate Forgery (*MECF*) Game

```
    $MECF(\kappa, C, \mathcal{A})$:
1:  $sk[\cdot], pk[\cdot] \leftarrow$ global arrays for the KeyGen and Certify oracles
2:  procedure KeyGen($\iota$)                              ▷ KeyGen oracle
3:      $(sk[\iota], pk[\iota]) \leftarrow C.\text{KeyGen}(1^\kappa)$          ▷ Generate key pair
4:      return $pk[\iota]$                                 ▷ Return public key
5:  end procedure
6:  procedure Certify($\iota, tbc$)                        ▷ Certify oracle
7:      return $C.\text{Certify}_{sk[\iota]}(\cdot)$                 ▷ Certify $tbc$ and return cert
8:  end procedure
9:  $(\iota, \psi) \leftarrow \mathcal{A}^{\text{KeyGen}(\cdot),\text{Certify}(\cdot,\cdot)}(1^\kappa)$   ▷ Give $\mathcal{A}$ oracle access and $pk$
10: $tbc \leftarrow C.\text{Decode}(\psi)$                   ▷ Decode $tbc$ from $\psi$
11: if $C.\text{Verify}_{pk[\iota]}(\psi) \wedge$                ▷ $\psi$ is valid
12:     $pk[\iota] \neq \perp \wedge$                          ▷ $pk[\iota]$ is not empty
13:     $tbc \neq \perp \wedge$                               ▷ $tbc$ is not empty
14:     $(\iota, tbc) \notin \{\mathcal{A}$'s inputs to Certify oracle$\}$   ▷ $\mathcal{A}$ did not cheat
15:         then return 1                              ▷ $\mathcal{A}$ won
16:         else return 0                              ▷ $\mathcal{A}$ failed
```

Definition 6. *Let $\Theta = (\Theta_C, \Theta_S)$ be a pair of invertible functions from sequences of name-value pairs with unique names, to binary strings, and let $\mathcal{S}$ be a signature scheme. The $C^{\mathcal{S},\Theta}$ certificate scheme is defined as:*

$$C^{\mathcal{S},\Theta}.\text{KeyGen}(1^n) \equiv \mathcal{S}.\text{KeyGen}(1^n)$$

$$C^{\mathcal{S},\Theta}.\text{Certify}_{sk}(tbc) \equiv \Theta_S\left(\left\{\begin{array}{l} (\text{`}tbs\text{'}, \Theta_C(tbc)), \\ (\text{`}\sigma\text{'}, \mathcal{S}.\text{Sign}_{sk}(\Theta_C(tbc))) \end{array}\right\}\right)$$

$$C^{\mathcal{S},\Theta}.\text{Verify}_{pk}(\psi) \equiv \mathcal{S}.\text{Verify}_{pk}(\Theta_S^{-1}(\psi)[\text{`}tbs\text{'}], \Theta_S^{-1}(\psi)[\text{`}\sigma\text{'}])$$

$$C^{\mathcal{S},\Theta}.\text{Decode}_{pk}(\psi) \equiv \Theta_C^{-1}(\Theta_S^{-1}(\psi)[\text{`}tbs\text{'}])$$

Lemma 2. *Let $\mathcal{S}$ be an existentially-unforgeable signature scheme, and $\Theta = (\Theta_C, \Theta_S)$ be a pair of invertible functions (from sequences of name-value pairs with unique names to binary strings). Then $C^{\mathcal{S},\Theta}$, defined in Definition 6, is an existentially-unforgeable certificate scheme.*

Proof: Direct reduction to the security of $\mathcal{S}$. □

4.2 PKIX (with CRLs)

We now explain the construction of PKIX^C, which implements PKIX and the CRL revocation mechanism using an underlying certificate scheme C.

Entities and state. In PKIX^C, the set N of stateful entities is comprised of certificate authorities (CAs) which issue and revoke public key certificates. Some CAs are *root CAs* (trust anchors), which relying parties trust directly (by maintaining their self-signed certificates in *store*). Other CAs are *intermediate CAs*, which relying parties trust based on a certificate chain ending at a root CA. Each CA maintains a local state *st* which contains the following information:

- $st.\iota$: the identifier of the CA.
- $st.sk$: the CA's secret signing key.
- $st.pk$: the CA's public verification key.
- $st.certs$: the set of certificates issued by the CA.
- $st.CRL$: the list of all revoked certificates.
- $st.\Delta_r$: CRL's validity period.

Certificate fields and scheme. PKIX^C and CT use the generic certification scheme $C^{\mathcal{S},\Theta}$ of Definition 6, where the encoding schemes are defined in [41] and the certificate fields listed in Table 2.

Implementation. We now present the PKIX implementation, with CRLs, consisting of the operations in Definition 1.

PKIX^C.Init *(See [57]).* The algorithm initializes the state *st* of the CA using *params*: the CA's identity $st.\iota$, the CRL validity period $st.\Delta_r$, and defines the sets of certificates issued ($st.certs$) and revoked ($st.CRL$) by the CA as empty. Then, it generates the CA's keypair and self-certifies the public key. Finally, it outputs the initialized state and the self-certificate.

PKIX^C.Certify *(See [57]).* The algorithm does not certify: self-certificates, certificates with type *'SelfCert'* or 'CRL', nor certificates with an empty serial number. Hence, in any of these cases, the algorithm returns ⊥. Otherwise, the algorithm sets the *tbc.issuer* field the entity's identifier. If the inputted *tbc.type* field is ⊥, then the algorithm sets it to *'PubKey'*. Then, the algorithm adds the field names 'issuer', 'type', and 'PKIadded' to the *tbc.PKIadded* field, and then it signs *tbc* using C.Certify and the CA's secret signing key. Then, a copy of ψ is stored locally and the algorithm outputs ψ. The PKIX^C.Certify algorithm does not use or modify the inputted auxiliary information *aux* and returns the same *aux* in its output.[16]

PKIX^C.Revoke *(See [57]).* The algorithm verifies that the certificate to be revoked was issued by the CA and is not expired. If so, the algorithm adds the certificate to the list of revoked certificates.

PKIX^C.Valid *(See [57]).* The algorithm ensures two main things: (1) the inputted certificate ψ has a valid chain of certificates from ψ to one of the root CAs, and that (2) every certificate in the chain (including ψ) was not revoked. The algorithm starts by validating that the inputted certificate ψ has type *'PubKey'*. Then, the algorithm verifies that for some root certificate ψ_r, the inputted *aux* parameter contains a chain ξ which contains a sequence of certificates from ψ to a certificate $\xi[j]$ which has $\psi_r.subject$ as its issuer such that:

(1) For each certificate $\xi[k]$ from ψ to $\xi[j]$, the inputted time t is within the validity period of $\xi[k]$, and
(2) For each certificate $\xi[k]$ from ψ to $\xi[j]$, $\xi[k+1]$ has indeed issued $\xi[k]$, and
(3) For each certificate $\xi[k]$ from ψ to the certificate before $\xi[j]$, $\xi[k+1]$ is a public key certificate for a certificate authority.

To ensure that none of the certificates were revoked, the algorithm ensures that *aux* contains a valid CRL ψ_{CRL} for every certificate $\xi[k]$ such that:

(1) ψ_{CRL} is a CRL certificate, and
(2) ψ_{CRL} has the same issuer as $\xi[k]$, and
(3) The relevant CRL does not contain the serial of $\xi[k]$, and
(4) ψ_{CRL} has a valid chain that terminates in a root CA.

If any of the aforementioned checks fail, the algorithm outputs ⊥, otherwise, the certificate is considered valid, and therefore, the algorithm outputs ⊤.

[16] Although the PKIX^C.Certify operation returns the inputted auxiliary information *aux* unchanged, the Certify operations of other PKIs may use or output other auxiliary information.

PKIX$^{\mathcal{C}}$.VCerts *(See [57]).* First, the algorithm ensures the inputted certificate is valid. Then, it identifies a certificate of a root CA that is the trust anchor of ψ, along with the minimal subset of certificates from the inputted auxiliary information *aux*, and output their union.

PKIX$^{\mathcal{C}}$.GetCRL *(See [57]).* First, the algorithm removes all the certificates that expired from the current CRL. Then, it generates an updated CRL, i.e., defines data to be certified *tbc* with CRL type, sets the issuer and validity period, and sets the list of revoked certificates. Finally, the algorithm signs *tbc* using $\mathcal{C}$.Certify and the CA's secret signing key and outputs the CRL certificate.

5 SECURITY ANALYSIS

We analyze the security of the three schemes we present (PKIX$^{\mathcal{C}}$, CT$^{\mathcal{C},\mathcal{MT}}$, and CTwAudit$^{\mathcal{C},\mathcal{MT}}$) against the security requirements defined in §3. First, in §5.1, we describe the models assumed by these protocols, following [17]; we provide model predicates in [57]. Then, in §5.2, we prove that all three PKIs satisfy Existential Unforgeability, Accountability and Accountable Δ-Revocation. Lastly, in [57] we sum up the results of the analysis of Δ-Transparency for CT$^{\mathcal{C},\mathcal{MT}}$ and CTwAudit$^{\mathcal{C},\mathcal{MT}}$; we present the complete analysis in [57].

5.1 Model Predicates

We reuse the following adversary and clock drift model predicates defined in [17]:

- The π^{F} predicate ensures that all benign entities, i.e., not in T.F, follow the protocol correctly. T.F is the set of faulty entities outputted by the adversary. More precisely, the adversary can view the state, corrupt the state, or corrupt the output of the entities in T.F but not of any other entity.
- The $\pi^{\text{Drift}}_{\Delta_{clk}}$ predicate ensures that clock drifts from real time are bounded by Δ_{clk}. We presented it in Algorithm 1 (§3.1).

In addition, we reuse the standard communication model predicate $\pi^{\text{Com}}_{\Delta_{com}}$, also from [17], which ensures reliable communication between non-faulty parties, with delays bounded by Δ_{com}. The $\pi^{\text{Com}}_{\Delta_{com}}$ predicate in [17] ensures that if an operation outputs a ('send', m, j) triplet, then after at most Δ_{com}, entity j would receive m. Since the CT implementation in [57] has several request-response operations, we used a simpler notation for sending messages, and make the following small adjustment to the predicate:

(1) Whenever an entity ι includes in its output a triplet of the form (α-req,ι',x), where α-req is one of the request operations of CT, then, within Δ_{com}, there is an α-req event at ι' with input x.
(2) Whenever an entity ι' outputs a pair of the form (α-resp,y) in the output of an α-req operation, and this α-req was invoked by some entity ι, then, within Δ_{com}, there is an α-resp event at ι with input y.

The π^{Init} model predicate (See [57]) ensures correct initialization of all entities at the beginning of the execution.

The CT$^{\mathcal{C},\mathcal{MT}}$ PKI requires periodic operations; to support that, we define the $\pi^{\text{WakeAt}}_{\Delta_{clk},\Delta_w}$ predicate shown in [57] . This model supports waking up at a specified local time (within Δ_w) and allows passing values to the wake-up event[17]; it is likely to be useful for the analysis of other protocols. According to the predicate, if ('WakeAt', t, $context$) is outputted, t is the current local time or later, and execution did not end too early, then there is a Wakeup event at the same entity at local time[18] at least t and at most $t+\Delta_w$, with input $context$.

Finally, the π^{1LogCert} model predicate ensures that honest monitors do not receive multiple different log certificates with the same log identifier as input to the 'Monitor' operation (See [57]).

5.2 Analysis of Existential Unforgeability, Accountability and Accountable Revocation

We first show that the three PKIs satisfy Existential Unforgeability, Accountability and Accountable Δ_{Rev}-Revocation, where Δ_{Rev} is defined in Theorem 3.

Theorem 3. *Let $\mathcal{C}$ be an existentially-unforgeable certificate scheme*[19]*.* PKIX$^{\mathcal{C}}$*,* CT$^{\mathcal{C},\mathcal{MT}}$ *and* CTwAudit$^{\mathcal{C},\mathcal{MT}}$ *satisfy the following requirements:*

- *Existential Unforgeability, under the $\pi^{\text{Init}} \wedge \pi^{\mathsf{F}}$ model.*
- *Accountability, under the trivial (always true) model.*
- *Accountable Δ_{Rev}-Revocation, under the $\pi^{\text{Init}} \wedge \pi^{\mathsf{F}} \wedge \pi^{\text{Drift}}_{\Delta_{clk}}$ model, where $\Delta_{\text{Rev}} \equiv \Delta_{clk} + \Delta_r$ and Δ_r is the CRL validity period used in* PKIX$^{\mathcal{C}}$.

Proof. The proof is identical for the three PKIs; for convenience, we refer to PKIX$^{\mathcal{C}}$. We show that if each of the three requirements does not hold, then there is a PPT adversary $\mathcal{A}_{\mathcal{C}}$ that can forge $\mathcal{C}$-certificates with non-negligible probability, contradicting the assumption that $\mathcal{C}$ is an existentially-unforgeable certificate scheme.

Existential Unforgeability. Assume to the contrary that PKIX$^{\mathcal{C}}$ does not ensure Existential Unforgeability. By definition, this means that there exists a PPT adversary $\mathcal{A}_{\text{EUF}}$ that satisfies:

$$\Pr\left[\begin{array}{c}\pi_{\text{EUF}}\,(T) = \bot,\ \text{where}\\ T \leftarrow \mathbf{Exec}_{\mathcal{A}_{\text{EUF}},\text{PKIX}_{\mathcal{C}}}(params)\end{array}\right] \notin Negl(|params|) \tag{1}$$

From Equation (1), with non-negligible probability over the transcripts T of executions of PKIX$_{\mathcal{C}}$ with $\mathcal{A}_{\text{EUF}}$, we have $\pi_{\text{EUF}}\,(T) = \bot$. Following the implementation of π_{EUF} (Algorithm 2) and of PKIX$^{\mathcal{C}}$.VCerts and PKIX$^{\mathcal{C}}$.Valid, the adversary $\mathcal{A}_{\text{EUF}}$ managed to generate a certificate ψ_0 which is valid for time t, root store *store* and auxiliary information *aux*, and yet: (1) $\psi_0.issuer$ is a benign entity, (2) the PKIX$^{\mathcal{C}}$.Certify operation was not invoked at the benign issuer $\psi_0.issuer$ with the non-PKIadded fields of ψ_0 given as input, and (3) the validation of ψ_0 does not use a fraudulent certificate for the issuer $\psi_0.issuer$. In addition, from PKIX$^{\mathcal{C}}$.Valid we know that $\psi_0.type$ ='PubKey', since ψ_0 is valid.

We first show that $\mathcal{A}_{\text{EUF}}$ could not have generated ψ_0 by abusing PKIX$^{\mathcal{C}}$'s implementation and could not have used $\mathcal{C}$.Certify

[17]The $\pi^{\text{Wake-up}}_{\Delta_{clk}}$ predicate in [17] supports waking up after a delay (within Δ_{clk}), which is not optimal for CT$^{\mathcal{C},\mathcal{MT}}$ as it could cause gradual drift from the correct period time. Also, it does not support passing values from the request to the wake-up event.

[18]The reason why we use the *local time* is to keep the model realistic; if we required the wake-ups to be at *real time* at least t and at most $t+\Delta_w$, then entities may be able to use wake-ups to determine something about the real time, which should not be possible.

[19]From Lemma 2, this is equivalent to the unforgeability of the underlying signature scheme.

(through PKIX^{C}) at $\psi_0.issuer$ to certify $tbc \leftarrow C.\mathsf{Decode}(\psi_0)$. Then, we complete the proof by showing reduction to the security of C, i.e., the existence of $\mathcal{A}_{\mathsf{EUF}}$ means that C is not a secure certificate scheme.

According to the implementation of PKIX^{C}, the private key of an entity is generated using C.KeyGen in the PKIX^{C}.Init operation and stored locally in the state st. Following π^{Init}, no operation is called before the Init operation has been called at an entity, and the Init operation is always called with the correct inputs. Since the MoSS execution process ensures correctness of the states of benign entities, then following PKIX^{C}.Init, an entity calls C.KeyGen from PKIX^{C}.Init at most once, the first time that PKIX^{C}.Init is called at the entity. Non-faulty entities output a self-signed certificate only for a public key generated by the entity using C.KeyGen in PKIX^{C}.Init, and never output the corresponding private key. Thus, the adversary did not have direct access to the private key, and therefore, could not use it directly to generate ψ_0.

Moreover, the only time the private key is accessed is when used by the entity to certify information using C.Certify in PKIX^{C}.Init, PKIX^{C}.Certify and PKIX^{C}.GetCRL. However, out of these three functions, the only function where a certificate with type 'PubKey' is certified is PKIX^{C}.Certify, and as mentioned earlier, following π_{EUF}, we know that PKIX^{C}.Certify was not invoked at the benign issuer $\psi_0.issuer$ with the non-PKIadded fields of ψ_0 given as input. Following the implementation of PKIX^{C}.Certify, this implies that $\mathcal{A}_{\mathsf{EUF}}$ could not have used C.Certify (through PKIX^{C}) at $\psi_0.issuer$ to certify $tbc \leftarrow C.\mathsf{Decode}(\psi_0)$.

Consider an adversary $\mathcal{A}_C$ that receives as input a security parameter 1^{κ} and oracle access to the KeyGen$(\cdot)$ and Certify$(\cdot,\cdot)$ oracles defined in the *MECF* game (Algorithm 7). $\mathcal{A}_C$ runs $\mathcal{A}_{\mathsf{EUF}}$ internally against PKIX^{C}, with the following changes. First, whenever a benign entity ι calls C.KeyGen, $\mathcal{A}_C$ replaces this call with a call to the KeyGen oracle with input ι and then sets $(st.sk, st.pk) \leftarrow (\perp, pk)$, where pk is the public key returned by the KeyGen oracle. Second, whenever a benign entity ι calls C.Certify with input tbc, $\mathcal{A}_C$ replaces this call with a call to the Certify oracle with inputs ι and tbc. After the execution of $\mathcal{A}_{\mathsf{EUF}}$ and PKIX^{C} ends and $\mathcal{A}_C$ gets the execution transcript T, then $\mathcal{A}_C$ gets the certificate ψ_0 which is part of $T.out_{\mathcal{A}}$ and outputs $(\psi_0.issuer, \psi_0)$. If $\mathcal{A}_{\mathsf{EUF}}$ succeeds with non-negligible probability in π_{EUF} against PKIX^{C}, then $\mathcal{A}_C$ also succeeds with non-negligible probability in the *MECF* game, i.e., C does not ensure Multi-key Existential Unforgeability. Since if C does not ensure Multi-key Existential Unforgeability then it also does not ensure Existential Unforgeability (See [57]). This contradicts the assumption that C ensures Existential Unforgeability. □

Accountability. Assume to the contrary that PKIX^{C} does not ensure Accountability. By definition, there exists a PPT adversary $\mathcal{A}_{\mathsf{ACC}}$ which satisfies:

$$\Pr\left[\begin{array}{c}\pi_{\mathsf{ACC}}(T) = \perp,\ \text{where}\\ T \leftarrow \mathbf{Exec}_{\mathcal{A}_{\mathsf{ACC}},\mathsf{PKIX}^{C}}(params)\end{array}\right] \notin Negl(|params|) \qquad (2)$$

Namely, there is a non-negligible probability that $\pi_{\mathsf{ACC}}(T) = \perp$, where the probability is over the coin tosses in executions of PKIX_C with $\mathcal{A}_{\mathsf{ACC}}$. Following π_{ACC}, the adversary $\mathcal{A}_{\mathsf{ACC}}$ outputted a certificate ψ_0 which is valid for time t, root store *store* and auxiliary information *aux*, and yet, the output of $PKIX^{C}$.VCerts for ψ_0, t, *store*, and *aux does not ensure accountability.* By saying that the output does not ensure accountability we mean that either it includes no root certificate, or it includes some root certificate ψ_r such that none of the following three cases is true: (1) ψ_r is accountable for ψ_0, (2) ψ_r is a *bad* certificate (as defined in Section 3.3), or (3) ψ_r is accountable for a valid bad certificate among the certificates in the output of $\mathcal{P}$.VCerts.

The proof follows from the following three claims. *Claim 1* shows that if PKIX^{C}.Valid outputs true for some inputs ψ_0, t, *store*, and *aux*, then, for the same inputs, PKIX^{C}.VCerts outputs a root certificate and a valid certificate chain. *Claim 2* shows that if $\mathcal{A}_{\mathsf{ACC}}$ outputted ψ_0 which is valid for t, *store*, and *aux*, and yet the output of $PKIX^{C}$.VCerts for ψ_0, t, *store*, and *aux* does not ensure accountability, then $\mathcal{A}_{\mathsf{ACC}}$ has forged a certificate which is valid w.r.t. the public key of a benign entity. *Claim 3* completes the proof by a reduction to the security of C, i.e., showing that if an efficient $\mathcal{A}_{\mathsf{ACC}}$ can forge a certificate which is valid w.r.t. the public key of a benign entity, then C does not ensure Existential Unforgeability.

Claim 1: if PKIX^{C}.Valid *is true, then the output of* PKIX^{C}.VCerts *(for the same inputs) contains a root certificate and a corresponding valid certificate chain.*

Proof of claim 1: PKIX^{C}.Valid outputs $\top$ if and only if there exists a certificate ψ_r in *store* such that there exists a valid chain ξ in the output of $\textsc{ValidChain}(\psi_0, t, \psi_r, aux)$. PKIX^{C}.VCerts outputs $\{\psi_r'\} \cup \{Cert \in \xi'\}$ where $\psi_r' \in store$ and $\xi' \in \textsc{ValidChain}(\psi_0, t, \psi_r', aux)$;

Claim 2: if $\mathcal{A}_{\mathsf{ACC}}$ *outputted* ψ_0 *which is valid for t, store, and aux, and yet the output of PKIX*^{C}.VCerts *for* ψ_0*, t, store, and aux does not ensure accountability, then* $\mathcal{A}_{\mathsf{ACC}}$ *has forged a certificate which is valid w.r.t. the public key of a benign entity.*

Proof of claim 2: From claim 1, the output of $PKIX^{C}$.VCerts contains a root certificate, which we denote ψ_r, as well as a corresponding valid certificate chain from ψ_r to ψ_0.

If ψ_r is bad, then accountability holds, which contradicts the conditions of the claim. Therefore, ψ_r is not bad.

If for every certificate ψ_i in the chain, i.e., for every i s.t. $r > i \geq 0$), holds that $\psi_i.issuer$ has certified ψ_i, then ψ_r is accountable for ψ_0; in this case, accountability also holds, which contradicts the conditions of the claim.

Consider, therefore, the other case, where for some ψ_i on the chain ($r > i \geq 0$), $\psi_i.issuer$ did *not* certify ψ_i. WLOG, assume that i is the largest index for which this holds. It follows that either $\psi_{i+1} = \psi_r$ or ψ_r is accountable for ψ_{i+1}.

If $\psi_{i+1} \neq \psi_r$ and ψ_{i+1} is bad, then again accountability holds in contradiction to the conditions of the claim. Therefore, assume that ψ_{i+1} is not bad, i.e., $\psi_{i+1}.subject$ is benign and has self-certified $\psi_{i+1}.pk$. But ψ_i is valid w.r.t. $\psi_{i+1}.pk$, which is the public key of the benign entity $\psi_{i+1}.subject$; and $\psi_i.issuer$ did *not* certify ψ_i. Namely, $\mathcal{A}_{\mathsf{ACC}}$ has forged ψ_i, which is valid w.r.t. the public key of the benign entity $\psi_{i+1}.subject$, proving the claim.

Claim 3: if an efficient $\mathcal{A}_{\mathsf{ACC}}$ *can forge a certificate which is valid w.r.t. the public key of a benign entity, then* C *does not ensure Existential Unforgeability.*

Proof of claim 3: The proof follows similar reasoning to the proof of Existential Unforgeability for PKIX^{C}. Namely, we show that there exists an adversary $\mathcal{A}_C$ which runs $\mathcal{A}_{\mathsf{ACC}}$ internally against PKIX^{C} such that if $\mathcal{A}_{\mathsf{ACC}}$ succeeds with non-negligible probability in π_{ACC} against PKIX^{C}, then $\mathcal{A}_C$ also succeeds with non-negligible probability in the *MECF* game (Algorithm 7), i.e., C does not ensure Multi-key Existential Unforgeability.

Consider an adversary $\mathcal{A}_C$ that receives as input a security parameter 1^{κ} and oracle access to the KeyGen$(\cdot)$ and Certify$(\cdot,\cdot)$ oracles defined in the *MECF* game (Algorithm 7). $\mathcal{A}_C$ runs $\mathcal{A}_{\text{ACC}}$ internally against PKIX^{C}, with the following changes. First, whenever a benign entity ι calls C.KeyGen, $\mathcal{A}_C$ replaces this call with a call to the KeyGen oracle with input ι and then sets $(st.sk, st.pk) \leftarrow (\perp, pk)$, where pk is the public key returned by the KeyGen oracle. Second, whenever a benign entity ι calls C.Certify with input tbc, $\mathcal{A}_C$ replaces this call with a call to the Certify oracle with inputs ι and tbc. After the execution of $\mathcal{A}_{\text{ACC}}$ and PKIX^{C} ends and $\mathcal{A}_C$ gets the execution transcript T, it runs $PKIX^{C}$.VCerts on the values in $T.out_{\mathcal{A}}$ and then searches for a forged certificate in the output of $PKIX^{C}$.VCerts. If $\mathcal{A}_C$ finds such a forged certificate ψ, then it outputs $(\psi.issuer, \psi)$; otherwise it outputs $(\perp, \perp)$. If $\mathcal{A}_{\text{ACC}}$ succeeds with non-negligible probability in π_{ACC} against PKIX^{C}, then $\mathcal{A}_C$ also succeeds with non-negligible probability in the *MECF* game, i.e., C does not ensure Multi-key Existential Unforgeability. The claim follows, since if C does not ensure Multi-key Existential Unforgeability then it also does not ensure Existential Unforgeability (See [57]). This contradicts the assumption that C ensures Existential Unforgeability. □

Accountable Δ_{Rev}-Revocation. See [57] for this proof. □

5.3 Analysis of Transparency

We now provide an overview of the main results of our analysis of $\text{CT}^{C,\mathcal{MT}}$ and $\text{CTwAudit}^{C,\mathcal{MT}}$. The proofs and details are in [57].

5.3.1 $\text{CT}^{C,\mathcal{MT}}$ *ensures HL* Δ_{Tra}*-Transparency.*

Theorem 4. *Let C be an existentially-unforgeable certificate scheme and $\mathcal{MT}$ be a collision-resistant Merkle tree. Denote $\Delta_{\text{Tra}} \equiv 7 \cdot \Delta_{clk} + 2 \cdot \Delta_{MMD} + 2 \cdot \Delta_w + 5 \cdot \Delta_{com}$. Then, $\text{CT}^{C,\mathcal{MT}}$ satisfies the HL Δ_{Tra}-Transparency requirement under model predicate:*

$$\pi^{Init} \wedge \pi^{\text{F}} \wedge \pi^{1LogCert} \wedge \pi^{Drift}_{\Delta_{clk}} \wedge \pi^{Com}_{\Delta_{com}} \wedge \pi^{WakeAt}_{\Delta_{clk},\Delta_w} \quad (3)$$

Proof. See [57]. □

5.3.2 $\text{CT}^{C,\mathcal{MT}}$ *does not ensure Guaranteed* Δ*-Transparency or Audited* Δ*-Transparency.*

Theorem 5. *Let C be a secure certificate scheme and $\mathcal{MT}$ be a collision-resistant Merkle tree, and let Δ be any finite delay. Then,* $\text{CT}^{C,\mathcal{MT}}$ does not *satisfy the Guaranteed Δ-Transparency requirement or the Audited Δ-Transparency requirement under the model predicate of Equation 3.*

Proof. See [57]. □

5.3.3 $\text{CTwAudit}^{C,\mathcal{MT}}$ *ensures Audited* $\Delta^{AUD}_{\text{Tra}}$*-Transparency.*

Theorem 6. *Let C be an existentially-unforgeable certificate scheme and $\mathcal{MT}$ be a collision-resistant Merkle tree. Denote $\Delta^{AUD}_{\text{Tra}} \equiv 9 \cdot \Delta_{clk} + 2 \cdot \Delta_{MMD} + 2 \cdot \Delta_w + 5 \cdot \Delta_{com}$, where Δ_{MMD} is the maximal merge delay. Then, $\text{CTwAudit}^{C,\mathcal{MT}}$ satisfies the Audited $\Delta^{AUD}_{\text{Tra}}$-Transparency requirement under the model predicate of Equation 3.*

Proof. See [57]. □

6 CONCLUSIONS AND FUTURE WORK

PKI provides the foundation for the security of many deployed, critical distributed systems, in particular, the web and other applications using TLS, software signing, email security, and more. In spite of that, this work is the first to define security specifications for realistic PKI systems, supporting revocation, transitive trust (typically, certificate chains) and transparency, and considering realistic models allowing for corruptions, clock drift, delays and more.

A possible reason for this fundamental security infrastructure to remain without precise specifications and analysis may be that defining the requirements is tricky; they appear 'obvious' yet are hard to clearly define.

We applied our specifications to analyze the security of the two predominant PKI systems: PKIX and CT, as well as $\text{CTwAudit}^{C,\mathcal{MT}}$, a variant of CT implemented in the Chrome browser. Our analysis exposed several subtle issues with these systems, related to revocation and transparency. Due to these issues, the systems satisfy only relaxed variants of revocation and transparency.

This work makes only the first steps toward provable security of PKI schemes. Much work remains, including design and analysis of PKIs that will meet the stronger revocation and transparency requirements; formally defining other PKI schemes for the entire ecosystem (including browser specific implementations) and applying our specifications to such schemes; extending our specifications for non-certificate PKIs (e.g., [36]) or to additional properties (e.g., privacy, non-equivocation); and showing similar specifications and proofs under UC [6] or another framework allowing compositions of protocols. We also hope that similar specifications and analysis can be applied to other applied cryptographic protocols, e.g., blockchains.

ACKNOWLEDGEMENTS

We thank the reviewers for their insightful and constructive feedback. This work is partially supported by the National Science Foundation under Grants No. 2149765 and 2149766, by the United States-Israel Binational Science Foundation under Grant No. 2022701, by the Research Authority Fund of the College of Management Academic Studies, Rishon LeZion, Israel, and by the Comcast Corporation and by Synchrony Financial. The opinions expressed in this paper are those of the researchers and not of their university or funding sources.

REFERENCES

[1] Louise Axon and Michael Goldsmith. 2017. PB-PKI: A Privacy-aware Blockchain-based PKI. In *SECRYPT.*
[2] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. 2014. ARPKI: Attack Resilient Public-Key Infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 382–393.
[3] Alexandra Boldyreva, Marc Fischlin, Adriana Palacio, and Bogdan Warinschi. 2007. A Closer Look at PKI: Security and Efficiency. In *International Workshop on Public Key Cryptography.* Springer, 458–475.
[4] Johannes Braun. 2015. *Maintaining Security and Trust in Large Scale Public Key Infrastructures.* Ph.D. Dissertation. Technische Universität.
[5] Johannes Braun, Franziskus Kiefer, and Andreas Hülsing. 2013. Revocation & Non-Repudiation: When the first destroys the latter. In *European Public Key Infrastructure Workshop.* Springer, 31–46.
[6] Ran Canetti. 2020. Universally composable security. *Journal of the ACM (JACM),* 67, 5, 1–94.

[7] Ran Canetti, Daniel Shahaf, and Margarita Vald. 2016. Universally Composable Authentication and Key-exchange with Global PKI. In *Public-Key Cryptography–PKC 2016*. Springer, 265–296.

[8] BLUE BOOK CCITT. 1988. Recommendations X. 509 and ISO 9594-8. Information Processing Systems-OSI-The Directory Authentication Framework (Geneva: CCITT). (1988).

[9] Melissa Chase and Sarah Meiklejohn. 2016. Transparency Overlays and Applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 168–179.

[10] Council of European Union. 2021. Com/2021/281, Revision of the eIDAS Regulation - European Digital Identity (EUid). (2021).

[11] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. 2016. Secure Logging Schemes and Certificate Transparency. In *European Symposium on Research in Computer Security*. Springer, 140–158.

[12] John Dyer. 2015. China Accused of Doling Out Counterfeit Digital Certificates in 'Serious' Web Security Breach. VICE News. (Apr. 2015).

[13] Peter Eckersley. 2012. Sovereign Key Cryptography for Internet Domains. https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt ;hb=HEAD. (2012).

[14] Electronic Frontier Foundation (EFF). [n. d.] The EFF SSL Observatory. Retrieved May 30, 2019 from https://www.eff.org/observatory.

[15] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. 2014. A Decentralized Public Key Infrastructure with Identity Retention. *IACR Cryptology ePrint Archive*, 2014, 803.

[16] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. 2008. Universally Composable Security Analysis of TLS. In *International Conference on Provable Security*. Springer, 313–327.

[17] Amir Herzberg, Hemi Leibowitz, Ewa Syta, and Sara Wrótniak. 2021. MoSS: Modular Security Specifications framework. In *CRYPTO' 2021*. Full version at: https://eprint.iacr.org/2020/1040, 33–63.

[18] Amir Herzberg, Yosi Mass, Joris Mihaeli, Dalit Naor, and Yiftach Ravid. 2000. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. IEEE, 2–14.

[19] P. Hoffman, M. Blanchet, E. Lafon, Y. Galand, C. Elphick, and M. Moeller. 2002. International organization for standardization, information technology - asn.1 - basic encoding rules (ber). In *ITU-T Recommendation X.690 | ISO/IEC 8825-1:2002*. Also covers DER, 1–105.

[20] Jacob Hoffman-Andrews. 2023. Article 45 Will Roll Back Web Security by 12 Years. ACLU. https://www.eff.org/deeplinks/2023/11/article-45-will-roll-back-web-security-12-years. (Nov. 2023).

[21] Joel Hruska. 2015. Apple, Microsoft buck trend, refuse to block unauthorized Chinese root certificates. ExtremeTech. (Apr. 2015).

[22] Jingwei Huang and David M Nicol. 2017. An anatomy of trust in public key infrastructure. *International Journal of Critical Infrastructures*, 13, 2-3, 238–258.

[23] International Telecommunication Union. 1997. ITU-T X.509 recommendation version 3: information technology - open systems interconnection - the directory: authentication framework. (June 1997). https://www.itu.int/rec/T-REC-X.509-199708-S.

[24] Google LLC Joe DeBlasio. [n. d.] Opt-out SCT Auditing in Chrome. Other. https://docs.google.com/document/d/16G-Q7iN3kB46GSW5b-sfH5MO3nKSYyEb77YsM7TMZGE/. ().

[25] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. 2013. Accountable Key Infrastructure (AKI): A Proposal for a Public-Key Validation Infrastructure. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 679–690.

[26] Loren M Kohnfelder. 1978. *Towards a practical public-key cryptosystem*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[27] Murat Yasin Kubilay, Mehmet Sabir Kiraz, and Haci Ali Mantar. 2018. CertLedger: A new PKI model with Certificate Transparency based on blockchain. *arXiv preprint arXiv:1806.03914*.

[28] Ben Laurie. 2014. Certificate transparency. *Communications of the ACM*, 57, 10, 40–46.

[29] Ben Laurie and Emilia Kasper. 2012. Revocation Transparency. *Google Research, September*.

[30] Dimitrios Lekkas. 2003. Establishing and managing trust within the Public Key Infrastructure. *Computer Communications*, 26, 16, 1815–1825.

[31] Bingyu Li, Jingqiang Lin, Fengjun Li, Qiongxiao Wang, Qi Li, Jiwu Jing, and Congli Wang. 2019. Certificate transparency in the wild: exploring the reliability of monitors. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2505–2520.

[32] John Marchesini and Sean Smith. 2005. Modeling Public Key Infrastructure in the Real World. In *European Public Key Infrastructure Workshop*. Springer, 118–134.

[33] Stephanos Matsumoto and Raphael M Reischuk. 2017. IKP: Turning a PKI Around with Decentralized Automated Incentives. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 410–426.

[34] Ueli Maurer. 1996. Modelling a Public-Key Infrastructure. In *European Symposium on Research in Computer Security*. Springer, 325–350.

[35] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. 2015. CONIKS: Bringing Key Transparency to End Users. In *USENIX Security Symposium*, 383–398.

[36] [n. d.] Namecoin. (). https://www.namecoin.org/.

[37] Scientists Organisations and Researchers as signed. 2023. Joint statement of scientists and NGOs on the EU's proposed eIDAS reform. Other. https://nce.mpi-sp.org/index.php/s/cG88cptFdaDNyRr. (Nov. 2023).

[38] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. 1999. SPKI Certificate Theory. RFC 2693 (Experimental). RFC. Fremont, CA, USA: RFC Editor, (Sept. 1999). DOI: 10.17487/RFC2693.

[39] R. Housley, W. Polk, W. Ford, and D. Solo. 2002. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280 (Proposed Standard). RFC. Obsoleted by RFC 5280, updated by RFCs 4325, 4630. Fremont, CA, USA: RFC Editor, (Apr. 2002). DOI: 10.17487/RFC3280.

[40] J. Jonsson and B. Kaliski. 2003. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational). RFC. Obsoleted by RFC 8017. Fremont, CA, USA: RFC Editor, (Feb. 2003). DOI: 10.17487/RFC3447.

[41] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard). RFC. Updated by RFCs 6818, 8398, 8399. Fremont, CA, USA: RFC Editor, (May 2008). DOI: 10.17487/RFC5280.

[42] M. Lepinski and S. Kent. 2012. An Infrastructure to Support Secure Internet Routing. RFC 6480 (Informational). RFC. Fremont, CA, USA: RFC Editor, (Feb. 2012). DOI: 10.17487/RFC6480.

[43] B. Laurie, A. Langley, and E. Kasper. 2013. Certificate Transparency. RFC 6962 (Experimental). RFC. Obsoleted by RFC 9162. Fremont, CA, USA: RFC Editor, (June 2013). DOI: 10.17487/RFC6962.

[44] S. Josefsson and S. Leonard. 2015. Textual Encodings of PKIX, PKCS, and CMS Structures. RFC 7468 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, (Apr. 2015). DOI: 10.17487/RFC7468.

[45] E. Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, (Aug. 2018). DOI: 10.17487/RFC8446.

[46] B. Laurie, E. Messeri, and R. Stradling. 2021. Certificate Transparency Version 2.0. RFC 9162 (Experimental). RFC. Fremont, CA, USA: RFC Editor, (Dec. 2021). DOI: 10.17487/RFC9162.

[47] Steven B Roosa and Stephen Schultze. 2010. The "Certificate Authority" Trust Model for SSL: A Defective Foundation for Encrypted Web Traffic and a Legal Quagmire. *Intellectual property & technology law journal*, 22, 11, 3.

[48] Mark Dermot Ryan. 2014. Enhanced certificate transparency and end-to-end encrypted mail. In *NDSS*.

[49] Wazan Ahmad Samer, Laborde Romain, Barrere Francois, and Benzekri AbdelMalek. 2011. A formal model of trust for calculating the quality of X. 509 certificate. *Security and Communication Networks*, 4, 6, 651–665.

[50] Nicolas Serrano, Hilda Hadan, and Jean L. Camp. 2019. A complete study of P.K.I. (PKI's Known Incidents). Available at SSRN, https://ssrn.com/abstract=3425554. (July 2019).

[51] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, and Bryan Ford. 2015. Certificate Cothority: Towards Trustworthy Collective CAs. *Hot Topics in Privacy Enhancing Technologies (HotPETs)*, 7.

[52] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. 2016. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *Security and Privacy (SP), 2016 IEEE Symposium on*. Ieee, 526–545.

[53] Pawel Szalachowski, Stephanos Matsumoto, and Adrian Perrig. 2014. PoliCert: Secure and Flexible TLS Certificate Management. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 406–417.

[54] Alin Tomescu and Srinivas Devadas. 2017. Catena: Efficient Non-equivocation via Bitcoin. In *2017 38th IEEE Symposium on Security and Privacy (SP)*. IEEE, 393–409.

[55] Dan Wendlandt, David G Andersen, and Adrian Perrig. 2008. Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. In *USENIX Annual Technical Conference*. Vol. 8, 321–334.

[56] Wikipedia contributors. 2021. Diginotar — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=DigiNotar&oldid=1036090956. [Online; accessed 7-August-2021]. (2021).

[57] Sara Wrótniak, Hemi Leibowitz, Ewa Syta, and Amir Herzberg. 2019. Provable security for PKI schemes. Cryptology ePrint Archive, Paper 2019/807. https://eprint.iacr.org/2019/807. (2019). https://eprint.iacr.org/2019/807.

[58] Jiangshan Yu, Vincent Cheval, and Mark Ryan. 2016. DTKI: A New Formalized PKI with Verifiable Trusted Parties. *The Computer Journal*, 59, 11, 1695–1713.