Provably Model-Secure PKI Schemes

It is possible to build a cabin with no foundations, but not a lasting building.

- Eng. Isidor Goldreich [30].

Hemi Leibowitz[1], Amir Herzberg[2], and Ewa Syta[3]

[1] Dept. of Computer Science, Bar-Ilan University, Ramat Gan, Israel
[2] Dept. of Computer Science and Engineering, University of Connecticut, Storrs, CT
[3] Dept. of Computer Science, Trinity College, Hartford, CT**

Abstract. Public Key Infrastructure (PKI) schemes have significantly evolved since X.509, with more complex goals, e.g., *transparency*, to ensure security against corrupt issuers. However, their security properties have not been rigorously defined or established. This is concerning as PKIs are the basis for security of many critical systems, and security concerns exist, even for well known and deployed PKI schemes, e.g., Certificate Transparency (CT) [39].

We present the first rigorous security specifications for PKI schemes, with properties such as *transparency, revocation transparency* and *non-equivocation*. We show that these security definitions are satisfiable, by presenting PoC-PKI, a 'proof-of-concept' PKI scheme, and proving that the scheme meets all of the PKI security properties we defined. We also analyze CT, and show that the design specified in RFC6962 achieves some of the properties.

Lastly, we present the *Model-Secure framework* that offers a novel approach, where security requirements are defined with respect to a specific *model predicate* $\mathcal{M}$. This allows analysis of PKI schemes and other protocols under well-defined adversary, communication and synchronization models, in a modular and flexible way, e.g., both simplified and realistic models. The framework facilitates reuse of definitions, and, indeed, several of the model predicates and security requirements we define, seem 'generic' and reusable in analysis of other practical protocols.

1 Introduction

Public Key Infrastructure (PKI) provides an essential foundation for applications which rely on public key cryptography, and it is crucial to achieve security in open networks and systems. Since its introduction in 1988, the deployment of PKI has been dominated by the X.509 standard [15], likely due to its integration with the TLS/SSL protocol [53], the most widespread protocol used to secure connections

** The work was partially completed during a visiting position at the Dept. of Computer Science and Engineering, University of Connecticut, Storrs, CT

between servers and clients, most commonly web browsers. The resulting 'web-PKI' is necessary to provide confidentiality, integrity and authenticity of web services, and as such, is critical for the secure use of the web.

Unfortunately, the web-PKI deployment has inherent weaknesses. In particular, any CA is trusted to issue certificates for any domain [22], resulting in the weakest-link security model and making individual CAs prime targets for attacks. Over the years, we have seen many failures of this trusted-CA approach. For example, hackers stole the master keys of CAs [17, 52] and issued fake certificates for major websites. Furthermore, some CAs abused their powers by improperly delegating their certificate-issuing authority or even intentionally issuing unauthorized certificates [23]. Such PKI failures allow attackers to issue fake certificates, launch website spoofing and man-in-the-middle attacks, possibly leading to identity theft, surveillance, compromises of personal and confidential information, and other serious security breaches.

X.509 certificates are signed by the issuing CA, which ensures *accountability*: a CA cannot deny having issued a certificate - or, more precisely, that its private key was used to sign the certificate. For many years, this was considered a sufficient deterrent; however, the many PKI failures brought the realization that *accountability is not sufficient*. Accountability is only effective if and when the fake certificate is found - which may not occur, especially if abused 'stealthily', and only if the misbehaving authority can be effectively punished.

These failures motivated efforts to develop and adopt *improved-security PKI schemes*, i.e., PKI schemes that ensure security against corrupt CAs. During the recent years, there have been extensive efforts toward this goal by researchers, developers and the IETF. These efforts focus on security properties such as *transparency*, *non-equivocation* and more. Proposals and designs include *Certificate Transparency (CT) [38,39], Enhanced-CT [55], Sovereign Key [24], CONIKS [47], AKI [36], PoliCert [61], ARPKI [4], DTKI [64], CoSi [59, 60], IKP [44], Cert-Coin [27], PB-PKI [3], Catena [62]*, *CertLedger [37]*, among others.

The goals of these designs are beyond those of X.509, and are significantly more complex than the X.509 goals. However, so far, these goals have not been rigorously defined and proven. Only few works present any analysis: [16, 21] analyze (only) the logging mechanism of CT, and ARPKI [4, 64] use automated symbolic analysis for system-specific properties. However, no work defines security goals (or proves such properties hold). In fact, even for the simple, 'classical' X.509 PKI, there is *no definition of security requirements* (and no proof). Arguably, this may not be as critical, since for X.509, both definitions and proof are quite straightforward (see within). However, this lack of definition (and proof) implies that works analyzing security of PKI-based protocols, e.g., IPsec/IKE [12, 19] and TLS [34, 50], mostly completely ignore the PKI and simply assume the use of correct public keys. Few works [5, 14, 28] study security of cryptographic protocols based on a *simplification* of X.509; however, the simplification *ignores revocation* and other advanced aspects of X.509, not to mention the properties of post-X.509 PKI schemes, such as transparency.

This is alarming, as most practical applications of cryptography involve certificates, and their security depends on the security of the PKI. The extensive efforts to prove security of cryptographic protocols may be moot when these protocols are deployed over a PKI scheme which was not proven secure. The concerns are even greater, considering that attacks against PKI are not only a theoretical threat, but are a major concern in practice.

Indeed, defining and proving security for PKI schemes is a non-trivial challenge, especially for post-X.509 schemes, with more advanced and complex goals. PKI proposals vary greatly - even terms of the types of parties involved, and in the communication and attack models. Further, any existing definitions and requirements are often informally defined and tailored to a specific design. The lack of *proper* definitions and proofs makes it challenging to build (provably) secure systems, which depend on PKI schemes, and to improve, compare and select PKI schemes. Evaluation of PKI schemes with new properties is especially challenging; for example, there are several schemes designed to achieve different *privacy* goals, but these cannot be properly compared - and, of course, are not proven secure. It is impossible to design and analyze schemes modularly, by provable reductions to simpler, already-analyzed schemes.

First contribution: definitions and implementations of provably-secure PKI schemes. In this work, we present well-defined correctness and safety requirements for PKI schemes, with reduction-based proofs of security. The definitions supports a wide range of PKI schemes, from X.509 to advanced, improved-security PKI schemes, independently of their specific designs.

We focus on the classical security challenge of dealing with misbehaving parties, e.g., corrupt CAs, by *detecting* misbehavior and/or *preventing* damage due to misbehavior. Note that we do not address *trust-management* issues, such as the *decision to trust* a particular CA, typically based on cross-certification by already-trusted CAs ('basic constraints' in X.509), or restricting a CA to particular name-space ('naming constraints' in X.509). Other works address other important aspects of PKI schemes, mainly the *trust decision* - essentially, which CA should be trusted for a given certificate. A model of trust for PKI systems was proposed by Mauer [45], subsequently extended by [9, 43], and others [8, 31, 33, 41, 56, 65].

To define the security requirements, we reviewed and analyzed existing PKI schemes and the properties they claim to provide. We present game-based definition for the basic security requirements for PKI schemes designed for possibly faulty CAs. These include the following safety requirements: *accountability*, Δ-*transparency*, *non-equivocation* (detection and prevention), *revocation accountability*, *non-revocation accountability* and Δ-*revocation transparency*. We map these safety requirements to existing PKIs in Table 1.

To make sure that the PKI properties we identified are comprehensive and feasible, we analyzed two (very different) PKI schemes. The first is a 'Proof of Concept' PKI (PoC-PKI) scheme we present, which we prove to ensure all of the security requirements we defined. The second is CT_{comp}, which is essen-

tially *Certificate Transparency (CT)*, complemented with essential details left unspecified in [39].

Second contribution: the Model-Secure framework. Applied cryptographic protocols, and specifically PKI schemes, can be designed and analyzed under different assumptions as well as adversary, communication and synchronization models. For PKI schemes, in particular, these aspects have crucial implications on their security. It is well established that cryptographic protocols are specified and analyzed under specific adversary models; communication and synchronization models are not always relevant, and when they are, highly simplified models are often applied, e.g., perfect synchronization ('rounds model').

This creates a challenge for the application of provable security to applied cryptographic protocols, which are usually designed to operate under realistic communication and synchronization models. We have considered the use of existing rigorous frameworks for analysis of distributed algorithms, e.g., the I/O automata [35] or reactive systems [51]. However, these frameworks are overly complex, even when focusing on relatively simple algorithms and requirements; applying them to complex tasks such as PKI schemes and to cryptographic protocols, appears prohibitively challenging.

Instead, we present the *model-secure framework*, which cleanly separates the specification and analysis of security into three components: the *execution process*, *model* and *requirements*. The first component is a well-defined, and relatively straightforward, *adversary-driven execution process* (Algorithm 1). This execution process defines precisely the process of *executing a protocol $\mathcal{P}$ under adversary $\mathcal{A}$* giving the adversary an extensive control over the operation of the environment, including communication, local-clock values, inputs from the application, and faults. We then separately define a model $\mathcal{M}$ that enforces well-defined restrictions on the adversary's actions. (The execution process also imposes some basic limitations, e.g., events are ordered, rather than potentially only partially-ordered, as allowed in [35, 51].)

The second component of the framework is the definition of *model predicates*. A model predicate $\mathcal{M}$ maps the execution of $\mathcal{P}$ under $\mathcal{A}$ into 'valid' ($\top$) or 'invalid' ($\bot$), effectively enforcing one or multiple restrictions on the adversarial control of the execution, including initialization assumptions, limitations on number and type of faults, maximal delay and/or maximal clock drift. This execution process allows formal yet flexible and generic security definitions, which apply under a variety of adversary, communication and synchronization models. Unlike our approach, current definitions of security do not separate between the *security requirements* and the *adversarial, communication and synchronization models* assumed by a given protocol. This lack of separation, makes it hard to apply a fixed set of security requirements, to protocols designed and analyzed under different models.

This execution process is a significant deviation from the 'classical' approach of defining security using games (experiments), with a given protocol and adversary; in this 'classical' definitions, the model is an integral part of the game/experiment, while in our execution process, the model $\mathcal{M}$ is also specified as a parameter.

Like the adversary $\mathcal{A}$ and the protocol $\mathcal{P}$, the model is also well-defined as a PPT algorithm $\mathcal{M}$. We found this approach necessary for this work, to allow our definitions to apply to different PKI scenarios and applications with (very likely) different assumptions. The use of this non-standard execution process introduces some additional burden in this paper - to its readers as well as authors. We apologize, and assure the readers that we strived for simplicity and a 'clean' methodology, justifying this extra effort by providing flexible, general PKI definitions. Furthermore, we believe that the approach is likely to be beneficial for other cryptographic protocols and problems, allowing definitions to apply across different adversary, communication and synchronization models, and making it easier to compare models between different works; indeed, we expect the separation between the model aspects and the problem-specific aspects of security definitions, to allow cleaner definitions, as well as to allow the reuse of well-defined models $\mathcal{M}$. We therefore consider the execution process and methodology to be an additional important contribution of this work.

Organization. §2 reviews the PKI landscape with respect to the requirements we identify in our framework and summarizes the related work. §3 presents the execution model of the framework and §4 presents the PKI framework and its security requirements. §5 discusses how the framework can be applied in practice. We conclude and discuss future work in §6. Additional content is available in the appendices.

2 Security Properties of PKI Schemes

The first step in developing the PKI framework was the identification of the security properties of PKI schemes, where we focus on schemes designed for security against corrupt authorities (typically, corrupt CA). In this section, we first discuss, informally, these security properties which we then formally define (see §4.4 for the game-based definitions).

We put an extensive effort into a thorough comparison of the existing schemes with respect to the properties we identify. We present the result in Table 1 and discuss them in §2.3. This work significantly benefited from this comparison.

2.1 X.509 PKI schemes and related requirements

The basic goal of a PKI scheme is to ensure *authenticity* of information in *public key certificates*. Certificates are issued and endorsed by *Certificate Authorities (CAs)*. An *honest* CA issues a certificate only after it verifies that the entity requesting the certificate is eligible to receive it. A typical certificate contains an *identifier* and some *public information*, typically including a public key. A certificate also typically includes a signature generated by the issuing CA, over the certificate's information; the signature serves as the CA's endorsement of the mapping between the identifier and the public information in the certificate. The signature establishes the basic goal of PKI schemes: *accountability*.

*Accountability (*ACC*)* is the ability to *identify the CA that issued a given certificate*. Accountability provides a reactive defense against a corrupt CA; such CA can be ignored or otherwise punished. In most PKI schemes, including X.509, accountability is achieved by having the CA digitally sign certificates, i.e., a CA is accountable for any certificate signed using the CA's private key. CA accountability, in this sense, includes unauthorized use of the CA's private key, e.g., due to exposure or penetration, as well as intentionally issuing 'fake' certificate, where the public information does not correctly match the identifier. Note that we use the term accountability as a technical, well-defined property, which does not necessarily have any specific legal or financial implications.

*Revocation accountability (*ReACC*).* A certificate can be considered valid only after its *issue date* and until its *expiration date*, both of which specified in the certificate. The issuing CA, however, can invalidate a certificate before its expiration date by *revoking* it. A user can request to have their certificate revoked for a variety of reasons, including a loss or compromise of the private key corresponding to the public key endorsed in the certificate.

The two main revocation mechanisms in practice are the *certification revocation lists (CRLs)* [18] and *online certificate status protocol (OCSP)* [57]. Their main security property is *revocation accountability* (ReACC).

Revocation accountability (ReACC) ensures accountability of the revoked certificates. Namely, each revoked certificate can be traced back to the revoking CA. This helps to ensures that a client will not have their certificate revoked without a legitimate reason (e.g., their request), unless the CA is malicious, or an attacker corrupts or tricks the CA - in which case, this can be exposed. Revocation accountability is similar to the accountability property, which focuses on issuing certificates. In X.509 and most PKI schemes, only the issuer can revoke a certificate, but this is not necessarily always the case (in other PKI schemes).

Some PKI revocation mechanisms, most significantly OCSP [57], also support a 'positive' attestation for certificates, i.e., indicating the certificate was *not* revoked (up to given time). This 'non-revoked' certificate is signed by a CA - typically, the issuer. Such attestations allow a relying party to justify reliance on the certificate, more precisely, on the mapping of public information/key to the identifier, e.g., for accepting a signed document as evidence of it being approved by the subject of the certificate.

The non-revocation accountability (NReACC) property ensures that a honest CA would never sign both revocation and non-revocation for the same certificate, with overlapping periods; in fact, if the same CA signs such a pair of conflicting certificates, this provides a Proof of Misbehavior. In X.509, and even most post-X.509 PKI schemes, only the issuer can revoke certificates, making this property easy to define and achieve. However, considering the history of CA failures, it may be useful to allow revocation of certificates by other authorities, making it harder to define and achieve the non-revocation accountability property. We discuss post-X.509 schemes and requirements in the next subsection.

2.2 Post-X.509 PKI schemes and requirements

In §2.1, we discussed accountability, revocation accountability and non-revocation accountability, which correspond to the basic PKI properties, provided already by X.509. We now discuss additional security goals, pursued by more recent PKI schemes, designed to improve security against corrupt CAs. These include Δ-*transparency* (ΔTRA), Δ-*revocation transparency* (ΔReTRA), Δ-*equivocation detection* (ΔEQ-D) and *equivocation prevention* (EQ-P).

Δ-Transparency (ΔTRA). Accountability, as described above, mainly serves as a *deterrent* against misbehavior, i.e., only offers retroactive security by punishing a CA 'caught' misbehaving, e.g., issuing a fraudulent certificate. For many years this reactive measure was viewed as a sufficient defense, under the assumption that CAs were highly respectable and trustworthy entities who would not risk, intentionally or otherwise, being implicated in issuing fraudulent certificates. However, repeated cases of fake-certificates, by compromised or dishonest CAs, have proven this assumption to be overly-optimistic. It turned out that punishing CAs is non-trivial: beyond negative publicity, any punishment was arbitrary, short-lived and overall ineffective [2, 32, 54].

Furthermore, 'punishment' could only be applied *after* the damage was committed and *discovered* - if it is discovered at all. An attacker or corrupt CA could reduce the risk of discovery, by minimizing the exposure of the fraudulent certificate. Except for efforts such as the Perspectives Project [63], or the EFF SSL Observatory [25] that aim to gather and inspect *all* SSL certificates used in practice, the burden of detecting and responding to fraudulent certificates is mostly on the clients that receive them; browsers typically cannot detect fraudulent certificates, much less to report them to a (non-existing) 'enforcement agency'.

This significant issue has motivated more recent PKI designs, e.g. CT, where certificates are *transparently published*, to allow third parties (e.g., trusted 'monitors') to inspect and detect any fraudulent certificates. This design makes it possible to quickly detect misbehavior, such as issuing of a fraudulent certificate. Ideally, fraudulent certificates could likely be detected *before* they can be abused, or at least, before they can cause much harm.

Unfortunately, there is still no guarantee that such detection would in fact occur *before* certificate misuse occurs. In fact, even where detection is guaranteed to occur, this can only be guaranteed some time after issuing of the fake certificate - although this aspect is often overlooked. We denote this time by Δ, and hence we refer to this property as Δ-*transparency* (or ΔTRA); in a specific PKI scheme, the value of Δ would be a function of model-specific parameters (such as network delay). Transparency prevents a CA from 'silently' generating fraudulent yet validly-formed certificates, and exposing them only to selected victims during an attack.

Transparency requires a certificate to be authenticated (signed) by a party which takes responsibility for making the certificate known to all monitoring-entities, within the specified time frame Δ. By demanding transparency, a PKI system facilitates detection of fraudulent certificates, even when issued by a corrupt or compromised CA. Often, a certificate is considered fraudulent since it

uses a *misleading* identifier, such as a domain name which is identical or similar to that of a victim domain, e.g., g00gle.com, or with an identifier which users may expect to belong to a known domain, e.g., googleaccounts.com. Such misleading identifiers are often abused, e.g., for *phishing* attacks. A PKI which supports transparency, allows a domain to vigilantly watch for any certificate issued with identifiers which are identical, similar or otherwise misleading to be associated with its own domain names.

Equivocation: detection and prevention (ΔEQ-D *and* EQ-P*).* Fraudulent certificates which use the *same* identifier as the victim, may be abused for phishing, and for other attacks, e.g., stealing web cookies. PKI schemes may detect equivocation (ΔEQ-D) or even prevent it (EQ-P).

A PKI which prevents equivocation, will prevent a corrupt CA from issuing a fake certificate for an already-certified identifier, e.g., domain name. This could *prevent*, rather than merely *detect*, man-in-the-middle and other attacks impersonating existing secure domains [10].

Note that transparency implies ΔEQ-D, but not EQ-P. We still define equivocation detection as a separate property, since it does *not imply* transparency, i.e., ΔEQ-D is not equivalent to transparency. In fact, some PKI schemes, notably CONIKS [47], offer equivocation detection but *not transparency* - indeed, transparency would conflict with some of CONIKS privacy goals. Of course, providing non-equivocation but not transparency, may still allow issuing of misleading (but not identical) identifiers, e.g., misleading domain names which may be abused for phishing attacks.

Revocation transparency (ΔReTRA*).* Revocation accountability does not ensure that revocation would be performed *correctly*. Consider a scenario where a client asks to have her certificate revoked, but a corrupt CA does not properly revoke the certificate, and as a result, some (or all) relying parties are kept unaware of the revocation, and still consider the certificate as valid. Obviously, such behavior may endanger the client in many scenarios, e.g., when the corresponding private key was obtained by an attacker. Revocation transparency ensures that if a CA revoked a certificate, then *all* relevant authorities should be aware of the revocation , within some bounded time, preventing such undesirable scenarios.

Privacy. Some of the recent PKI schemes offer different privacy properties. However, the properties are non-trivial and also differ significantly. Therefore, we left to future work, the important and challenging task of extending the PKI framework to privacy properties. Note that, as discussed above, some privacy properties may conflict with transparency.

2.3 Properties of different PKI Schemes

The goal of the PKI framework is to allow analysis and provable-security, for existing and future PKI schemes. We designed the framework in a way that *embraces*, *complements* and *reflects* current PKI designs. To this end, we have methodically examined the existing PKI schemes by identifying and analyzing

System [reference]	Safety requirements						Additional req.	
	ACC	ΔTRA	ΔEQ-D	EQ-P	ReACC/ NReACC	ΔReTRA	Privacy[1]	Global name-space
X.509 and PKIX, with CRL or OCSP [2]	◐	n/s	n/s	n/s	◐	n/s	n/s	✓
Catena [62]	⊙[7]	⊙	⊙	⊙	⊙[7]	⊙	n/s	✓
CertCoin [27]	n/s	⊙	⊙	⊙	n/s	⊙	n/s	✓
PB-PKI [3]	n/s	⊙	⊙	⊙	n/s	⊙	⊙	✓
CoSi [60]	◐	⊙	⊙	⊙	n/s	n/s	n/s	✓
Enhanced-CT [55] DTKI [64] [3]	◐	⊙	⊙	n/s	◐	⊙	n/s	✓
AKI [36]	◐	⊙	⊙	n/s	◐	⊙	⊙	✓
CONIKS [47]	◐	n/s	⊙	n/s	◐	⊙	⊙	✗
ARPKI [4] [4]	◐	⊙	⊙	⊙	◐	⊙	n/s	✓
CertLedger [37]	◐	⊙	⊙	⊙	⊙	⊙	n/s	✓
Certificate Transparency (CT) [39]	◐	⊙[5]	⊙[5]	n/s	◐	n/s[6]		✓
CT_{comp} (CT completed, this work)	●	●	n/s	n/s	●	n/s	n/s	✓
PoC-PKI (this work)	●	●	●	●	●	●	n/s	✓

Table 1: Comparison of PKI schemes with respect to PKI framework. Symbols: ● - reduction-based proofs, ◐ - intuitively true, ⊙ - security arguments (a proof may require assumptions), n/s - not supported.
[1]Different privacy definitions, goals. [2]OCSP ensures NReACC. [3]DTKI has symbolic proofs of some aspects. [4]ARPKI has symbolic proofs of some aspects. [5]Proofs of logging properties in [16, 21]. [6]CT is extended to include revocation transparency in [38].

their properties. We present the results of our analysis in Table 1, and summarize them below. The table includes twelve existing PKI systems, and, in addition, PoC-PKI, a "proof-of-concept" PKI we defined, and CT_{comp}, a minor extension of the CT specifications, which appears essential to ensure CT's security properties. We compared all schemes with respect to the requirements formally presented in §4; we also mention two additional properties, privacy, discussed informally in §2.2, and global name-spaces.

Notations in Table 1. We use the n/s (not supported) symbol to indicate when a scheme does not seem to support a requirement. Otherwise, we use one of the three following symbols, ●, ⊙, or ◐, to indicate the support of a requirement by the scheme. The ● symbol indicates that a system comes with *rigorous*, reduction-based proof of the requirement. We indicate with an appropriate comment when a scheme is supported by automated symbolic proof for a given property; note that such proofs are often of property specific to that scheme, not properties defined for arbitrary PKI schemes. The ◐ symbol indicates that although no formal proofs were provided, it seems *intuitively true* that the system achieves a requirement; e.g., accountability in X.509 follows from the use of signature scheme to sign the certificate. The ⊙ symbol depicts the property is justified using an (informal) security argument; note that this may imply that additional assumptions or details may be needed to ensure security.

Following our discussion of the 'basic' PKI security properties in §2.1, we observe that most systems aim to achieve accountability, with the exception of CertCoin and PB-PKI. Both CertCoin and PB-PKI build on top of Namecoin [1], which is a decentralized namespace system rather than a centralized, CA-oriented system, where the CAs grant identifiers to clients. Instead, due to the fully decentralized nature, anyone can claim an identifier so long it is available; consequently, there is no accountability for assigning identifiers. Notice also that Catena is a witnessing (logging) scheme that allows to witness public-key directories using the Bitcoin blockchain. As a result, accountability of issuing certificates is handled by the directories themselves, which require unusual additional assumptions (which can be modelled using the framework).

Interestingly, many systems directly focus on more advanced properties, such as transparency and non-equivocation, and treat more 'basic' properties, such as accountability and revocation, as intrinsic to PKI, often without even stating them. This phenomenon is especially apparent in case of revocation; many systems (e.g., CertCoin, Catena, PB-PKI, CoSi) do not directly address revocation at all, and do not discuss how revocation should be handled, by whom and under which conditions. Other PKI schemes use the X.509 notion of a certificate, and implicitly rely on the X.509 revocation mechanisms (CRLs and OCSP). This approach is somewhat understandable due to the pervasiveness of X.509, but also establishes the X.509 revocation mechanisms as the status quo of revocation, despite known weaknesses.

In Table 1, we label accountability, revocation accountability and non-revocation accountability as 'intuitively true' for all systems, except for CertCoin, Catena, PB-PKI, and CoSi. Thesep properties are typically achieved using a secure signing scheme, and therefore a formal proof seems straightforward and not essential. Note that CertCoin, PB-PKI and CONIKS allow clients to revoke their own certificates, but revocation can also be done by an adversary that compromised the client's secret keys, or alternatively, the client may be unable to perform revocation if the secret keys are lost.

Transparency, on the other hand, is supported by all post-X.509 PKI schemes, except CONIKS. The fact that transparency is so pervasively provided is likely in response to one of the main weaknesses of X.509 widely abused in practice, i.e., the lack of a mechanism to effectively propagate all issued certificates among CAs and clients. CONIKS, on the other hand, offers a limited notion of transparency of the identity / value map, which hides the actual identifiers and their corresponding values, as a trade-off between security and privacy. The clients can only query for individual identifiers. Furthermore, even that must be within a specific namespace, as CONIKS does not support global namespaces, where multiple CAs are authorized to issue for the same namespace. The use of separate namespaces, while problematic for the web PKI, works well for many applications such as chat rooms or messaging boards, that require secure key distribution but are under control of a single entity.

As Table 1 indicates, most previously-published PKI schemes have only informal security arguments for transparency. The exception are CT, DTKI, and

ARPKI, which have different types of automated proofs for scheme-specific properties. Namely, the properties and their proofs are not relevant to PKIs per se. Rather, they focus on details of the design of the particular scheme. Specifically, Dowling et al. [21] formalized security properties and provided reduction-based proofs for logging aspects of CT, that cover two classes of security goals involving malicious loggers and malicious monitors. Chase and Meiklejohn [16], on the other hand, focus on formalizing transparency through "transparency overlays", a generic construction they use to rigorously prove transparency in CT and Bitcoin. While their approach is elegant and can be used in other systems as a primitive that achieves transparency, it focuses on the "CT-style transparency" and does not consider other PKI properties such as revocation or non-equivocation.

Some of the systems, such as DTKI and ARPKI, verify their core security properties using automated symbolic proofs via the Tamarin prover [46]. Symbolic proofs provide an important added value for the security of proposed systems. Unfortunately, symbolic proofs often use abstractions; for example, in DTKI and ARPKI, a Merkle tree is modeled as a list. Such abstractions present an obstacle towards 'air-tight' security proofs. This strengthens the importance of a formal framework which on the one hand does not rely on specific implementations, yet, on the other, can be easily used by any implementation. We leave it to future work to explore ways to use symbolic proofs to add automatic verification capabilities to the framework described in this paper.

The post-X.509 safety requirements - transparency, non-equivocation and revocation transparency - are significantly more complex to understand, define and to achieve, compared to the X.509 properties of accountability and revocation accountability. Hence, we did not consider any of these post-X.509 properties to be 'intuitively true' - we believe they all require a proper definition and proof, as we provide in this paper; we spent considerable effort in properly defining these requirements in a precise and complete manner, and made every effort to keep things simple - but we admit that these definitions still require considerable effort to fully understand.

We separated between properties which are not-supported, and properties which are claimed to be supported using some security arguments. Note also that most systems do not discuss revocation transparency at all, even though in certain cases, e.g., CoSi, it seems relatively easy to achieve it. CT originally did not have a built-in support for revocation transparency, and it was only later formalized as Revocation Transparency [38].

3 Adversary-Driven Execution Process Framework

The security of distributed systems and cryptographic protocols should be stated and analyzed with respect to a specific *model*. By the term *model*, we mean a combination of *adversary capabilities*, *communication assumptions* and *clock-synchronization assumptions*:

Adversary capabilities: often referred to as the adversary model, define the computational resources of the adversary, e.g., probabilistic polynomial time

(PPT), as well as other capabilities, e.g., from cipher-text only (CTO) to chosen ciphertext attacks (CCA) (for encryption schemes).

Communication assumptions: properties of the underlying communication mechanism, such as reliable/unreliable communication, FIFO or non-FIFO, authenticated or not, bounded/fixed delay or asynchronous, and so on.

Clock-synchronization assumptions: define assumptions regarding the availability and properties of per-entity clocks. Common models include purely asynchronous clocks (no synchronization), bounded-drift clocks, and synchronized clocks.

However, research on cryptographic schemes and protocols often focuses on properties that do not depend on communication and clock synchronization. For example, when we define security of encryption schemes or zero-knowledge proofs, there is no significance to the properties of the communication between the parties, or to the synchronization of their clocks. Therefore, in such works, the definitions typically do not refer to communication and synchronization assumptions and only define the adversary's capabilities. However, communication and synchronization assumptions are crucial to the operation and security of distributed systems, including PKI schemes. In fact, several of the properties we discussed for PKI schemes involve *time*, and as such, are impacted by the communication and synchronization assumptions.

One way to address this challenge would be to adopt a specific, simple model for communication and synchronization, e.g., the synchronous 'rounds model', where all parties operate in lockstep, round by round, and messages sent at round i are delivered in round $i+1$. Using such a fixed model has the benefit of making it easy to focus on the cryptographic aspects, e.g., present rigorous definitions and prove security (by reductions). However, real systems are more complex; clocks are not fully synchronized, and communication is rarely, if ever, perfectly synchronous. This creates a dilemma in the design and analysis of cryptographic protocols; should we use a simplified model in order to focus on the cryptographic aspects of a protocol, or should we use a more realistic model to allow analysis of practical systems that must take communication and clock synchronization into account? While we are specifically interested in PKI schemes, this issue applies to other protocols as well. We now discuss our approach, which consists of the generic *adversary-driven execution process*, the definition of *model predicates*, and definition of *experiment-predicate based security requirements*.

The adversary-driven execution process $\mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$ *(Algorithm 1).* The execution process specifies the details of running a given protocol $\mathcal{P}$ with a given adversary $\mathcal{A}$, both modeled as efficient (PPT) functions. The execution process does not enforce any assumptions or model on the communication, synchronization, or inputs to the protocol; all of that is controlled entirely by the adversary $\mathcal{A}$. Furthermore, the execution allows the adversary to set and retrieve the state of any entity as well as messages sent and received, allowing for different failure models.

The model predicate $\mathcal{M}$. All model-specific details are handled by a *model predicate* $\mathcal{M}$, as follows. After the execution process finishes, it produces a

transcript of the execution; this transcript is input to $\mathcal{M}$. When $\mathcal{M}$returns $\top$ (*TRUE*), we say that the execution is *correct* according to $\mathcal{M}$, i.e., satisfies the model $\mathcal{M}$; otherwise, the execution is *invalid*, i.e., does not satisfy the model. This allows the model to specifies the adversary, communication and synchronization constraints, as well as protocol-specific assumption.

The definition of the model as a predicate, facilitates *modular* definition of models, as a conjunction of more basic model predicates. The models of both PKI schemes we analyze, are composed of specific sub-model predicates, each dealing with very specific aspects - the adversary model, the communication model, the synchronization model and, where necessary, models for protocol-specific assumption. This allows *reuse* of the same (sub)models by multiple schemes, making it easier to compare different schemes and protocols. See §3.2.

The security requirements. The execution process and the model, facilitate precise definition of security requirements, as predicates ('experiments') over the results of the execution. This is separate from the validation of the model, allowing modular specification of security requirements. Namely, different works may reuse the same security requirements (and execution process) but use other, possibly more realistic (and more complex) models, expressing different adversary capabilities, restrictions on usage, and assumptions on communication and synchronization. Similarly, different works may reuse the same models to study additional security requirements. In particular, we prove security for two PKI schemes under different adversary models, but with respect to the same security requirements. The separation between the definition of the model and of the requirements, also allows definition of *generic requirements.* Generic requirements are applicable to different protocols and problems. We identify four generic requirements, that appear relevant to many security protocols. These requirements focus on attribute of messages, i.e., non-repudiation, and on detection of misbehaving entities (see §3.3). This approach is quite different from the current way of defining security for cryptographic schemes and protocols; it takes some time and effort to get used to the separate model and security definitions. However, we found that with a little use, the advantages become clear and the approach becomes natural and convenient, facilitating modularity and reuse of requirements and models, and allowing for proper comparison of security guarantees between different schemes.

3.1 Design of Adversary-Driven Execution Process

We now present the Adversary-Driven Execution Process as defined by pseudocode in Algorithm 1. We explain and justify our design decisions as we discuss the specifics of the execution process.

The execution process $\mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$ has four inputs: the two algorithms (an adversary $\mathcal{A}$ and protocol $\mathcal{P}$), a unary security parameter 1^κ and a finite set N of *entities* (or *parties*). The protocol $\mathcal{P}$ refers to a specific protocol executed within the execution process given a specific adversary $\mathcal{A}$. A model $\mathcal{M}$ is not an input or a part of the process and only applied to its transcript after the process finishes, making it *oblivious* to the specific assumptions expressed in $\mathcal{M}$.

Algorithm 1 Adversary-Driven Execution Process $\mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$

1: $\forall i \in \mathsf{N}: \ s_i \leftarrow (sec = 1^\kappa, \mathsf{N}, \iota = i, s.initCounter = 0)$, $s_{\mathcal{A}} \leftarrow (1^\kappa, \mathsf{N})$
2: $t \leftarrow 1$
3: **repeat**
// $\mathcal{A}$ selects entity $i[t]$, operation $\alpha[t]$, and input $inp[t]$ and clock $clk[t]$.
4: $(i[t], \alpha[t], inp[t], clk[t]) \leftarrow \mathcal{A}(s_{\mathcal{A}})$
5: **if** $\alpha[t]$ ='Set' **then**
6: $(s_{i[t]}, out[t]) \leftarrow inp[t]$
7: $out[t] \leftarrow \perp$
8: **else if** $\alpha[t]$ ='Get' **then then** $out[t] \leftarrow s_{i[t]}$
9: **else** $(s_{i[t]}, out[t]) \leftarrow \mathcal{P}\left(s_{i[t]}, \alpha[t], inp[t], clk[t]\right)$
10: **end if**
// Update $s_{\mathcal{A}}$ and set $out_{\mathcal{A}}$ to either $\perp$ (to continue), or to other output (to terminate)
11: $(s_{\mathcal{A}}, out_{\mathcal{A}}) \leftarrow \mathcal{A}\left(s_{\mathcal{A}}, out[t]\right)$
12: $t \leftarrow t + 1$
13: **until** $out_{\mathcal{A}} \neq \perp$
14: $t \leftarrow t - 1$
// $\mathcal{A}$ selects the global, real time clock value $\tau[t]$ for each t
15: $\left\{\hat{t} \in \{1, \ldots, t\} \ : \ \tau[\hat{t}]\right\} \leftarrow \mathcal{A}(s_{\mathcal{A}})$
// Output:
16: $\mathsf{N_F} \leftarrow \left\{\iota \in \mathsf{N} \ : \ (\exists \hat{t}) i[\hat{t}] = \iota \wedge \alpha[\hat{t}] \in \{\text{'Set','Get'}\}\right\}$
17: $R \leftarrow \left\{\hat{t} \in \{1, \ldots, t\} \ : \ i[\hat{t}], \alpha[\hat{t}], inp[\hat{t}], clk[\hat{t}], \tau[\hat{t}], out[\hat{t}]\right\}$
18: Return $[\mathsf{N_F}, R, t, out_{\mathcal{A}}]$

To allow the process to apply to protocols with multiple functions and operations, we define the entire protocol $\mathcal{P}$ as a *single* PPT algorithm and use parameters to specify the exact operations and their inputs. Specifically, the protocol has four parameters: (s, α, inp, clk), where s is the *state* of the entity, α identifies the specific 'operation' or 'function' to be invoked, *inp* is the set of inputs to that operation/function, and *clk* is the current value of the clock of the entity. The protocol always outputs a pair (s, out), where s is the state of the entity *after* the operation, and *out* is the output. The execution process reserves two specific strings, 'Set' and 'Get' (lines 5-8), to define *generic operations*. We use these to allow the adversary to expose ('Get') the state of an entity, corresponding to a 'honest but curious' fault, and to control ('Set') the output of an entity, corresponding to a malicious fault; we discuss these further below. Namely, protocols should not use 'Set' and 'Get' as names of operations/functions.

Notation. We use the following notation in Algorithm 1: i is an entity from the set of all entities N, s_i is i's state, α is an operation to be executed, *inp* is an input value, *clk* is a local clock value, τ is a real clock value, and *out* is an output produced after some operation was executed. We use t to index execution steps within the execution process. When discussing only the input parameter *inp* of a specific protocol operation α, we often abuse notation and simply refer to it as $\mathcal{P}.\alpha(inp)$. This is especially useful for any *stateless* functions, i.e., functions that do not depend on or modify the state; such functions are defined as part

of the protocol, but may be applied outside of the execution process, e.g., to define experiments and requirements. We use a *dot notation* to refer to elements of 'structures', and an *index notation* to refer to cells of 'arrays'. For example, $R.out[t]$ refers to the value of the t^{th} entry of the array $R.out$, which is part of the event log R.

We now discuss the three main components of the execution process, that is, the initialization, main loop execution and termination.

Initialization. In line 1, we set the initial state for each entity i and the adversary $\mathcal{A}$, s_i and $s_{\mathcal{A}}$ respectively. In line 2, we set the initial value of t, which we use to index the operations (steps) of the execution, i.e., increment by one (line 12) each time we complete one 'execution loop' (lines 3-13). The value t does not represent any clock value and is independent of any clock synchronization model, and specifically, is not controlled by the adversary. Rather, t allows us to index a sequence of operations performed within one execution loop, and precisely refer to each step of the execution. The local clock clk and real time clock τ are used to represent and model different clock assumptions.

Main execution loop (lines 3-13). The design allows the adversary $\mathcal{A}$ to have a generous control over the execution. Specifically, in each step t, $\mathcal{A}$ determines (line 4) an operation $\alpha[t]$ to be applied to an entity $i[t] \in \mathsf{N}$, with input $inp[t]$ and its local clock value $clk[t]$. Additionally, in line 15, the adversary selects the global, real time clock value $\tau[t]$. The adversary selects $\tau[t]$ after the main loop run since it is not needed earlier and it may want to decide on these values based on the run, allowing for greater flexibility and control over the execution. We use both $clk[t]$ and $\tau[t]$ to accommodate different clock synchronization models. The execution process does not place any restrictions on these values and enforcing any constraints is left to the appropriate clock synchronization model. The specific clock synchronization model provided in this work ensures that $\tau[t]$ is monotonously increasing and uses it to enforce other communication and clock properties (see Sections 3.2.3 and 3.2.4).

After the adversary defines the specific operation and inputs (line 4), the event is executed (lines 5-9). There are three options for each event specified as an operation $\alpha[t]$. More concretely, $\alpha[t] =$ 'Set' (lines 5-7) lets the adversary set the state $s_{i[t]}$ of entity $i[t]$ (to $inp[t]$); $\alpha[t] =$ 'Get' (line 8) outputs the state $s_{i[t]}$ of entity $i[t]$, including any private state, e.g., private keys, by setting it into $out[t]$; and $\alpha[t]$ set to any other operation (line 9) invokes the protocol $\mathcal{P}$ specific function over the state $s_{i[t]}$ of entity $i[t]$, with inputs $\alpha[t], inp[t], clk[t]$. This results in a new state $s_{i[t]}$ and output $out[t]$ for entity $i[t]$.

The 'Set' and 'Get' operations are particularly important as the adversary uses them to control and interact with the entities by being able to define (set) and retrieve (get) their state. Certain other operations may be defined and used in the model predicate $\mathcal{M}$ to enforce the protocol assumptions. Specifically, we use 'Init', 'Incoming', 'Sleep', 'Wake-up' in the communication and synchronization models to ensure that communication and time-driven events invoke proper handler functions (see Sections 3.2.2, 3.2.3 and 3.2.4 for details).

The execution process allows the adversary to set and get the state of any entity; however, a specific execution model $\mathcal{M}$ may forbid such operations, e.g., return $\perp$ for executions where the adversary performs them. In line 11, the adversary processes the output $out[t]$ of the protocol. The adversary may modify its state $s_{\mathcal{A}}$, and outputs a value $out_{\mathcal{A}}$; when $out_{\mathcal{A}} \neq \perp$, the execution moves to the termination phase; otherwise the loop continues.

Termination (lines 14-18). Upon termination, the process returns the relevant values from the execution. Most of these values are in the *events log* R (line 17), namely, the values of $i[t'], \alpha[t'], inp[t'], clk[t']$, $\tau[t']$ and $out[t']$, for each of the rounds $t' \leq t$, where t is the index of the last round, set in line 14. Private values such as entity's private keys are not part of the output unless provided as $inp[t']$ for $\alpha[t'] =$ 'Set' or properly extracted using $\alpha[t'] =$ 'Get'. In addition to R, the execution also returns the set of faulty entities $\mathsf{N_F}$ (produced in line 16), the index of the last round t, the adversary's output $out_{\mathcal{A}}$.

Limitations. The execution process supports a large variety of models. For example, the adversary may control ('Set') and learn ('Get') the state of every party $i[t] \in \mathsf{N}$ but this may be restricted (or fully prohibited) by specific model $\mathcal{M}$, allowing different fault models (honest-but curious, threshold, adaptive, proactive, etc.). However, for simplicity and focus, some generalizations are left for future work. In particular, we allow the adversary to control all inputs events, while typical definitions of confidentiality and indistinguishability requirements provide randomized inputs which are not directly observable by the adversary. Similarly, we allow the adversary to *control and observe all communication events*; a specific model predicate $\mathcal{M}$ may prohibit the adversary from changing the messages, but an extension is required to prevent the adversary from even observing the communication (e.g., if anonymous or confidential channels must be assumed without providing appropriate mechanisms at the protocol level). We leave extending the execution model to allow such additional generalities to future work.

3.2 The Model Predicates

The execution process described in Algorithm 1 specifies the details of running a protocol $\mathcal{P}$ against an adversary $\mathcal{A}$; however, it does not, on its own, *restrict* the adversary. An essential part of any security analysis, is to define the *adversary model*, i.e., the exact capabilities of the adversary. In this subsection, we present different *model predicates*, which precisely define a model for the adversary. We use the model predicates to restrict the capabilities of the adversary as well as the events that can happen in the execution process. This includes limiting of the possible faults, defining initialization assumptions, and defining the communication and synchronization models. Our approach allows to analyze schemes designed for different model predicates $\mathcal{M}$, and to define the adversary capabilities as well as the communication and synchronization assumptions, separately from the specification of the protocol and from the security definitions.

We enforce restrictions on the adversary $\mathcal{A}$ by applying a model $\mathcal{M}$ to $\mathcal{A}$ and ensuring that $\mathcal{A}$ *satisfies* $\mathcal{M}$, as defined below. Let both $\mathcal{A}$ and $\mathcal{M}$ be PPT algorithms, where $\mathcal{M}$ is a predicate (i.e., outputs $\top$ or $\bot$).

Definition 1 (Model-satisfying adversary). *We say that adversary $\mathcal{A}$ satisfies model $\mathcal{M}$, if for every protocol $\mathcal{P}$ and every set of entities N holds:*

$$Pr\left[\mathcal{M}(1^\kappa, \mathsf{N}, \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})) = \bot\right] \in Negl(1^\kappa)$$

We define and use several model predicates. Most of these predicates focus on some specific aspect of the model, e.g., $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$, which defines a typical clock-synchronization model (with bounded-drift from global real time). By separately defining different aspects of the assumptions made, we can precisely define the models and then reuse them in the analysis of different protocols and even different problems, allowing combinations of new and known model predicates as necessary.

Before we proceed to present the individual model predicates, we first describe and discuss $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, the 'composite' model predicate, which encompasses adversary, clock synchronization and communication assumptions, that we use in the analysis of the PoC-PKI PKI scheme; see §5.1 for the description of PoC-PKI, and proofs of security under $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$. Of course, $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ is quite specific for PoC-PKI; and applications and schemes, e.g., CT_{comp}, require different models. However, most of the 'sub-models' of PoC-PKI, are generic and can be used in definition of models for other tasks and schemes; indeed, the model of CT_{comp} is based on the same component models.

$\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ is defined as the conjunction of four component model predicates, each focusing on a different aspect: $\mathcal{M}^{\hat{r}\text{-rounds}}_{\text{SecInit}}$, modeling a trusted-setup (initialization) with an authenticated message-response exchange, $\mathcal{M}^{|\mathsf{N_F}|\leq f}$, an f-Byzantine faults model, $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$, an authentic-sender, bounded-delay communications model, and $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$, a bounded-drift clock-synchronization model. Namely:

$$\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}(\xi) \equiv \mathcal{M}^{\text{3-rounds}}_{\text{SecInit}} \wedge \mathcal{M}^{|\mathsf{N_F}|\leq\lfloor(|\mathsf{N}|/3)\rfloor}(\xi) \wedge \mathcal{M}^{\text{COM}}_{\Delta_{com}}(\xi) \wedge \mathcal{M}^{\text{CLK}}_{\Delta_{clk}}(\xi) \quad (1)$$

where we used ξ as a shorthand for $(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}})$, the inputs provided to the model predicate. Notice the specific parameters, $\hat{r} = 3$ and $f = \lfloor(|\mathsf{N}|/3)\rfloor$.

3.2.1 $\mathcal{M}^{|\mathsf{N_F}|\leq f}$: up to f Byzantine (malicious) faults model

We next define $\mathcal{M}^{|\mathsf{N_F}|\leq f}$, a specific adversary model allowing the adversary to choose, and completely control, up to f of the entities in N. We refer to such failures, where the adversary completely controls the entity, as *malicious* or *Byzantine* faults. We use f as a function applied to the total number of entities $|\mathsf{N}|$. We refer to this particular faults model as $\mathcal{M}^{|\mathsf{N_F}|\leq f}$, where $f : \mathbb{N} \to \mathbb{N}$ bounds the number of faulty entities as a function of the total number of entities. Specifically, the adversary may corrupt entities by performing the 'Get' and 'Set' operations. To enforce the model, we simply ensure that 'Get' and 'Set'

operations can be applied only to entities in $\mathsf{N_F}$, and that $|\mathsf{N_F}| \leq f(|\mathsf{N}|)$. We define $\mathcal{M}^{|\mathsf{N_F}|\leq f}$ as:

$$\mathcal{M}^{|\mathsf{N_F}|\leq f}(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = (|\mathsf{N_F}| \leq f\,(|\mathsf{N}|)) \wedge (\forall t)(R.\alpha[t] \in \{\text{`Get', `Set'}\} \Rightarrow i[t] \in \mathsf{N_F}) \tag{2}$$

PoC-PKI, our proof-of-concept PKI, uses $\mathcal{M}^{|\mathsf{N_F}|\leq f}$ with f defined as: $f(n) = \lfloor (n/3) \rfloor$. Namely, the faulty entities can arbitrarily misbehave so long the adversary controls at most a third of all entities. Note that $\mathcal{M}^{|\mathsf{N_F}|\leq f}$ only restricts the set $\mathsf{N_F}$ of faulty entities, which does not yet restrict the adversary's ability to interfere with the communication, clock and local inputs of any entity - including non-faulty entities (in $\mathsf{N} - \mathsf{N_F}$). By using $\mathcal{M}^{|\mathsf{N_F}|\leq f}$ in conjunction with additional model predicates, e.g., $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$ and $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$ (described next), we can also restrict the adversary in such additional ways, as desired for a particular analysis. Notice, in particular, that $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$, which defines restrictions on the communication events, completely excludes events where the sender or recipient are faulty (i.e., in $\mathsf{N_F}$). This allows the adversary to completely control all of faulty entities.

3.2.2 $\mathcal{M}^{\hat{r}\text{-rounds}}_{\text{SecInit}}$: the $\hat{r}$-rounds secure initialization model.

Cryptographic protocols are often designed assuming a secure initialization process, e.g., assuming shared secret keys. However, in the execution process (Algorithm 1), entities can only communicate via the adversary. As a result, we cannot simply *assume* shared secret keys but the entities can only use their local randomness to generate secret keys, and communicate, using cryptography, to securely establish shared secret values. We next define a simple secure initialization model, $\mathcal{M}^{\hat{r}\text{-rounds}}_{\text{SecInit}}$. This model ensures $\hat{r}$ secure 'rounds' of $|\mathsf{N}|$ steps each, where in step t (where $1 \leq t \leq \hat{r} \cdot |\mathsf{N}|$) holds:

- Entities are invoked with the special operation 'Init', i.e., $\alpha[t] =$ 'Init', and in 'round robin', i.e., $i[t] = t \mod |\mathsf{N}|$. Note, in particular, that this prevents the adversary, during the initialization, from invoking the special 'Set' and 'Get' operations, to control the state or output of an entity ('Set') or to expose the state of an entity ('Get').
- Authenticated, reliable communication. Namely, every message received by entity i from entity j at round $2 \leq r \leq \hat{r}$, was indeed sent by j in the previous round to i; and vice verse, i.e., every message sent by i to j at round $1 \leq r \leq (\hat{r} - 1)$, is correctly received by j, from sender i, in the next round.

It is convenient to capture each of these two aspects by a separate model predicate, i.e., $\mathcal{M}^{\hat{r}\text{-rounds}}_{\text{SecInit}}(\xi) = \mathcal{M}^{\hat{r}\text{-rounds}}_{\text{InitOps}}(\xi) \wedge \mathcal{M}^{\hat{r}\text{-rounds}}_{\text{InitCom}}(\xi)$, where $\mathcal{M}^{\hat{r}\text{-rounds}}_{\text{InitOps}}$ captures the first aspect ('operations') and $\mathcal{M}^{\hat{r}\text{-rounds}}_{\text{InitCom}}$ captures the second aspect ('communications'). We now define each of these more precisely; for convenience, let $\mathsf{N} = \{1, 2, \ldots\}$:

$$\mathcal{M}^{\hat{r}\text{-rounds}}_{\text{InitOps}}\,(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \begin{pmatrix} \forall i' \in \mathsf{N},\ r \in \{0, \ldots, \hat{r} - 1\} : \\ (R.i[i' + r \cdot |\mathsf{N}|] = i') \wedge (R.\alpha[i' + r \cdot |\mathsf{N}|] = \text{'Init'}) \end{pmatrix}$$

$$\mathcal{M}^{\hat{r}\text{-rounds}}_{\text{InitCom}}\ (1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \\ = \begin{pmatrix} \forall i', j \in \mathsf{N},\ r' \in \{0, \ldots, \hat{r}-2\},\ m \in \{0,1\}^* \ : \\ (j,m) \in R.out[i' + r' \cdot |\mathsf{N}|] \Leftrightarrow (i',m) \in R.inp[j + (r'+1) \cdot |\mathsf{N}|]) \end{pmatrix}$$

Different protocols may require different number of initialization rounds. In particular, we present and analyze two PKI schemes: PoC-PKI requires three rounds and CT_{comp} requires only two. And, while the initialization model seems appropriate for many protocols, there are surely many protocols which require a different initialization model.

3.2.3 $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$: authentic-sender, bounded-delay communication model

We next present $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$, an authentic-sender, bounded-delay communication model. It is convenient to define $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$ as a conjunction of two simpler predicates: $\mathcal{M}^{\text{COM-rcv}}_{\Delta_{com}}$, ensuring authentic-sender for message-receive events, and $\mathcal{M}^{\text{COM-send}}_{\Delta_{com}}$, ensuring reliable, bounded-delay for message-send events. Namely:

$$\mathcal{M}^{\text{COM}}_{\Delta_{com}}(\xi) = \mathcal{M}^{\text{COM-send}}_{\Delta_{com}}(\xi) \wedge \mathcal{M}^{\text{COM-rcv}}_{\Delta_{com}}(\xi)$$

We first present $\mathcal{M}^{\text{COM-rcv}}_{\Delta_{com}}$, which ensures authentic-sender for message-receive events. The adversary decides on the function $\alpha[t_r]$ to be invoked at every round t_r as well as the input $inp[t_r]$. To cause a message receipt event, the adversary sets $\alpha[t_r]$ to the special operation 'Incoming', and the input $inp[t_r]$ to the pair (m, j) where m is the message and $j \in \mathsf{N}$ is the purported sender. We use dot notation to refer to the message ($inp[t_r].m$) and to the sender ($inp[t_r].j$).

The authentic-sender property ($\mathcal{M}^{\text{COM-rcv}}_{\Delta_{com}}$ model) implies that $inp[t_r].j$ indeed sent this message to $i[t_r]$, during some previous round $t_s < t$. We allow the sender $i[t_s]$ to specify, as part of its output $out[t_s]$, one or more triplets of the form $(\text{'send'}, m, j)$, specifying sending of message m to $j \in \mathsf{N}$. The $\mathcal{M}^{\text{COM-rcv}}_{\Delta_{com}}$ model follows:

$$\mathcal{M}^{\text{COM-rcv}}_{\Delta_{com}}\ (1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \\ = \begin{bmatrix} (\forall t_r\ s.t.\ R.\alpha[t_r] = \text{'Incoming'})(R.i[t_r], R.inp[t_r].j \in \mathsf{N} - \mathsf{N_F}) \Rightarrow \\ \exists t_s < t_r\ s.t.\ (\text{'send'}, R.inp[t_r].m, R.i[t_r]) \in R.out[t_s]\ \wedge R.i[t_s] = R.inp[t_r].j \end{bmatrix}$$

The $\mathcal{M}^{\text{COM-send}}_{\Delta_{com}}$ model ensures reliable, bounded-delay delivery of messages sent. Assume that at round t_s of the execution, the output $out[t_s]$ generated by $i[t_s]$, includes a $(\text{'send'}, m, j)$ triplet, i.e., $i[t_s]$ sends message m to $j \in \mathsf{N}$. If the $\mathcal{M}^{\text{COM-send}}_{\Delta_{com}}$ model is true for this execution, then after at most Δ_{com}, if the execution did not terminate already, then entity j would receive m from $i[t_s]$. Namely:

$$\mathcal{M}^{\text{COM-send}}_{\Delta_{com}}(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \left[\begin{pmatrix} (\forall t_s < t)\ (\exists(\text{'send'}, m, j) \in R.out[t_s]) \wedge \\ \wedge (R.\tau[t] \geq R.\tau[t_s] + \Delta_{com}) \wedge (i[t_s] \in \mathsf{N} - \mathsf{N_F}) \end{pmatrix} \Rightarrow \begin{pmatrix} \exists t_r > t_s \ s.t.\ R.\tau[t_s] + \Delta_{com} \geq R.\tau[t_r] \wedge \\ \wedge R.inp[t_r] = (m, R.i[t_s]) \end{pmatrix} \right]$$

We remark that: $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$ only applies when both sender and recipient are non-faulty (i.e., in $\mathsf{N} - \mathsf{N_F}$); $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$ only ensures delivery, sender authentication and bounded delay. This still allows receipt of duplicate messages, which may involve unbounded delay. To simplify $\mathcal{M}^{\text{COM-send}}_{\Delta_{com}}$, we use the adversary-controlled $\tau[\cdot]$ values (line 6 of Algorithm 1). For this to be meaningful, we depend on the synchronization properties of the $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$ model, discussed next.

3.2.4 $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$: bounded-drift clock synchronization assumptions

Finally, we present $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$ which models the bounded-drift clock synchronization assumptions. We split this into two predicates, $\mathcal{M}^{\text{Drift}}_{\Delta_{clk}}$ which limits the drift between the clock values $clk[t]$ which the adversary provides as input to the protocol, and the 'real time' values $\tau[t]$; and $\mathcal{M}^{\text{Wakeup}}_{\Delta_{clk}}$, which provides a 'wake-up service' to the protocol. Namely:

$$\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}(\xi) = \mathcal{M}^{\text{Drift}}_{\Delta_{clk}}(\xi) \wedge \mathcal{M}^{\text{Wakeup}}_{\Delta_{clk}}(\xi)$$

We begin with $\mathcal{M}^{\text{Drift}}_{\Delta_{clk}}$, which bounds the clock drift. It enforces two requirements on the execution: each local-clock value ($clk[t]$) must be within Δ_{clk} drift from the 'real-time' $\tau[t]$, and the real-time values should be monotonously increasing. Namely:

$$\mathcal{M}^{\text{Drift}}_{\Delta_{clk}}(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \left[\begin{array}{l} (\forall t' \leq t)(|R.clk[t'] - R.\tau[t']| \leq \Delta_{clk}) \wedge \\ (R.\tau[t'-1] \neq \perp) \Rightarrow (R.\tau[t'] \geq R.\tau[t'-1]) \end{array} \right]$$

As a special case, when $\Delta_{clk} = 0$, this function defines a model where the local clocks are fully synchronized, i.e., there is no difference between entities' clocks. Finally, $\mathcal{M}^{\text{Wakeup}}_{\Delta_{clk}}$ provides a 'wake-up service' allowing the protocol to perform time-driven activities and ensuring that appropriate functions are invoked properly.

$$\mathcal{M}^{\text{Wakeup}}_{\Delta_{clk}}(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \left[\begin{array}{c} \begin{pmatrix} (\forall t' \leq t)\ ((\textit{'Sleep'}, x) \in R.out[t']) \wedge \\ \wedge\ (\exists r > t')(R.\tau[r] \geq R.\tau[t'] + x + \Delta_{clk}) \end{pmatrix} \\ \Rightarrow \\ \begin{pmatrix} (\exists t'' > t')(|R.\tau[t''] - R.\tau[t'] - x| \leq \Delta_{clk}) \\ \wedge\ (R.i[t'] = R.i[t])\ \wedge\ (R.\alpha[t'] = \text{'Wake-up'}) \end{pmatrix} \end{array} \right]$$

3.3 Model-Secure Requirements

To complete the presentation of the execution process, we now discuss how it is used to define specific security requirements and properties, and to analyze whether these properties are ensured by a given protocol $\mathcal{P}$, under given model $\mathcal{M}$, interacting with any PPT adversary $\mathcal{A}$. While we present *game/experiment*-based definitions, future work may consider other forms of definitions, such as simulation-based.

A protocol $\mathcal{P}$ would typically have multiple security properties, i.e., satisfy multiple security requirements. We define a *security requirement* as a pair $(\xi, \mathbf{Exp}^{\xi}_{\mathcal{P}})$, where ξ is the *name* of the requirement and $\mathbf{Exp}^{\xi}_{\mathcal{P}}$ is an efficiently computable predicate, which we refer to as the ξ *experiment*. Let b be the outcome of the ξ experiment predicate $\mathbf{Exp}^{\xi}_{\mathcal{P}}$, applied to the inputs and outputs of the execution process, i.e., $b \leftarrow \mathbf{Exp}^{\xi}_{\mathcal{P}}(1^{\kappa}, \mathsf{N}, \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^{\kappa}, \mathsf{N}))$; if $b = \top$ then we say that *requirement ξ was not satisfied* in this execution of $\mathcal{P}$, or that the *adversary won* in this execution; and if $b = \bot$, then we say that *requirement ξ was satisfied* in this execution, or that the *adversary lost*. We say that protocol $\mathcal{P}$ ensures requirement ξ under model $\mathcal{M}$, if for any PPT adversary $\mathcal{A}$, there is only negligible probability of an execution in which $\mathcal{A}$ wins, i.e., which does not satisfy ξ, i.e., where $\mathbf{Exp}^{\xi}_{\mathcal{P}}$ returns $\top$.

Definition 2 (Protocol $\mathcal{P}$ ensures requirement ξ under model $\mathcal{M}$). *Let $(\xi, \mathbf{Exp}^{\xi}_{\mathcal{P}})$ be a security requirement, i.e., a name ξ and an efficiently computable predicate $\mathbf{Exp}^{\xi}_{\mathcal{P}}$. We say that protocol $\mathcal{P}$ ensures requirement $(\xi, \mathbf{Exp}^{\xi}_{\mathcal{P}})$ under model $\mathcal{M}$, or, abusing notation, that $\mathcal{P}$ ensures ξ under $\mathcal{M}$, if for every PPT adversary $\mathcal{A}$ that satisfies $\mathcal{M}$ (Definition 1), and for every set N, holds:*

$$Pr\left[\begin{array}{l}(\mathsf{N_F}, R, t, out_{\mathcal{A}}) \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^{\kappa}, \mathsf{N}) \\ \mathbf{Exp}^{\xi}_{\mathcal{P}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \top\end{array}\right] \in Negl(1^{\kappa})$$

Most of the requirements depend on the specific cryptographic scheme or protocol; specifically, in §4, we describe such requirements for PKI protocols. However, there are some *generic* security requirements, which apply to many protocols. We next identify a few generic requirements, which apply to PKI schemes, but also appear applicable to other security protocols, focusing on the basic security goals of *detection of misbehaving entities* and *attribution of statements*.

3.3.1 Verifiable Attribution of Statements (VAS) Requirement

The output of many protocols may include *attributable statements*. An *attributable statement* is a tuple (m, σ, ι), where m is string, $\iota \in \mathsf{N}$ is the *purported origin* of the statement, and σ provides *evidence* (typically, a signature), allowing attribution of statement m to entity ι. We next explain the *validation* process, which uses the evidence σ to establish if ι has, in fact, originated m.

We focus on the typical case, where attribution is based on the use of a *digital signature scheme* $\mathcal{S}$ (Appendix A.2), applied by the protocol $\mathcal{P}$. Namely,

σ is the result of applying the signing algorithm $\mathcal{S}.Sign$ to the message m, using some (private) signing key sk belonging to the origin ι. Therefore, we say that the attributable statement (m, σ, ι) is *valid*, i.e., that σ really 'proves' that ι is the origin of m, if $\mathcal{S}.\mathsf{Ver}(pk, m, \sigma) = \top$, where pk is the public signature-verification key of ι, i.e., the public key that validates signatures computed using sk. This *attributes* the message m to the 'owner' of the public key pk (and the corresponding signing key sk). To attribute m to ι, it remains to establish the association between ι and the public key pk, i.e., to attribute pk, and messages verified by it, to ι. We focus on protocols where this association is known and secure ('off-band'), e.g., CA public keys in PKI schemes.

We formalize this by assuming that each entity $\iota \in \mathsf{N}$ *identifies* its public key pk by outputting the pair $(\text{'public key'}, pk) \in out[\hat{t}]$, in some step $\hat{t}$; namely, we use 'public key' as a 'label', to identify output of the public key. Typically, entities output the public key when they generate the key, i.e., $i[\hat{t}] = \iota$, possibly as an initialization operation, i.e. $\alpha[\hat{t}] =$ *'Init'*. Notice that entities may often also send their public keys to each other, using the $(\text{'send'}, m, j)$ output convention described in § 3.2.3; however, we prefer to keep the two conventions separate, since we believe that not every protocol that uses verification of attribution, would necessarily send public keys in precisely the same way.

More precisely, the following *Key Attribution Predicate* V_{ka} outputs $\top$ if entity ι has identified pk as its public key, in a given log of events R output of an execution of the protocol $\mathcal{P}$ (Algorithm 1):

$$V_{ka}(\iota, pk, R) = \left\{\exists \hat{t} \ s.t. \ R.i[\hat{t}] = \iota \ \wedge (\text{'public key'}, pk) \in R.out[t]\right\} \tag{3}$$

We now define the Verifiable Attribution of Statements requirement, and the corresponding experiment. The adversary $\mathcal{A}$ 'wins' in the experiment, if its output $out_{\mathcal{A}}$ includes both a valid attributable statement (m, σ, ι) for non-faulty entity $\iota \in \mathsf{N} - \mathsf{N_F}$, and a verification key pk associated with ι, yet ι did *not* originate m. To allow us to identify events $\hat{t}$ in which an entity $\iota = i[\hat{t}]$ intentionally signed message m, we adopt the following convention: whenever signing a message m, the party adds the pair $(\text{'signed'}, m)$ as part of its output, i.e., $(\text{'signed'}, m) \in out[\hat{t}]$. Since this is *always* done, whenever the protocol signs a message, we will *not* explicitly include the $(\text{'signed'}, m)$ pairs as part of the output, which would make the pseudo-code cumbersome. Note that often the entity will also send the signed message, however, different protocols may send in different ways, hence this convention makes it easier to define the requirement.

The requirement is defined with respect to specific signature scheme $\mathcal{S}$, and the V_{ka} predicate define above (Eq. 3). For simplicity, and since $\mathcal{S}$ is typically obvious (as part of $\mathcal{P}$), we do not explicitly specify $\mathcal{S}$ as a parameter of the requirement.

Requirement 1. *Verifiable Attribution of Statements (VAS). Protocol $\mathcal{P}$ ensures VAS, for signature scheme $\mathcal{S}$, if any PPT adversary $\mathcal{A}$ would have negligible probability to win in the Verifiable Attribution of Statements experiment*

$\mathbf{Exp}^{VAS}_{\mathcal{P}}$, *i.e., to cause its value to be* $\perp$*:*

$$\mathbf{Exp}^{VAS}_{\mathcal{P}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \begin{bmatrix} ((m, \sigma, \iota), pk) \leftarrow out_{\mathcal{A}} \ ; \iota \in \mathsf{N} - \mathsf{N_F} \wedge \\ \mathcal{S}.\mathsf{Ver}(pk, m, \sigma) = \top \wedge \ V_{ka}(\iota, pk, R) = \top \wedge \\ \nexists \ \hat{t} \ s.t. \ i[\hat{t}] = \iota \ \wedge (\textit{'signed'}, m) \in out[\hat{t}] \end{bmatrix}$$

3.3.2 Generic misbehavior detection requirements

Many security protocols are required to be *resilient to misbehaviors*, i.e., to achieve their goals, even if some of the entities, say entities in $\mathsf{N_F} \subset \mathsf{N}$, are faulty, and may misbehave (arbitrarily or in some specified manner). This resiliency to faulty, misbehaving entities, is often based on *detection of misbehavior*; furthermore, often, many security protocols are required only to *detect misbehaviors*, which would be followed by taking some additional measures to deter and/or neutralize an attack. While misbehavior can be detected in different ways, detection is typically based either on some *evidence* that a certain entity is dishonest, where the evidence should be *verifiable by any third party*, or based on an *accusation*, where one entity (the *accuser*) accuses another entity (the *suspect*) of some misbehavior. Such an accusation may not be true, and therefore, it is harder to use this approach to deter and/or neutralize the attack; however, many misbehaviors do not leave any *evidence* verifiable by a third party, in which case, accusations may provide some security benefits, e.g., *detection* of the attack. A typical example of such misbehavior that does not leave any evidence, is when a party *fails to act* in a required way, e.g., to send a required message or response; such failure may be plausibly blamed on communication issues, or on failure of the intended recipient. Often, a party, say Alice, detects such failure, say of Mal, to send a required message, after Alice waits for some *maximum delay*, and then Alice issues an 'accusation' against Mal, to alert others; for example, see [40]. An honest entity would only accuse a misbehaving party; however, because an accusation cannot be verified, a misbehaving entity could falsely accuse anyone, even an honest entity.

To formalize these concepts, we define two requirements: one to ensure that honest entities cannot be 'framed' as misbehaving, i.e., evidences are always verifiable with correct outcome, and another one to express that honest entities never accuse other honest entities, i.e., only accuse misbehaving entities.

The Non-frameability requirement and Proof of Misbehavior. The first security requirement is called *non-frameability* (of honest entities), and ensures that a specific protocol would not allow any entity to produce a *valid Proof of Misbehavior* of a non-faulty entity. The requirement is therefore defined with respect to a given *Proof of Misbehavior Validation Predicate* V_p, which receives two inputs: a Proof-Validation Key pk, and a purported-proof ζ. The output of $V_p(pk, \zeta)$ is $\top$, if and only if ζ is a valid Proof of Misbehavior, as indicated by pk; i.e., a misbehavior by an entity who knows the corresponding private key, typically, the 'owner' of pk, which can be validated using the Key Attribution Predicate V_{ka}. The natural way is to define the Proof of Misbehavior Validation Predicate V_p, to be *protocol specific*, as the notions of misbehavior, and valid

proof of misbehavior, depend on the specific protocol requirements. We specify for $\mathcal{P}$ a special *stateless* operation $\alpha =$'V_p', which does not modify the state or depend on it, or on the local clock. Abusing notation, we denote this operation simply as $\mathcal{P}.V_p(pk, \zeta)$. The use of a protocol-defined $\mathcal{P}.V_p$ allows us to define, below, the Non-frameability requirement.

Requirement 2. *Non-frameability (NF). The adversary should have negligible probability to win in the Non-frameability (NF) experiment, defined as follows, i.e., to output a Proof of Misbehavior for an honest entity. Let* $V_p : \{0,1\}^* \times \{0,1\}^* \rightarrow \{\top, \bot\}$ *be a predicate. We define the non-frameability experiment as follows:*

$$\mathbf{Exp}^{NF}_{\mathcal{P}}(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \{(\iota, \zeta, pk) \leftarrow out_{\mathcal{A}} \; ; \; V_{ka}(\iota, pk, R) \wedge \mathcal{P}.V_p(pk, \zeta) \wedge \iota \notin \mathsf{N_F}\} \tag{4}$$

Note: we present two PKI schemes: CT_{comp}, which relies on Proof of Misbehavior as well as on accusations, and PoC-PKI, which does not depend on detection mechanisms, since it *prevents* the corresponding attack, by relying on majority (and threshold signatures). For prevention-based protocol such as PoC-PKI, the Proof of Misbehavior Validation Predicate (V_p) always returns $\bot$, hence, the requirement is trivially satisfied.

Accusations and the No False Accusations requirement. Recall that, in the execution process, the adversary can use the 'Set' operation to set the output and the state of a party; we refer to such party as *faulty*, and denote by $\mathsf{N_F}$ the set of faulty parties in an execution. In many protocols, one party, say Alice, may *detect* that another party, say Mal, is faulty, typically, by receiving an invalid message from Mal - or simply by *not* receiving a message expected from Mal by a specific 'deadline' (for bounded-delay communication models).

Intuitively, the *No False Accusations (NFA)* requirement states that a non-faulty entity $a \notin \mathsf{N_F}$ (Alice), would *never (falsely) accuse* of a fault, another non-faulty entity, $b \notin \mathsf{N_F}$ (Bob). To properly define this requirement, we first define a convention for one party, say $a \in \mathsf{N}$ (for 'Alice'), to output an Indicator of Accusation, i.e., 'accuse' another party, say $m \in \mathsf{N}$ (for 'Mal'), of a fault. Specifically, we say that at step t_A of the the execution, entity $i[t_A]$ *accuses* entity m (Mal), if $out[t_A]$ is a triplet of the form (IA, m, x). The last value in this triplet, x, should contain the clock value at the *first* time that Alice accused Mal; we discuss this after the requirement, as the value x is not relevant for the requirement, and just used as a convenient convention for some protocols.

Requirement 3. *No False Accusations (NFA). The adversary should have negligible probability to win in the No False Accusations (NFA) experiment* $\mathbf{Exp}^{NFA}_{\mathcal{P}}$*, defined as follows, i.e., to cause one honest entity, say Alice, to accuse another honest entity, say Bob (i.e., both Alice and Bob are in* $\mathsf{N} - \mathsf{N_F}$*). Namely,* $\mathbf{Exp}^{NFA}_{\mathcal{P}}(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}})$ *returns* $\bot$ *only if* $out[t] = (IA, j, x)$*, for some* $j \in \mathsf{N}$*, and there is no step* t_1 *in which the adversary 'corrupted' either* j *or* $i[t]$*, i.e., where* $\alpha[t_1] =$ *'Set' and* $i[t_1] \in \{i[t], j\}$*. More precisely:*

$$\mathbf{Exp}^{NFA}_{\mathcal{P}}(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \{(i[t] \notin \mathsf{N_F}) \wedge (\exists j \in \mathsf{N} - \mathsf{N_F}, x)((IA, j, x) \in out[t])\} \tag{5}$$

As noted above, in an accusation, the output $out[t_A]$ contains a triplet of the form (IA, m, x), where x is a clock value, and *should* contain the clock value at the *first* time that Alice accused Mal. We found this convenient in definition of protocol-specific requirements, where a party may accuse another party multiple times, and the requirement is related to the time of the *first* accuse event. To allow the use of this convention, we define the following 'technical' requirement and experiment, which merely confirms that honest entities always indicate, in any accuse event, the time of the first time they accused the same entity.

Requirement 4. *Use First-Accuse Time (UFAT). The adversary should have negligible probability to win in the Use First-Accuse Time (UFAT) experiment* $\mathbf{Exp}_{\mathcal{P}}^{UFAT}$*, defined as follows. To simplify the experiment, let* $fc(i, m, R)$ *be the value of* $clk[t']$*, where* t' *is the first event in* R *in which entity* i *accused entity* $m \in \mathsf{N}$ *(or* $\perp$ *if no such event exists).*

$$\mathbf{Exp}_{\mathcal{P}}^{UFAT}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \{(i[t] \notin \mathsf{N_F}) \wedge (\exists m \in \mathsf{N})(out[t] = (IA, m, x) \wedge (x \neq fc(i, m, R)\} \tag{6}$$

4 Defining a Secure PKI Scheme

In this section, we first define a PKI scheme, and then define its security requirements.

4.1 PKI: Entities and Certificates

PKI schemes are protocols for a set N of *authorities*, such as *certificate authorities (CAs)*. For simplicity and generality, our definitions do not consider different types of authorities; however, specific PKI schemes may define different types of authorities, for example, we later present the CT_{comp} scheme, where some authorities in N act as loggers, monitors or CAs.

PKI schemes have two types of users (clients): *subjects*, which typically use a CA to obtain and manage their public key certificate, and *relying parties*, which use the certificates of the subjects, in order to determine if they want to communicate with the subject (using the certified public key). Note, however, that we do not explicitly model clients as entities in the execution; instead, the *adversary* $\mathcal{A}$ 'acts for' the clients, i.e., sets the inputs to the authorities, including all client requests, and receives the outputs generated by the authorities, including all responses, as per the execution process (Algorithm 1). Authorities are responsible for the entire certificate life cycle, where the main events of issuing, upgrading and revoking certificates are driven by the clients' requests (as generated by the adversary).

A *certificate authority (CA)*, also referred to as an *issuer*, is an authority (in N) that issues *certificates* to subjects, where a certificate is a verifiable association of an identifier *id* (for a subject), with public information *pub* related to the subject; in a *public key certificate*, the public information includes a subject's public key.

We allow certificates to contain a number of *attributes*, from a set AttrSet. Attributes define properties of the subject and/or of the certificate, typically, validated by one of the authorities $\iota \in \mathsf{N}$. Different PKI schemes may use different sets AttrSet of attributes. For example, the 'classical' X.509 PKI, only uses Accountability (ACC) and Revocation accountability (ReACC), both signed, typically, by the *issuing CA* (aka the *issuer*). However, more advanced PKI schemes, e.g., Certificate Transparency, require additional attributes, often signed by other entities, e.g., a *logger*; see details when we discuss CT_{comp}.

We define the ACC,ΔTRA,ΔEQ-D,EQ-P,ReACC (REV),NReACC (NREV) and ΔReTRA attributes, each implying a corresponding security property of the certificate: accountability, Δ-transparency, equivocation detection, equivocation prevention, revocation accountability, non-revocation accountability, and Δ-revocation transparency requirements, respectively. We present game-based definition of these security requirements in §4.4.

Definition 3 (Certificate). *A* certificate *is a tuple:* $\psi = (id, pub, sd, ed, \rho)$*:*

- $\psi.id$*: identifier of the entity for which* ψ *was issued.*
- $\psi.pub$*: public information associated with* $\psi.id$*. The 'classical' public information is a cryptographic public key, but our definitions apply as well to any other public information.*
- $\psi.sd$ *and* $\psi.ed$*:* start *and* end *of the certificate validity period, respectively.*
- $\psi.\rho$*: certificate's signatures and attributes :*
  - $\psi.\rho[attr].\sigma$*: a attestation that* ψ *has an attribute* $attr$*.*
  - $\psi.\rho[attr].\iota$*: identity of the authority who attests for* $attr$*.*
  - $\psi.\rho[attr].clk$*: the local clock value, when* $\psi.\rho[attr].\iota$ *has attested for* $attr$*.*

To *issue* a certificate, we invoke the $\mathcal{P}$.Issue operation. Intuitively, this represent handling of a certificate request coming from some client, who provides the identifier *id* and the public key (or information) *pub*; formally, however, it is the *adversary* who invokes this operation, since the adversary is modeling an arbitrary sequence of client requests, as well as the arbitrary behavior of the environment.

Certificates are issued by some *issuer* (CA), $\iota \in \mathsf{N}$, and the output normally has (only) the ACC attribute, attesting to the fact that ι has issued the certificate, i.e., is accountable for it. Intuitively, ι is attesting that it has performed the required verification of the identity of the requesting client. However, this verification process is not part of the properties we model.

Once a certificate is issued, it can be then *upgraded* using the $\mathcal{P}$.Upgrade algorithm, 'adding' to the certificate one or more *attributes* from AttrSet. Formally, of course, these are two different certificates, since the output certificate contains this additional attribute; but they have the same '*Core*', i.e., the same 'basic data' (identifier, public key, and validity dates). This is captured by the *Core* function, defined as follows.

Definition 4 (The *Core* function). *Given a certificate* $\psi = (id, pub, sd, ed, \rho)$*, its Core is defined as* $Core(\psi) = (\psi.id, \psi.pub, \psi.sd, \psi.ed)$*, i.e., the entire certificate except for its attributes* $\psi.\rho$*. For convenience, we define* $Core(\perp) = \perp$*.*

Pending Certificates. The execution model required every protocol operation to be instantaneous, i.e., return immediate response. However, the $\mathcal{P}$.Issue, $\mathcal{P}$.Revoke and $\mathcal{P}$.Upgrade algorithms may not be able to *immediately* return the requested certificate, since, in some PKI systems, they may need to first interact with other authorities, e.g., to avoid equivocation (when required). Such non-immediate response is supported by *pending* certificates. The pending certificate ψ_p is a *commitment* to produce the requested certificate (or failure indication), within Δ time; Δ is typically known from the given attribute or request. Namely, the response is guaranteed, once the $\mathcal{P}$.Upgrade algorithm is invoked with the pending certificate ψ_p, Δ time or more after the pending certificate was sent. The final output is either the expected *non-pending* upgraded certificate, or $\perp$, i.e., the request was declined. We emphasize that whether ψ_p was generated by the $\mathcal{P}$.Issue, $\mathcal{P}$.Revoke, or the $\mathcal{P}$.Upgrade algorithm, if a pending certificate is received, then the final response can only be obtained using the $\mathcal{P}$.Upgrade algorithm.

Revocation. Revocation is a major aspect of PKI schemes, addressed already in X.509. X.509 defines one revocation mechanism, the *certificate revocation lists (CRLs)*; however, in practice, most deployments of X.509 follow the *OCSP* specifications [57]. Both CRLs and OCSP are non-trivial, with multiple variants and extensions, beyond our scope; we focus on the *functionality* which is required to properly support revocation. Notice that most post-X.509 PKIs do not explicitly address revocation, implicitly adopting the X.509 mechanisms (CRLs or OCSP).

With both CRLs and OCSP, revocation of certificates is handled by separate signed objects - the CRLs and the OCSP responses. Our definitions of PKI schemes and their security requirements are applicable to both CRLs and OCSP responses, however, without *explicitly* referring to any specific additional data structure (beyond the certificates themselves). Instead, we view the revocation process as resulting in a certificate with the same *Core*, but with revocation-related attributes.

Specifically, we consider *three revocation-related attributes*: Revoked REV attribute: attests that the certificate was revoked by a given authority, at a given local-clock value. Revocation is invoked by the $\mathcal{P}$.Revoke operation of the PKI scheme. A revocation attribute (REV) is useful, in particular, to allow subjects that revoked their certificate, to confirm that their request was honored. Signed OCSP responses provide an existing mechanism for REV; CRLs do not support REV. Non-revoked NREV attribute attests that the certificate was valid (*not* revoked), at a given local-clock value. Such attestations allow a relying party to justify reliance on the certificate, more precisely, on the mapping of public information/key to the identifier, e.g., for accepting a signed document as evidence of it being approved by the subject of the certificate. Both CRLs and OCSP provide NReACC. Δ-revocation transparency ΔReTRA: attests that the revocation shall be made available, in particular, to any 'monitoring' authorities, by Δ time units from the specified time (on the local clock). Monitoring authorities may use OCSP queries learn of revocations in timely manner, however, that requires a query for each of these certificates, which would not be viable.

To find out if a given certificate is revoked, invoke the $\mathcal{P}$.IsRevoked operation at the issuer of the certificate. If ψ was revoked, then $\mathcal{P}$.IsRevoked returns its revoked version ψ_r, with the REV attribute, and with the same *Core* (*Core*$(\psi_r) =$ *Core*(ψ)). If ψ was not revoked so far, then $\mathcal{P}$.IsRevoked also returns a certificate with the same *Core*, but with the NREV attribute.

Checking certificate validity: beyond checking for revocation by invoking $\mathcal{P}$.IsRevoked, PKI schemes also provide two functions, to check certificate status: the *stateless* $\mathcal{P}$.WasValid function, and the *stateful* $\mathcal{P}$.Audit function. However, the two functions differ in their goals, interfaces and usage. Details follow.

4.2 PKI Algorithms

We now define a PKI scheme $\mathcal{P}$, which supports the following set of operations:

$$\mathcal{P} = (\mathsf{Init}, \mathsf{Issue}, \mathsf{Upgrade}, \mathsf{Revoke}, \mathsf{IsRevoked}, \mathsf{Audit}, \mathsf{WasValid}, \mathsf{V_p})$$

Two of these operations, Init and V_p, are generic, i.e., relevant for many protocols and tasks, and therefore discussed already in §3.1. We next briefly discuss the other operations and their inputs[4].

To avoid clutter, we use the simplified notations defined in §3.1, i.e., explicitly write only the input parameters of each operation, although the implementation will, obviously, also refer to the clock and the state. (Note that the WasValid operation is *stateless*, hence, its implementation cannot refer to the clock or state.)

- Issue$(id, pub, sd, ed) \rightarrow \psi/\psi_p/\perp$: The algorithm takes as input an identity id, public information pub, start date sd and end date ed, and outputs a certificate ψ for (id, pub) that is valid from sd till ed. The algorithm may also output a pending certificate ψ_p, if it cannot immediately issue the certificate, e.g., if it needs to check with other authorities. If the operation fails, e.g., due to discovery of conflicting certificate, then the algorithm returns $\perp$.
- Upgrade$(\psi, attr) \rightarrow \psi'/\psi_p/\perp$: The algorithm takes as input a certificate ψ and an attribute $attr$. If the upgrade request is valid, the algorithm outputs an upgraded certificate ψ', with same core, and with the $attr$ attribute. The algorithm may also output a pending certificate ψ_p, if it requires time for interactions with other entities. If the upgrade fails, the algorithm returns $\perp$. If the input ψ is (already) a pending certificate, and the time specified was reached, then the algorithm does not return a pending certificate; it either returns the upgraded certificate or failure ($\perp$).
- Revoke$(\psi) \rightarrow \psi_r/\psi_p/\perp$: The algorithm takes as input a certificate ψ, and outputs a revoked certificate ψ_r, a pending-revoked certificate ψ_p, or failure indicator $\perp$.

[4] Implementations may allow additional optional inputs, in which case, our requirements should be interpreted as holding for any values of these additional inputs. The two PKI schemes we study do not use such optional inputs.

$\mathsf{IsRevoked}(\psi) \rightarrow \psi'/\psi_r/\perp$: The algorithm takes as input a certificate ψ. If ψ is known to be a valid non-revoked certificate, the algorithm outputs ψ', which is identical to ψ along with a proof of non-revocation until the current local time using the NREV attribute. If ψ was already revoked, the algorithm returns the revoked certificate ψ_r. In any other case, the algorithm returns $\perp$.

$\mathsf{Audit}(id, attr) \rightarrow \psi/\perp$ or $\mathsf{Audit}(\psi, attr) \rightarrow \top/\text{IA}/\zeta/\perp$: When Audit is invoked with an identifier id and an attribute $attr \in \mathsf{AttrSet}$, it either returns a valid set of certificates Ψ issued for id, correctly endorsed for $attr$, and known to the entity Audit is invoked on, or $\perp$ otherwise (no such certificates). When Audit is invoked with a certificate ψ and an attribute $attr \in \mathsf{AttrSet}$, it outputs $\top$ if the current state indicates that ψ is valid, and, in particular, that $attr$ has been legitimately included. In contrast, if the current state indicates $attr$ should *not* have been included in ψ, then Audit outputs an Indicator of Accusation (IA) or a proof of misbehavior ζ. Otherwise, e.g., if the current state does not provide the necessary information, then the algorithm returns $\perp$. For example, in CT_{comp}, we use Audit to validate the ΔTRA attribute; if a certificate has the ΔTRA attribute, but is not 'known' to a monitoring entity, then invoking $\mathcal{P}$.Audit in that entity will result in either proof of misbehavior or Indicator of Accusation.

$\mathsf{WasValid}(\psi, pk, attr\ [, tms]) \rightarrow \top/Pending/\perp$: This is a *stateless* function, that takes as input a certificate ψ, a public key pk, an attribute $attr \in \mathsf{AttrSet}$, and, optionally, timestamp tms. If ψ is a valid certificate, with attribute $attr$ attested to using public key pk, then the algorithm outputs $\top$. If ψ has the $attr$ attribute but in a pending state, the algorithm outputs $Pending$. If tms is used, then the output is with respect to time tms. This also applies to checking if the certificate is expired (that is, tms is outside of the sd and ed dates). In any other case, the algorithm outputs $\perp$.

4.3 PKI Correctness Requirements

Let $(\xi, \mathbf{Exp}^{\xi}_{\mathcal{P}})$ be a requirement; recall (Definition 2) that protocol $\mathcal{P}$ *ensures* requirement $(\xi, \mathbf{Exp}^{\xi}_{\mathcal{P}})$ under model $\mathcal{M}$, if for every PPT adversary $\mathcal{A}$ that satisfies $\mathcal{M}$, and every set of entities N, there is negligible probability that a random execution of $\mathcal{A}$ with $\mathcal{P}$, will not satisfy ξ.

We present the $(\mathsf{Correct}\text{-}\alpha, \mathbf{Exp}^{\mathsf{Correct}\text{-}\alpha}_{\mathcal{P}})$ requirements, confirming that the α operation of $\mathcal{P}$, for $\alpha \in \{\mathsf{Issue}, \mathsf{Revoke}, \mathsf{Upgrade}, \mathsf{IsRevoked}\}$, either return a valid certificate with the expected attributes (depending on α), or failure indication ($\perp$). Recall that for protocol $\mathcal{P}$, *validity* of a certificate is defined by the (stateless) $\mathcal{P}$.WasValid function; hence, the experiment requires the adversary to generate an execution *ending* in an $\alpha \in \{\mathsf{Issue}, \mathsf{Revoke}, \mathsf{Upgrade}, \mathsf{IsRevoked}\}$ operation with incorrect results, as defined by $\mathcal{P}$.WasValid.

Requirement 5. *Correctness requirements. Let* $\alpha \in \{$*Issue, Revoke, Upgrade, IsRevoked*$\}$. *The* $(\textit{Correct-}\alpha, \mathbf{Exp}_{\mathcal{P}}^{\textit{Correct-}\alpha})$ *requirement is defined by the following experiments:*

$$\mathbf{Exp}_{\mathcal{P}}^{\textit{Correct-Issue}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \left\{ \begin{array}{l} R.i[t] \notin \mathsf{N_F} \ \wedge \ V_{ka}(R.i[t], out_{\mathcal{A}}, R) \ \wedge \\ \wedge \ R.\alpha[t] = \textit{'Issue'} \ \wedge \ R.inp[t] = (ip, pub, sd, ed) \ \wedge \ R.out[t] \neq \perp \ \wedge \\ \wedge \ \left(\begin{array}{c} \mathcal{P}.\mathsf{WasValid}(R.out[t], out_{\mathcal{A}}, \mathsf{ACC}) \perp \vee \\ \vee R.out[t].\rho[\mathsf{ACC}].\iota \neq R.i[t] \ \vee \\ \vee \mathit{Core}(R.out[t]) \neq \mathit{Core}(R.inp[t]) \end{array} \right) \end{array} \right\}$$

$$\mathbf{Exp}_{\mathcal{P}}^{\textit{Correct-Revoke}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \left\{ \begin{array}{l} R.i[t] \notin \mathsf{N_F} \ \wedge \ V_{ka}(R.i[t], out_{\mathcal{A}}, R) \ \wedge \\ \wedge \ R.\alpha[t] = \textit{'Revoke'} \ \wedge \ R.out[t] \neq \perp \ \wedge \\ \wedge \ \left(\begin{array}{c} \mathcal{P}.\mathsf{WasValid}(R.out[t], out_{\mathcal{A}}, \mathsf{REV}) \perp \vee \\ \vee R.out[t].\rho[\mathsf{REV}].\iota \neq R.i[t] \ \vee \\ \vee \mathit{Core}(R.out[t]) \neq \mathit{Core}(R.inp[t]) \end{array} \right) \end{array} \right\}$$

$$\mathbf{Exp}_{\mathcal{P}}^{\textit{Correct-Upgrade}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \left\{ \begin{array}{l} R.i[t] \notin \mathsf{N_F} \ \wedge \ V_{ka}(R.i[t], out_{\mathcal{A}}, R) \ \wedge \\ \wedge \ R.\alpha[t] = \textit{'Upgrade'} \ \wedge \ R.inp[t] = (\psi, attr) \ \wedge \ R.out[t] \neq \perp \ \wedge \\ \wedge \ \left(\begin{array}{c} \mathcal{P}.\mathsf{WasValid}(R.out[t], out_{\mathcal{A}}, attr) \perp \vee \\ \vee R.out[t].\rho[attr].\iota \neq R.i[t] \ \vee \\ \vee \mathit{Core}(R.out[t]) \neq \mathit{Core}(R.inp[t]) \end{array} \right) \end{array} \right\}$$

$$\mathbf{Exp}_{\mathcal{P}}^{\textit{Correct-IsRevoked}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \left\{ \begin{array}{l} R.i[t] \notin \mathsf{N_F} \ \wedge \ V_{ka}(R.i[t], out_{\mathcal{A}}, R) \ \wedge \\ \wedge \ R.\alpha[t] = \textit{'IsRevoked'} \ \wedge \ R.out[t] \neq \perp \ \wedge \\ \wedge \ \left(attr \in \left\{ \begin{array}{c} \mathsf{ReACC}, \\ \mathsf{NReACC} \end{array} \right\} \right) \left(\begin{array}{c} \mathcal{P}.\mathsf{WasValid}(R.out[t], out_{\mathcal{A}}, attr) \perp \vee \\ \vee R.out[t].\rho[attr].\iota \neq R.i[t] \ \vee \\ \vee \mathit{Core}(R.out[t]) \neq \mathit{Core}(R.inp[t]) \end{array} \right) \end{array} \right\}$$

4.4 PKI Security Requirements

Let $(\xi, \mathbf{Exp}_{\mathcal{P}}^{\xi})$ be a security requirement; recall (Definition 2) that protocol $\mathcal{P}$ *ensures* security requirement $(\xi, \mathbf{Exp}_{\mathcal{P}}^{\xi})$ under model $\mathcal{M}$, if for every PPT adversary $\mathcal{A}$ that satisfies $\mathcal{M}$, and every set of entities N, there is negligible probability that a random execution of $\mathcal{A}$ with $\mathcal{P}$, will not satisfy ξ. Therefore, to define the requirements, it only remains to specify their names and experiments.

We now present a description and an experiment for the *accountability* property.

Requirement 6 (Accountability (ACC))**.** *An adversary* $\mathcal{A}$ *wins in the* accountability *experiment* $\mathbf{Exp}_{\mathcal{P}}^{\mathsf{ACC}}$*, if it produces an accountable certificate* ψ *which is valid, yet the specified issuing authority* $\psi.\rho[\mathsf{ACC}].\iota$ *did* **not** *issue* ψ*. See Algorithm 2. See Algorithm 2.*

Algorithm 2 $\mathbf{Exp}^{\mathsf{ACC}}_{\mathcal{P}}$

$\underline{\mathbf{Exp}^{\mathsf{ACC}}_{\mathcal{P}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}})}$:

1: $(\mathsf{ACC}, \psi, \iota, pk) \leftarrow out_{\mathcal{A}}$

2: **return** :

3: // pk is a valid key
$V_{ka}(\iota, pk, R) \wedge$

4: // ψ is a valid accountable certificate
$\mathcal{P}.\mathsf{WasValid}(\psi, pk, \mathsf{ACC}) \wedge$

5: // ψ was issued by honest authority ι
$\iota = \psi.\rho[\mathsf{ACC}].\iota \ \wedge \iota \in \mathsf{N} - \mathsf{N_F} \wedge$

// However, ι did not issue ψ

6: $\nexists \ \hat{t} \ s.t. \ R.i[\hat{t}] = \iota \ \wedge \alpha[\hat{t}] = \text{`}\mathsf{Issue}\text{'} \wedge R.inp[\hat{t}] = (\psi.id, \psi.pub, \psi.sd, \psi.ed)$

The $\mathbf{Exp}^{\mathsf{ACC}}_{\mathcal{P}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}})$ takes as input a security parameter 1^{κ}, a set of entities N and the output of the **Exec** algorithm, i.e., the subset of faulty entities $\mathsf{N_F}$, transcript R of the execution process, index of the last round t of the execution, and the transcript of the adversary's choices $out_{\mathcal{A}}$. In line 1, the algorithm extracts from $out_{\mathcal{A}}$ the certificate returned by the adversary (ψ), the selected honest authority (ι), and ι's public key (pk). In line 3, we validate the key outputted by the adversary using the Key Attribution Predicate (V_{ka}). The adversary *wins* the game if ψ is a valid accountable certificate issued by the honest authority ι (lines 4-5), yet ι was never instructed to execute the $\mathcal{P}.\mathsf{Issue}$ algorithm along with the correct inputs $\psi.id$, $\psi.pub$, $\psi.sd$, $\psi.ed$ (line 6).

Requirement 7 (Revocation accountability (ReACC))**.** *An adversary $\mathcal{A}$ wins in the* revocation accountability *experiment* $\mathbf{Exp}^{\mathsf{ReACC}}_{\mathcal{P}}$ *if it produces a valid revoked certificate ψ_r issued by an honest authority $\psi_r.\rho[\mathsf{REV}].\iota$, where $\psi_r.\rho[\mathsf{REV}].\iota$ did* not *revoke ψ_r. See Algorithm 3.*

Algorithm 3 $\mathbf{Exp}^{\mathsf{ReACC}}_{\mathcal{P}}$

$\underline{\mathbf{Exp}^{\mathsf{ReACC}}_{\mathcal{P}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}})}$:

1: $(\mathsf{ReACC}, \psi_r, \iota, pk) \leftarrow out_{\mathcal{A}}$

2: **return** :

3: // pk is a valid key
$V_{ka}(\iota, pk, R) \wedge$

4: // ψ_r is a valid revoked certificate
$\mathcal{P}.\mathsf{WasValid}(\psi_r, pk, \mathsf{REV}) \wedge$

5: // ψ_r was revoked by an honest authority
$\iota = \psi_r.\rho[\mathsf{REV}].\iota \ \wedge \ \iota \in \mathsf{N} - \mathsf{N_F} \wedge$

// However, ι did not revoke ψ_r

6: $\nexists \ \hat{t}, \psi \ s.t. \ R.i[\hat{t}] = \iota \ \wedge \alpha[\hat{t}] = \text{`}\mathsf{Revoke}\text{'} \wedge R.inp[\hat{t}] = (\psi) \ \wedge Core(\psi) = Core(\psi_r)$

In line 1 of $\mathbf{Exp}^{\mathsf{ReACC}}_{\mathcal{P}}$, the algorithm extracts from $out_{\mathcal{A}}$ the certificate returned by the adversary (ψ_r), the selected honest authority (ι), and the its public

key (pk). The adversary wins the game if ψ_r is a valid revoked certificate revoked by the honest authority ι (lines 4-5), yet ι was never instructed to revoke such a certificate, i.e., to execute the $\mathcal{P}.\mathsf{Revoke}$ algorithm, given as input a certificate which has the same core as ψ_r (line 6).

Requirement 8 (Non-revocation accountability (NReACC))**.** *An adversary $\mathcal{A}$ wins in the* non-revocation accountability *experiment $\mathbf{Exp}_{\mathcal{P}}^{\mathsf{NReACC}}$ if it produces a certificate ψ_r revoked by an honest authority $\psi_r.\rho[\mathsf{REV}].\iota$, and a certificate ψ with the same core as ψ_r, which has the non-revoked attribute attested by an honest authority $\psi.\rho[\mathsf{NREV}].\iota$, where the attestation was performed* after *ψ_r was revoked. See Algorithm 4.*

In line 1 of $\mathbf{Exp}_{\mathcal{P}}^{\mathsf{NReACC}}$, the algorithm extracts from $out_{\mathcal{A}}$ two certificates (ψ, ψ_r) and corresponding public keys (pk,pk_r) (line 1). The adversary wins the $\mathbf{Exp}_{\mathcal{P}}^{\mathsf{NReACC}}$ game if it produces a valid, revoked certificate ψ_r and a matching non-revoked certificate ψ (line 4) such that ψ was produced after ψ_r (line 6), and both were produced by by the same honest authority (line 5).

Algorithm 4 $\mathbf{Exp}_{\mathcal{P}}^{\mathsf{NReACC}}$

$\mathbf{Exp}_{\mathcal{P}}^{\mathsf{ReACC}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}})$:

1: $(\mathsf{NReACC}, \psi, \psi_r, pk, pk_r) \leftarrow out_{\mathcal{A}}$

2: **return** :

3: // pk and pk_r are valid keys
$V_{ka}(\psi_r.\rho[\mathsf{REV}].\iota, pk_r, R) \ \wedge V_{ka}(\psi.\rho[\mathsf{NREV}].\iota, pk, R) \ \wedge$

4: // ψ_r is a valid revoked certificate and ψ is a matching, valid non-revoked certificate
$\mathcal{P}.\mathsf{WasValid}(\psi_r, pk_r, \mathsf{REV}) \ \wedge \mathcal{P}.\mathsf{WasValid}(\psi, pk, \mathsf{NREV}) \ \wedge Core(\psi_r) = Core(\psi) \ \wedge$

5: // both REV and NREV were signed by honest authorities
$\{\psi.\rho[\mathsf{REV}].\iota, \psi.\rho[\mathsf{NREV}].\iota\} \subseteq \mathsf{N} - \mathsf{N_F}$

6: // ψ was produced after ψ_r
$\psi.\rho[\mathsf{NREV}].clk > \psi_r.\rho[\mathsf{REV}].clk$

Requirement 9 (Δ-Transparency ($\Delta\mathsf{TRA}$))**.** *An adversary $\mathcal{A}$ wins in the Δ-transparency experiment $\mathbf{Exp}_{\mathcal{P}}^{\Delta\mathsf{TRA}}$ if it produces a valid certificate ψ, which is transparent at time $\psi.\rho[\Delta\mathsf{TRA}].clk$, yet there is any pair of honest authorities ι, ι' who are not aware of ψ after time $\psi.\rho[\Delta\mathsf{TRA}].clk + \Delta$, and neither issued an Indicator of Accusation for the authority who endorsed the $\Delta\mathsf{TRA}$ attribute for ψ, or neither has a proof of misbehavior of that authority. See Algorithm 5.*

Algorithm 5 $\mathbf{Exp}_{\mathcal{P}}^{\Delta\mathsf{TRA}}$

$\underline{\mathbf{Exp}_{\mathcal{P}}^{\Delta\mathsf{TRA}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}):}$

1: $(\Delta\mathsf{TRA}, \psi, pk) \leftarrow out_{\mathcal{A}}$

2: $\iota = R.i[t], \iota' = R.i[t\text{-}1]$

3: **return** :

4: // pk is a valid public key
$V_{ka}(\psi.\rho[\Delta\mathsf{TRA}].\iota, pk, R) \wedge$

5: // ψ is a valid transparent certificate
$\mathcal{P}.\mathsf{WasValid}(\psi, pk, \Delta\mathsf{TRA}) \wedge$

6: // both ι and ι' are honest
$R.i[t\text{-}1] = R.i[t\text{-}2] \ \wedge \{\iota, \iota'\} \in \mathsf{N} - \mathsf{N_F} \wedge$

7: // check for $\Delta\mathsf{TRA}$ ψ that matches id
$R.\alpha[t\text{-}2] = \text{`}\mathsf{Audit}\text{'} \wedge$
$R.inp[t\text{-}2] = (\psi.id, \Delta\mathsf{TRA}) \wedge$

8: // check for any issues with $\Delta\mathsf{TRA}$ ψ
$(t' = t\text{-}1, t)\ R.\alpha[t'] = \text{`}\mathsf{Audit}\text{'} \wedge$
$R.inp[t'] = (\psi, \Delta\mathsf{TRA}) \wedge$

9: // time for ψ to be $\Delta\mathsf{TRA}$ has passed
$R.clk[t\text{-}2] \geq \psi.\rho[\Delta\mathsf{TRA}].clk + \Delta \wedge$

10: // ψ is not transparent
$\psi \notin R.out[t\text{-}2] \wedge$

11: // neither reported $\psi.\rho[\Delta\mathsf{TRA}].\iota$ as bad
or neither has a valid proof of
$\psi.\rho[\Delta\mathsf{TRA}].\iota$'s misbehavior
$(t' = t\text{-}1, t)$
$((\text{IA}, \psi.\rho[\Delta\mathsf{TRA}].\iota, x)|_{x \leq \psi.\rho[\Delta\mathsf{TRA}].clk + \Delta} \notin R.out[t']$
$\vee$

12: $\mathcal{P}.V_p(pk, R.out[t']) = \perp)$

The adversary wins the $\mathbf{Exp}_{\mathcal{P}}^{\Delta\mathsf{TRA}}$ the game if it can produce (line 1) a valid, transparent certificate ψ (line 5) such that there exist two honest authorities ι, ι' (line 6) who after the appropriate time for ψ to be $\Delta\mathsf{TRA}$ (line 9) are not aware of ψ (line 10) and they do not consider the authority who endorsed ψ for $\Delta\mathsf{TRA}$ faulty, i.e., issued an Indicator of Accusation at the appropriate time (line 11) or have a valid proof of its misbehavior validated using $\mathcal{P}.V_p$ (line 12). This requirement is validated by verifying that ι and ι' were instructed to invoke $\mathcal{P}.\mathsf{Audit}$ with the appropriate inputs (lines 7-8) and produces appropriate outputs as described above. This ensures that if $\mathcal{P}.\mathsf{Audit}$ is invoked on two different honest entities, the answers will be consistent, i.e., a certificate will not be transparent from one entity's point of view but not the other, and if the certificate is not transparent, the issuer will be accused of misbehavior (using IA) or there will be a valid proof of its misbehavior.

Requirement 10 (Revocation transparency ($\Delta\mathsf{ReTRA}$))**.** *An adversary $\mathcal{A}$ wins in the Δ-*revocation transparency *experiment $\mathbf{Exp}_{\mathcal{P}}^{\Delta\mathsf{ReTRA}}$ if it produces a valid certificate ψ, which is $\Delta\mathsf{ReTRA}$ at time $\psi.\rho[\Delta\mathsf{ReTRA}].clk$, and a matching, valid certificate ψ_r, which is revoked at time $\psi_r.\rho[\mathsf{REV}].clk$ by an honest authority, yet there is any pair of honest authorities ι, ι' who are not aware of*

ψ_r after time $\max(\psi_r.\rho[\mathsf{REV}].clk, \psi.\rho[\Delta\mathsf{ReTRA}].clk) + \Delta$*, and neither issued an Indicator of Accusation for the authority who endorsed the* $\Delta\mathsf{ReTRA}$ *attribute for* ψ*, or neither has a proof of misbehavior of that authority. See Algorithm 6.*

Algorithm 6 $\mathbf{Exp}_{\mathcal{P}}^{\Delta\mathsf{ReTRA}}$

$\underline{\mathbf{Exp}_{\mathcal{P}}^{\Delta\mathsf{ReTRA}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}})}$:

1: $(\Delta\mathsf{ReTRA}, \psi, \psi_r, pk, pk_r) \leftarrow out_{\mathcal{A}}$

2: $\iota = R.i[t], \iota' = R.i[t\text{-}1]$

3: **return** :

// pk and pk_r are valid keys

4: $V_{ka}(\psi.\rho[\Delta\mathsf{ReTRA}].\iota, pk, R) \wedge$

$V_{ka}(\psi_r.\rho[\mathsf{REV}].\iota, pk_r, R) \wedge$

// ψ is a valid $\Delta\mathsf{ReTRA}$ certificate

5: $\mathcal{P}.\mathsf{WasValid}(\psi, pk, \Delta\mathsf{ReTRA}) \wedge$

// ψ_r is a valid revoked certificate

6: $\mathcal{P}.\mathsf{WasValid}(\psi_r, pk_r, \mathsf{REV}) \wedge$

// ψ and ψ_r have the same Core

7: $Core(\psi) = Core(\psi_r) \wedge$

// ψ_r revoked by an honest entity

8: $\psi.\rho[\mathsf{REV}].\iota \in \mathsf{N} - \mathsf{N_F} \wedge$

// both ι and ι' are honest

9: $R.i[t\text{-}1] = R.i[t\text{-}2] \ \wedge \{\iota, \iota'\} \in \mathsf{N} - \mathsf{N_F} \wedge$

// check for $\Delta\mathsf{ReTRA}$ ψ that matches id

10: $R.\alpha[t\text{-}2] =$ 'Audit' $\wedge$

$R.inp[t\text{-}2] = (\psi.id, \Delta\mathsf{ReTRA}) \wedge$

// check for any issues with $\Delta\mathsf{ReTRA}$ ψ

11: $(t' = t\text{-}1, t)\ R.\alpha[t'] =$ 'Audit' $\wedge$

$R.inp[t'] = (\psi, \Delta\mathsf{ReTRA}) \wedge$

// time to know REV status has passed

12: $R.clk[t\text{-}2] \geq \max(\psi_r.\rho[\mathsf{REV}].clk,$

$\psi.\rho[\Delta\mathsf{ReTRA}].clk) + \Delta \wedge$

// ψ is not $\Delta\mathsf{ReTRA}$

13: $\psi \notin R.out[t\text{-}2] \wedge$

// neither reported $\psi.\rho[\Delta\mathsf{ReTRA}].\iota$
as bad or neither has a valid proof of
$\psi.\rho[\Delta\mathsf{ReTRA}].\iota$'s misbehavior

14: $(t' = t\text{-}1, t)$

$((\mathrm{IA}, \psi.\rho[\Delta\mathsf{ReTRA}].\iota, x)|_{x \leq \psi.\rho[\Delta\mathsf{ReTRA}].clk + \Delta} \notin R.out[t']$

$\vee$

15: $\mathcal{P}.V_p(pk, R.out[t']) = \perp)$

The adversary wins the $\mathbf{Exp}_{\mathcal{P}}^{\Delta\mathsf{ReTRA}}$ game if the following requirements are met. First, ψ must be a valid $\Delta\mathsf{ReTRA}$ certificate (line 5), ψ_r must be a matching, valid revoked certificate (lines 6-7) produced by an honest authority (line 8). Second, there exist two honest authorities ι, ι' (line 9) who after the appropriate time for the revocation status to be known, i.e., at most Δ after ψ_r was revoked or ψ became $\Delta\mathsf{ReTRA}$ whichever is later (line 12) are not aware of ψ_r (line 13)

and they do not consider the authority who endorsed ψ for ΔReTRA faulty (lines 14-15), validated as in the ΔTRA experiment described above.

Requirement 11 (Equivocation detection (ΔEQ-D))**.** *An adversary $\mathcal{A}$ wins in the Δ-*equivocation detection *experiment* $\mathbf{Exp}_{\mathcal{P}}^{\Delta\text{EQ-D}}$ *if it produces two valid, non-revoked* ΔEQ-D *certificates ψ, ψ' for the same identifier ($\psi.id = \psi'.id$) and for overlapping validity periods such that each certificate has different public information ($\psi.pub \neq \psi'.pub$), yet none of the entities in* N *was able to detect the equivocation before the Δ time of the* ΔEQ-D *property has passed. See Algorithm 7.*

Algorithm 7 $\mathbf{Exp}_{\mathcal{P}}^{\Delta\text{EQ-D}}$

$\underline{\mathbf{Exp}_{\mathcal{P}}^{\Delta\text{EQ-D}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}})}$:

1: $(\Delta\text{EQ-D}, \psi, \psi\text{'}, pk, pk', pk_{\iota}) \leftarrow out_{\mathcal{A}}$

2: $\iota = R.i[t]$

3: **return** :

// pk, pk' and pk_{ι} are valid keys

4: $V_{ka}(\psi.\rho[\Delta\text{EQ-D}].\iota, pk, R) \wedge$

$V_{ka}(\psi\text{'}.\rho[\Delta\text{EQ-D}].\iota, pk', R) \wedge$

$V_{ka}(\iota, pk_{\iota}, R) \wedge$

// ι is honest

5: $t' = (t\text{-}4, t\text{-}3, t\text{-}2, t\text{-}1)\ R.i[t'] = R.i[t] \wedge$

$\iota \in \mathsf{N} - \mathsf{N_F} \wedge$

// ψ and ψ' are unequivocal

6: $\mathcal{P}.\mathsf{WasValid}(\psi, pk, \Delta\text{EQ-D}) \wedge$

7: $\mathcal{P}.\mathsf{WasValid}(\psi\text{'}, pk', \Delta\text{EQ-D}) \wedge$

// ψ and ψ' have the same identifier

8: $\psi.id = \psi\text{'}.id \wedge$

// ψ and ψ' have overlapping validity
// periods, yet different $PubInfo$

9: $\psi.sd < \psi\text{'}.sd < \psi.ed \ \wedge \ \psi.pub \neq \psi\text{'}.pub \wedge$

// both certs are non-revoked

10: $(t' = t\text{-}3, t\text{-}4)\ R.\alpha[t'] = \text{`}\mathsf{IsRevoked}\text{'} \wedge$

11: $R.inp[t\text{-}3] = (\psi, \mathsf{NREV}) \wedge$

$R.inp[t\text{-}4] = (\psi\text{'}, \mathsf{NREV}) \wedge$

12: $(t' = t\text{-}3, t\text{-}4)$

$(\mathcal{P}.\mathsf{WasValid}(out[t'], pk_{\iota}, \mathsf{NREV})$

// The Δ time for detection has passed

13: $R.\tau[t] > \Delta + max(\psi.\rho[\Delta\text{EQ-D}].clk,$

$\psi\text{'}.\rho[\Delta\text{EQ-D}].clk)$

// No one detected the equivocation

14: $(\nexists t' \leq t)\ ((\psi, \psi\text{'}, \text{`Equivocation'}) \leftarrow out[t'])$

$\wedge$

// check for certs that match id

15: $R.\alpha[t\text{-}2] = \text{`}\mathsf{Audit}\text{'} \wedge$

$R.inp[t\text{-}2] = (\psi.id, \Delta\text{EQ-D}) \wedge$

// audit for ψ and ψ'

16: $R.\alpha[t\text{-}1] = \text{`}\mathsf{Audit}\text{'} \wedge$

$R.inp[t\text{-}1] = (\psi, \Delta\text{EQ-D}) \wedge$

17: $R.\alpha[t\text{-}1] = \text{`}\mathsf{Audit}\text{'} \wedge$

$R.inp[t\text{-}1] = (\psi, \Delta\text{EQ-D}) \wedge$

// either neither cert is known, or one is known, and auditing again with the other one does not cause detection of attack

18: $(\{\psi, \psi\text{'}\} \cap R.out[t\text{-}2] \ \vee \ R.out[t\text{-}1] = \top)$

The adversary wins the $\mathbf{Exp}_{\mathcal{P}}^{\Delta\text{EQ-D}}$ game if the following requirements are met. First, ψ and ψ' must be valid, non-equivocal certificates (line 6) for the same identifier (line 8) with overlapping validity periods but with different public info (line 9) such that ψ was never revoked (lines 10-12). Second, the time for

detection (line 13) has passed and no one detected the equivocation (line 14) and an honest entity ι either did not issue an Indicator of Accusation or has a proof of misbehavior (lines 15-17).

We note that detecting equivocating certificate does not automatically indicate a need to accuse another entity of misbehavior and does not, on its own, constitute a proof of misbehavior since the interpretation of equivocation is application dependent.

Algorithm 8 $\mathbf{Exp}_{\mathcal{P}}^{\mathsf{EQ\text{-}P}}$

$\underline{\mathbf{Exp}_{\mathcal{P}}^{\mathsf{EQ\text{-}P}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}})}$:

1: $(\mathsf{EQ\text{-}P}, \psi, \psi', pk, pk', pk_{\iota}) \leftarrow out_{\mathcal{A}}$

2: $\iota = R.i[t]$

3: **return** :

// pk, pk' and pk_{ι} are valid keys

4: $V_{ka}(\psi.\rho[\mathsf{EQ\text{-}P}].\iota, pk, R) \wedge$

$V_{ka}(\psi'.\rho[\mathsf{EQ\text{-}P}].\iota, pk', R) \wedge$

$V_{ka}(\iota, pk_{\iota}, R) \wedge$

// ι is honest

5: $i[t] = i[t\text{-}1] \ \wedge \iota \in \mathsf{N} - \mathsf{N_F} \wedge$

// ψ and ψ' are non-equivocal

6: $\mathcal{P}.\mathsf{WasValid}(\psi, pk, \mathsf{EQ\text{-}P}) \wedge$

$\mathcal{P}.\mathsf{WasValid}(\psi', pk', \mathsf{EQ\text{-}P}) \wedge$

// ψ and ψ' have the same identifier

7: $\psi.id = \psi'.id \wedge$

// ψ and ψ' have overlapping validity periods, yet different $PubInfo$

8: $\psi.sd < \psi'.sd < \psi.ed \ \wedge \ \psi.pub \neq \psi'.pub \wedge$

// both certs are non-revoked

9: $(t' = t\text{-}1, t)\, R.\alpha[t'] = \text{`}\mathsf{IsRevoked}\text{'} \wedge$

10: $R.inp[t\text{-}1] = (\psi, \mathsf{NREV}) \wedge$

$R.inp[t] = (\psi', \mathsf{NREV}) \wedge$

11: $(t' = t\text{-}1, t)\, \mathcal{P}.\mathsf{WasValid}(out[t'], pk_{\iota}, \mathsf{NREV})$

Requirement 12 (Equivocation prevention (EQ-P))**.** *An adversary $\mathcal{A}$ wins in the* equivocation prevention *experiment* $\mathbf{Exp}_{\mathcal{P}}^{\mathsf{EQ\text{-}P}}$ *if it produces two valid, non-revoked certificates ψ, ψ' for the same identifier ($\psi.id = \psi'.id$) and for overlapping validity periods, where each certificate has different public information ($\psi.pub \neq \psi'.pub$). See Algorithm 8.*

The adversary wins the game $\mathbf{Exp}_{\mathcal{P}}^{\mathsf{EQ\text{-}P}}$ if there exist two valid, unequivocal certificates ψ and ψ' (line 6) for the same identifier (line 7), with overlapping validity periods and issued for the same public information (line 8) such that neither certificate is revoked as attested by the same honest entity (lines 9-11).

5 Two Provably-Secure PKI Schemes

In this section, we describe the implementation and analysis of two PKI schemes, proving that each of them achieves a (subset) of our security requirements. This analysis allowed us to fine-tune the requirements, and ensure that they are feasible. The first scheme is a *proof-of-concept* PKI scheme called PoC-PKI, a rather simple scheme that achieves **all** of the requirements. The second scheme is *Certificate Transparency (CT)*, the only widely-deployed 'post-X.509' PKI scheme.

For lack of space, we only provide in this section very brief description of the schemes, their models and the highlights from their analysis. For the complete designs and analyses, see Appendices A-E.

5.1 PoC-PKI: a Proof-of-Concept PKI scheme

PoC-PKI is a conceptually-simple PKI scheme, which ensures advanced properties such as transparency and prevention of equivocation. To ensure such properties, PoC-PKI uses threshold signature scheme to simulate a *collectively* trusted entity that manages the certificates. Similar ideas were proposed in CoSi [60] and ARPKI [4]. In PoC-PKI, an accountable certificate ψ can be upgraded to be $\Delta_{\text{PoC-PKI}}$-transparent by obtaining a commitment from *one* of the authorities, where that authority commits to inform the rest of the authorities about ψ in at most $\Delta_{\text{PoC-PKI}}$ time. Similarly, an accountable (or $\Delta_{\text{PoC-PKI}}$-transparent) certificate can be upgraded to a non-equivocal one; however, this is not immediate. PoC-PKI first returns a *pending* non-equivocation certificate, and propagates the non-equivocation request to the rest of the authorities. If they approve, they contribute their partial signature, attesting of their approval, and more importantly, they will not approve any other certificate as non-equivocal if it was issued for the same identifier for an overlapping validity period.

The PoC-PKI scheme uses a secure signature scheme $\mathcal{S}$, a secure encryption scheme $\mathcal{E}$ and a secure and robust threshold signature scheme $\mathcal{TS}$. Hence, the scheme's fully-qualified name is PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$; however, for brevity, we often use the 'shorthand' notation PoC-PKI.For the full description of PoC-PKI, see Appendix B.

5.1.1 The PoC-PKI Model Function $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$

We define the model of PoC-PKI in Eq. (1), as $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}(\xi) = \mathcal{M}^{\text{3-rounds}}_{\text{SecInit}}(\xi) \wedge \mathcal{M}^{|\mathsf{N_F}|\leq\lfloor(|\mathsf{N}|/3)\rfloor}(\xi) \wedge \mathcal{M}^{\text{COM}}_{\Delta_{com}}(\xi) \wedge \mathcal{M}^{\text{CLK}}_{\Delta_{clk}}(\xi)$, where, $\mathcal{M}^{\text{3-rounds}}_{\text{SecInit}}$ is the secure initialization model (§ 3.2.2), $\mathcal{M}^{|\mathsf{N_F}|\leq\lfloor(|\mathsf{N}|/3)\rfloor}$ is the fault model (§ 3.2.1), $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$ is the communication model (§ 3.2.3) and $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$ is the clock synchronization model as (§ 3.2.4).

We use $\Delta_{\text{PoC-PKI}} = \Delta_{com} + \Delta_{clk}$ to denote the delay ensured by PoC-PKI. The reason for this value is that when an authority upgrades a certificate with transparency or revocation transparency, it immediately broadcasts the upgraded certificate to the rest of the entities. Since the $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ model ensures that the adversary cannot prevent these messages from being delivered, it

takes at most Δ_{com} time for the messages to arrive to other entities. However, entities' clocks might not be completely synchronized, which can add up to Δ_{clk} additional delay.

5.1.2 Analysis: PoC-PKI provably achieves security requirements

PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ uses a signature scheme, encryption scheme and threshold signature scheme; however, as the following two Theorems show, PoC-PKI achieves many requirements, even assuming only the signature scheme is secure (existentially unforgeable). The first proof focuses on the 'generic' requirements (§4.4), and the second covers several PKI-specific requirements.

Theorem 1. *Let $\mathcal{S}$ be an existentially-unforgeable signature scheme. Then* PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves Verifiable Attribution of Statements (VAS) for $\mathcal{S}$, Non-frameability (NF), No False Accusations (NFA) and Use First-Accuse Time (UFAT) under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$.*

Theorem 2. *Let $\mathcal{S}$ be an existentially-unforgeable signature scheme. Then* PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves accountability, revocation accountability, non-revocation accountability, $\Delta_{\text{PoC-PKI}}$-transparency, and $\Delta_{\text{PoC-PKI}}$-revocation transparency under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$.*

Theorem 3. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves equivocation prevention under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, provided that $\mathcal{TS}$ is ($t \leq \mathcal{M}^{|\mathsf{N_F}| \leq \lfloor (n/3) \rfloor}$)-existentially unforgeable, and $\mathcal{E}$ is CPA-indistinguishable.*

For the proofs of theorems 1-3, refer to the analysis of PoC-PKI in Appendix C.

5.2 CT$_{comp}$: simplified, 'completed' Certificate Transparency PKI

CT is the most widely known post-X.509 PKI scheme, and the first to be deployed, as well as standardized [39]. It is also subject of significant criticism regarding its purported properties; all this makes analysis highly desirable. Unfortunately, the RFC6962 left crucial under-specified aspects, e.g., the gossip protocol, which is mentioned but not specified.

We analyzed CT$_{comp}$, which is our best effort to complete the missing specifications in RFC6962, in the simplest possible way, to create a well-defined protocol whose properties *can* be analyzed.

CT$_{comp}$ models three types of entities: certificate authorities, loggers and monitors. Loggers keep public logs of certificates issued by different CAs, and monitors validate that logs are published consistently. Note that clients and auditors are not modeled in CT$_{comp}$, since they are irrelevant with respect to the security properties. See Appendix D for details.

Loggers, in CT_{comp}, produce a *signed tree hash (STH)*, once every MMD (maximum merge delay) seconds. All loggers use the same value of MMD; this is consistent with the current deployment of CT. Each honest monitor maintains a full copy of each log it watches and after each MMD period, it fetches the new STH along with all the newly added certificates. Then, the monitor ensures that the new STH complies with the updated copy of the log it maintains. In addition, CT_{comp} performs naive gossip by exchanging all the new certificates and STHes between monitors every MMD.

5.2.1 The CT_{comp} Model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$

We define the model of CT_{comp} as:

$$\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}(\xi) = \mathcal{M}^{\text{2-rounds}}_{\text{SecInit}}(\xi) \wedge \mathcal{M}^{\text{COM}}_{\Delta_{com}}(\xi) \wedge \mathcal{M}^{\text{CLK}}_{\Delta_{clk}}(\xi) \wedge \mathcal{M}^{Auth}_{Mapping}(\xi)$$

where most of these model predicates were already defined earlier:

$\mathcal{M}^{\text{2-rounds}}_{\text{SecInit}}$ is the secure initialization model (§ 3.2.2), $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$ is the communication model (§ 3.2.3), and $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$ is the clock synchronization model (§ 3.2.4). The one new component is the $\mathcal{M}^{Auth}_{Mapping}$ model, which ensures consistency between the mapping of monitors to loggers, and the monitors it invokes in the transparency experiment. For details, see § D.2.

Let $\Delta_{CT_{comp}} \equiv 4\Delta_{com} + 2\Delta_{clk}$; this delay, $\Delta_{CT_{comp}}$, accounts for the time it takes to include the certificate in the log ($MMD \leq \Delta_{com}$), plus the time it takes for other entities to learn about new certificates (Δ_{com}), plus the time it takes to send a gossip message (Δ_{com}) and receive a gossip message (Δ_{com}). On top of that, because the clocks might be skewed by at most Δ_{clk}, we add one Δ_{clk} for when entities learn of new certificates and another Δ_{clk} for the gossip.

5.2.2 Analysis: CT_{comp} provably achieves most requirements

Our analysis proves that CT_{comp} achieves most of the properties of the PKI framework. Specifically, we prove that CT_{comp} achieves *Verifiable Attribution of Statements (VAS) for $\mathcal{S}$, Non-frameability (NF), No False Accusations (NFA), Use First-Accuse Time (UFAT)*, *accountability*, *revocation accountability*, *non-revocation accountability*, and *$\Delta_{CT_{comp}}$-transparency*. All of these are achieved under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, and assuming security of $\mathcal{S}$, $\mathcal{GMT}$, and $\mathcal{H}$, under their respective definitions. For lack of space, the relevant theorems and their proofs are in section E.

The theorems refers to $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$, which specifies the signature scheme $\mathcal{S}$, a generalized Merkle tree scheme $\mathcal{GMT}$, and CRHF $\mathcal{H}$ used by CT_{comp}; the proofs assume that these three primitives satify their respective security definitions. See §A for details.

6 Conclusions and Future Work

Public Key Infrastructure (PKI) is the basis for security of most applied cryptographic systems, and modern, post-X.509 PKI schemes have non-trivial design

and features. Hence, it is crucial to carefully define their security requirements, and prove these are satisfied. In this paper, we provide the first rigorous specifications for PKI schemes, and two (very different) provably-secure schemes: PoC-PKI, a 'proof-of-concept' scheme that meets all requirements, and CT_{comp}, a completion of the Certificate Transparency (CT) specifications, which meets most requirements. A significant challenge we faced, was the lack of appropriate framework for rigorous analysis of applied, stateful cryptographic protocols, with different communication, synchronization and adversary models. We addressed this challenge by presenting the *Model-Secure framework*, where security is defined with respect to a specific *model predicate* $\mathcal{M}$. The framework appears useful for other tasks; it allows comparison of protocols based on the requirements they satisfy, and the models they assume. Definitions of models and requirements may be reused across different types of protocols and schemes; we identified several 'generic' requirements, which appear to be applicable to many different tasks.

We see this work as only the stepping stone to further research, both of provably-secure PKI schemes, and of the Model-Secure framework. In particular, we expect that our PKI requirements need to be improved and extended, much like the similar research process for 'classical' cryptographic primitives. Indeed, while we did our best to define and present the best requirements, simplifying the experiments as much as we could, they still require considerable effort to fully understand. This shows the challenge and importance of properly analysing these complex schemes; at the same time, it surely shows the need for further efforts to refine and extend the requirements. Some extensions were discussed, informally, in previous works, most notably, notions of privacy, and the study of the requirements of Authenticated Key Exchange protocols, from the key certification mechanism, see [7]. Another significant challenge is the design of a practical PKI scheme that will meet all desired requirements, minimizing or avoiding simplifications.

Another important challenge is to extend the Model-Secure framework. In particular, extensions are necessary to allow specification of *privacy* requirements such as indistinguishability; this was not required for the PKI requirements we focused on, and therefore we did not yet pursue this challenge. Possibly a larger challenge is to support secure compositions, possibly following UC [11]. Specifically, it would be interesting to extend [13], which present a UC definition for a simplified basic PKI (a reduced functionality of X.509 PKI).

References

1. Namecoin, https://www.namecoin.org/
2. Asghari, H., Van Eeten, M., Arnbak, A., van Eijk, N.A.: Security Economics in the HTTPS Value Chain. In: Twelfth Workshop on the Economics of Information Security (WEIS 2013), Washington, DC (2013)
3. Axon, L., Goldsmith, M.: PB-PKI: A Privacy-aware Blockchain-based PKI. In: SECRYPT (2017)
4. Basin, D., Cremers, C., Kim, T.H.J., Perrig, A., Sasse, R., Szalachowski, P.: ARPKI: Attack Resilient Public-Key Infrastructure. In: Proceedings of the 2014

ACM SIGSAC Conference on Computer and Communications Security. pp. 382–393. ACM (2014)
5. Boldyreva, A., Fischlin, M., Palacio, A., Warinschi, B.: A Closer Look at PKI: Security and Efficiency. In: International Workshop on Public Key Cryptography. pp. 458–475. Springer (2007)
6. Boneh, D., Shoup, V.: A graduate course in applied cryptography. Draft 0.4 (2017)
7. Boyd, C., Cremers, C., Feltz, M., Paterson, K.G., Poettering, B., Stebila, D.: Asics: Authenticated key exchange security incorporating certification systems. International Journal of Information Security **16**(2), 151–171 (2017)
8. Braun, J.: Maintaining Security and Trust in Large Scale Public Key Infrastructures. Ph.D. thesis, Technische Universität (2015)
9. Braun, J., Kiefer, F., Hülsing, A.: Revocation & Non-Repudiation: When the first destroys the latter. In: European Public Key Infrastructure Workshop. pp. 31–46. Springer (2013)
10. Callegati, F., Cerroni, W., Ramilli, M.: Man-in-the-Middle Attack to the HTTPS Protocol. IEEE Security & Privacy **7**(1), 78–81 (2009)
11. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on. pp. 136–145. IEEE (2001), online at https://eprint.iacr.org/2000/067.pdf, last updated Dec. 2018.
12. Canetti, R., Krawczyk, H.: Security analysis of ike's signature-based key-exchange protocol. In: Yung, M. (ed.) Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2442, pp. 143–161. Springer (2002)
13. Canetti, R., Shahaf, D., Vald, M.: Universally Composable Authentication and Key-exchange with Global PKI. Cryptology ePrint Archive, Report 2014/432 (2014), https://eprint.iacr.org/2014/432
14. Canetti, R., Shahaf, D., Vald, M.: Universally Composable Authentication and Key-exchange with Global PKI. In: Public-Key Cryptography–PKC 2016. pp. 265–296. Springer (2016)
15. CCITT, B.B.: Recommendations X. 509 and ISO 9594-8. Information Processing Systems-OSI-The Directory Authentication Framework (Geneva: CCITT) (1988)
16. Chase, M., Meiklejohn, S.: Transparency Overlays and Applications. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 168–179. ACM (2016)
17. Comodo™: Incident Report. Published online, http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html (March 2011)
18. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Tech. rep. (2008)
19. Cremers, C.: Key exchange in ipsec revisited: Formal analysis of ikev1 and ikev2. In: European Symposium on Research in Computer Security. pp. 315–334. Springer (2011)
20. Damgård, I.B.: A Design Principle for Hash Functions. In: Conference on the Theory and Application of Cryptology. pp. 416–427. Springer (1989)
21. Dowling, B., Günther, F., Herath, U., Stebila, D.: Secure Logging Schemes and Certificate Transparency. In: European Symposium on Research in Computer Security. pp. 140–158. Springer (2016)

22. Durumeric, Z., Kasten, J., Bailey, M., Halderman, J.A.: Analysis of the HTTPS Certificate Ecosystem. In: Proceedings of the 2013 conference on Internet measurement conference. pp. 291–304. ACM (2013)
23. Dyer, J.: China Accused of Doling Out Counterfeit Digital Certificates in 'Serious' Web Security Breach. VICE News (April 2015)
24. Eckersley, P.: Sovereign Key Cryptography for Internet Domains. https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=HEAD (2012)
25. Electronic Frontier Foundation (EFF): The EFF SSL Observatory, https://www.eff.org/observatory
26. Fouque, P.A., Stern, J.: Fully Distributed Threshold RSA under Standard Assumptions. Advances in CryptologyASIACRYPT 2001 pp. 310–330 (2001)
27. Fromknecht, C., Velicanu, D., Yakoubov, S.: A Decentralized Public Key Infrastructure with Identity Retention. IACR Cryptology ePrint Archive **2014**, 803 (2014)
28. Gajek, S., Manulis, M., Pereira, O., Sadeghi, A.R., Schwenk, J.: Universally composable security analysis of tls. In: International Conference on Provable Security. pp. 313–327. Springer (2008)
29. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust Threshold DSS Signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 354–371. Springer (1996)
30. Goldreich, O.: Foundations of Cryptography. Cambridge University Press (2009)
31. Herzberg, A., Mass, Y., Mihaeli, J., Naor, D., Ravid, Y.: Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000. pp. 2–14. IEEE (2000)
32. Hruska, J.: Apple, Microsoft buck trend, refuse to block unauthorized Chinese root certificates. ExtremeTech (April 2015)
33. Huang, J., Nicol, D.M.: An anatomy of trust in public key infrastructure. International Journal of Critical Infrastructures **13**(2-3), 238–258 (2017)
34. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: Authenticated confidential channel establishment and the security of tls-dhe. J. Cryptology **30**(4), 1276–1324 (2017)
35. Kaynar, D.K., Lynch, N., Segala, R., Vaandrager, F.: The theory of timed i/o automata. Synthesis Lectures on Distributed Computing Theory **1**(1), 1–137 (2010)
36. Kim, T.H.J., Huang, L.S., Perrig, A., Jackson, C., Gligor, V.: Accountable Key Infrastructure (AKI): A Proposal for a Public-Key Validation Infrastructure. In: Proceedings of the 22nd international conference on World Wide Web. pp. 679–690. ACM (2013)
37. Kubilay, M.Y., Kiraz, M.S., Mantar, H.A.: CertLedger: A new PKI model with Certificate Transparency based on blockchain. arXiv preprint arXiv:1806.03914 (2018)
38. Laurie, B., Kasper, E.: Revocation Transparency. Google Research, September (2012)
39. Laurie, B., Langley, A., Kasper, E.: Certificate Transparency. RFC 6962 (Jun 2013). https://doi.org/10.17487/RFC6962, https://rfc-editor.org/rfc/rfc6962.txt
40. Leibowitz, H., Piotrowska, A.M., Danezis, G., Herzberg, A.: No Right to Remain Silent: Isolating Malicious Mixes. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 1841–1858 (2019)
41. Lekkas, D.: Establishing and managing trust within the Public Key Infrastructure. Computer Communications **26**(16), 1815–1825 (2003)
42. Li, J., Yuen, T.H., Kim, K.: Practical Threshold Signatures Without Random Oracles. In: Susilo, W., Liu, J.K., 0001, Y.M. (eds.) Provable Security, First International Conference, ProvSec 2007, Wollongong, Australia, November 1-2, 2007,

Proceedings. Lecture Notes in Computer Science, vol. 4784, pp. 198–207. Springer (2007)
43. Marchesini, J., Smith, S.: Modeling Public Key Infrastructure in the Real World. In: European Public Key Infrastructure Workshop. pp. 118–134. Springer (2005)
44. Matsumoto, S., Reischuk, R.M.: IKP: Turning a PKI Around with Decentralized Automated Incentives. In: Security and Privacy (SP), 2017 IEEE Symposium on. pp. 410–426. IEEE (2017)
45. Maurer, U.: Modelling a Public-Key Infrastructure. In: European Symposium on Research in Computer Security. pp. 325–350. Springer (1996)
46. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: International Conference on Computer Aided Verification. pp. 696–701. Springer (2013)
47. Melara, M.S., Blankstein, A., Bonneau, J., Felten, E.W., Freedman, M.J.: CONIKS: Bringing Key Transparency to End Users. In: USENIX Security Symposium. pp. 383–398 (2015)
48. Merkle, R.C.: A Digital Signature Based on a Conventional Encryption Function. In: Conference on the theory and application of cryptographic techniques. pp. 369–378. Springer (1987)
49. Merkle, R.C.: One Way Hash Functions and DES. In: Conference on the Theory and Application of Cryptology. pp. 428–446. Springer (1989)
50. Morrissey, P., Smart, N., Warinschi, B.: The tls handshake protocol: A modular analysis. Journal of Cryptology **23**, 187–223 (Apr 2010)
51. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. pp. 184–200. IEEE (2001)
52. Prins, J.: DigiNotar Certificate Authority breach Operation Black Tulip (2011)
53. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard) (Aug 2018). https://doi.org/10.17487/RFC8446, https://www.rfc-editor.org/rfc/rfc8446.txt
54. Roosa, S.B., Schultze, S.: 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 (2010)
55. Ryan, M.D.: Enhanced certificate transparency and end-to-end encrypted mail. In: NDSS (2014)
56. Samer, W.A., Romain, L., Francois, B., AbdelMalek, B.: A formal model of trust for calculating the quality of X. 509 certificate. Security and Communication Networks **4**(6), 651–665 (2011)
57. Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, D.C.: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960 (Jun 2013). https://doi.org/10.17487/RFC6960, https://rfc-editor.org/rfc/rfc6960.txt
58. Shoup, V.: Practical Threshold Signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 207–220. Springer (2000)
59. Syta, E., Tamas, I., Visher, D., Wolinsky, D.I., Ford, B.: Certificate Cothority: Towards Trustworthy Collective CAs. Hot Topics in Privacy Enhancing Technologies (HotPETs) **7** (2015)
60. Syta, E., Tamas, I., Visher, D., Wolinsky, D.I., Jovanovic, P., Gasser, L., Gailly, N., Khoffi, I., Ford, B.: Keeping Authorities Honest or Bust with Decentralized Witness Cosigning. In: Security and Privacy (SP), 2016 IEEE Symposium on. pp. 526–545. Ieee (2016)

61. Szalachowski, P., Matsumoto, S., Perrig, A.: PoliCert: Secure and Flexible TLS Certificate Management. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 406–417. ACM (2014)
62. Tomescu, A., Devadas, S.: Catena: Efficient Non-equivocation via Bitcoin. In: 2017 38th IEEE Symposium on Security and Privacy (SP). pp. 393–409. IEEE (2017)
63. Wendlandt, D., Andersen, D.G., Perrig, A.: Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. In: USENIX Annual Technical Conference. vol. 8, pp. 321–334 (2008)
64. Yu, J., Cheval, V., Ryan, M.: DTKI: A new formalized PKI with verifiable trusted parties. The Computer Journal **59**(11), 1695–1713 (2016)
65. Zhou, M., Bisht, P., Venkatakrishnan, V.: Strengthening XSRF Defenses for Legacy Web Applications Using Whitebox Analysis and Transformation. In: Information Systems Security, pp. 96–110. Springer (2011)

A Preliminaries

PoC-PKI and CT_{comp} use several underlying cryptographic schemes. In this section, we recall the definitions of these schemes and the security definitions. Specifically, we recall the definitions and security games of secure encryption scheme (Appendix A.1), secure signature scheme (Appendix A.2) and secure threshold signature scheme (Appendix A.3). Then, we recall the definition of CRHF (Appendix A.4) and a generalized Merkle tree scheme (Appendix A.5).

A.1 Encryption Scheme

Definition 5. *An encryption scheme* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *consists of the following probabilistic algorithms:*

- *Key generation* $\mathsf{Gen}(1^\kappa) \rightarrow (dk, ek)$*, with input a security parameter* 1^κ *and outputs: private decryption key* dk *and public encryption key* ek*.*
- *Encryption* $\mathsf{Enc}(ek, m) \rightarrow c$*, with inputs public key* ek *and message* m*, and output ciphertext* c*.*
- *Decryption* $\mathsf{Dec}(dk, c) \rightarrow m$*, with inputs private key* dk *and ciphertext* c*, and output message* m*.*

Definition 6. *An encryption scheme* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* ***CPA-indistinguishable*** *(CPA-IND), if for every PPT adversary* $\mathcal{A}$*:*

$$Pr[\mathbf{Exp}^{CPA-IND}_{\mathcal{E},\mathcal{A}}(1^\kappa) = 1] \in Negl(1^\kappa)$$

where $\mathbf{Exp}^{CPA-IND}_{\mathcal{E},\mathcal{A}}(1^\kappa)$*:*

1. $(dk, ek) \leftarrow \mathcal{E}.\mathsf{Enc}(1^\kappa)$*.*
2. *Adversary* $\mathcal{A}$ *chooses two messages* m_0, m_1*.*
3. *The game randomly chooses* $b \in \{0, 1\}$*.*
4. $\mathcal{A}$ $c = \mathcal{E}.\mathsf{Enc}(sk, m_b)$ *and outputs value* $b' \in \{0, 1\}$*.*
5. *The experiment outputs* 1 *if* $b = b'$*, otherwise, the experiment outputs* 0*.*

A.2 Digital Signature Scheme

Definition 7. *A signature scheme* $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *consists of the following probabilistic algorithms:*

- *Key generation* $\mathsf{Gen}(1^\kappa) \rightarrow (sk, vk)$*, with input security parameter* 1^κ *and output private signing key* sk *and public verification key* vk*.*
- *Signing* $\mathsf{Sign}(sk, m) \rightarrow (\sigma)$*, with input private signing key* sk *and a message* m*, and output signature* σ*.*
- *Verification* $\mathsf{Ver}(vk, m, \sigma) \rightarrow (\top/\bot)$*, with inputs public verification key* vk*, message* m *and signature* σ*, and output: true* ($\top$) *if* σ *is a valid signature over* m*, otherwise false* ($\bot$)*.*

Definition 8. *A signature scheme* $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *is* ***existentially unforgeable*** *if for every PPT adversary* $\mathcal{A}$*:*

$$Pr\left[\mathbf{Exp}^{EU}_{\mathcal{S},\mathcal{A}}(1^\kappa) = 1\right] \in Negl(1^\kappa)$$

where $\mathbf{Exp}^{EU}_{\mathcal{S},\mathcal{A}}(1^\kappa)$*:*

1. $(sk, vk) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$
2. *Adversary* $\mathcal{A}$ *receives* vk *and has an oracle access to* $\mathcal{S}.\mathsf{Sign}$ *to sign any message it desires.*
3. $\mathcal{A}$ *outputs message* m *and signature* σ*.*
4. *The experiment outputs* 1 *if* $\mathcal{S}.\mathsf{Ver}(vk, m, \sigma) = \top$ *and* $\mathcal{A}$ *did not use the oracle access on* m*, otherwise, the experiment outputs* 0*.*

A.3 Threshold Signature Scheme

While definitions 5,6,7,8 are standard definitions of encryption and signature schemes, choosing a definition of a threshold signature scheme is slightly more complex; in particular, it appears there is no one dominant definition. Furthermore, as opposed to classic encryption and signature schemes which are local, i.e., each operation of the scheme is performed by only one entity and does not require interaction with other entities, threshold signature schemes are inherently different, and existing definitions we have seen involve a *distributed algorithm* as part of the definition. For example, in [29], the threshold scheme is presented similarly to a classical signature scheme, with the key generation algorithm modified to output shares of the signature key; however, the signing algorithm is a distributed algorithm that outputs the *final* group signature. Another example is [58], where a signature share verification algorithm allows to check whether a specific partial signature is valid, i.e., was indeed signed by the matching entity on the respective message. Other works, such as [26, 42], present and discuss other possible designs, definitions and implementations. However, in all of these, the definition includes a protocol or distributed algorithm. This introduces dependency on the communication and synchronization models, and make it less convenient to use threshold signatures as part of a design supporting different models, as we do.

Instead, we define the threshold scheme using the following *four* algorithms: $\mathcal{TS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Ver})$, where: $\mathcal{TS}.\mathsf{Gen}$ is the key generation algorithm, $\mathcal{TS}.\mathsf{Sign}$ is the *individual* signing algorithm, $\mathcal{TS}.\mathsf{Combine}$ is the signatures combining algorithm, and $\mathcal{TS}.\mathsf{Ver}$ is the group signature verification algorithm. We also define matching security and robustness definitions. We believe that our definitions generalize existing definitions, and allow a simpler presentation of PoC-PKI and other protocols; however, further work, beyond the scope of this paper, is required to confirm the relationships between the different threshold signature definitions.

Definition 9. *A (t, n)-*threshold-signature *scheme $\mathcal{TS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Ver})$ consists of the following probabilistic algorithms:*

- *Key-generation* $\mathsf{Gen}(1^\kappa, n, t) \rightarrow (tvk, \{tsk_i\}_{i=1}^n)$*, with inputs security parameter 1^κ, total number of entities n, the threshold value $t < n$, and outputs a group verification key tvk and n secret shares $\{tsk_i\}_{i=1}^n$ of the signature key.*
- *Signing* $\mathsf{Sign}(tsk_i, m) \rightarrow \sigma_i$*, with input secret share key tsk_i and message m, and output* partial *signature σ_i.*
- *Combining* $\mathsf{Combine}(\{\sigma_i\}) \rightarrow \sigma/\perp$*, with input set of partial signatures $\{\sigma_i\}$ and output threshold signature σ or failure $\perp$.*
- *Verification* $\mathsf{Ver}(tvk, m, \sigma) \rightarrow (\top/\perp)$*, with input group verification key tvk, messages m and threshold signature σ, and output $\top$ or $\perp$.*

Definition 10. *A (t, n)-threshold-signature scheme $\mathcal{TS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Ver})$ is **robust**, if executing the $\mathcal{TS}.\mathsf{Combine}$ algorithm with more than t valid partial signatures, i.e., signatures generated by $\mathcal{TS}.\mathsf{Sign}$ on the same message using valid key shares,* always *generates a valid group signature. Namely, for any message space M and any set of entities N where $|\mathsf{N}| = n$, and any adversary $\mathcal{A}$ that controls up to t entities (denoted by $\mathsf{N_F} \subset \mathsf{N}$ s.t. $|\mathsf{N_F}| \leq t$), and given keys $(tvk, \{tsk_i\}_{i\in\mathsf{N}}) \leftarrow \mathcal{TS}.\mathsf{Gen}(1^\kappa, |\mathsf{N}|, t)$, it holds that:*

$$\begin{aligned}(\forall m \in M, W \subseteq \{\sigma_i = \mathcal{TS}.\mathsf{Sign}(tsk_i, m) \mid i \in \mathsf{N} - \mathsf{N_F}\},\\ W' \leftarrow \mathcal{A}) |W| \geq t + 1 \Rightarrow \\ \mathcal{TS}.\mathsf{Ver}(tvk, m, \mathcal{TS}.\mathsf{Combine}(W \cup W')) = \top\end{aligned} \tag{7}$$

Definition 11. *Let $OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(\iota, m)$ be a PPT oracle algorithm that can produce a cryptographically valid individual signature signed by any authority $\iota \in \mathsf{N}$ on messages m using the $\mathcal{TS}.\mathsf{Sign}$ algorithm, i.e., $OTSign_{\{tsk\}_{i\in\mathsf{N}}}(\iota, m) \equiv \mathcal{TS}.\mathsf{Sign}(tsk_\iota, m)$. A (t, n)-threshold-signature scheme $\mathcal{TS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Ver})$ is $g-$existentially unforgeable, where g is a function of the number of shares n, if there is no PPT adversary $\mathcal{A}$ that controls up to f players ($f \leq t$) and has an oracle access to $OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(\cdot, \cdot)$ that can produce a cryptographically valid group signature on any previously unsigned message m, without performing more than $t - f$ oracle requests for message m. Namely, for every adversary $\mathcal{A}$:*

$$Pr\left[\mathbf{Exp}^{EU}_{\mathcal{TS},\mathcal{A}}(1^\kappa, \mathsf{N}, t) = 1\right] \in Negl(1^\kappa)$$

where $\mathbf{Exp}^{EU}_{\mathcal{TS},\mathcal{A}}(1^\kappa, \mathsf{N}, t)$ *is defined as:*

1. $(tvk, \{tsk\}_{i\in\mathsf{N}}) \leftarrow \mathcal{TS}.\mathsf{Gen}(1^\kappa, |\mathsf{N}|, t)$
2. $(m, \sigma) \leftarrow \mathcal{A}^{OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(\cdot,\cdot)}_{tvk}$ *($\mathcal{A}$ outputs message m and group signature σ, given tvk and oracle access to $OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}$)*
3. *Return* 1 *if* $\mathcal{TS}.\mathsf{Ver}(tvk, m, \sigma) = \top$ *and $\mathcal{A}$ did not use the oracle access on m more than $t - f$ times, otherwise, the experiment outputs* 0.

A.4 Collision-Resistant Hash Functions

We recall the definitions of keyed and keyless collision resistant hash functions (CRHFs), as described in [6].

Definition 12. *Let κ be some sufficiently large security parameter, $M = \{0,1\}^*$ be some message space, $K = \{0,1\}^*$ be some key space and $T = \{0,1\}^\kappa$ be a digest space. A* ***collision resistant keyed hash function*** *H is an efficient deterministic algorithm that takes two inputs, a key $k \in K$ and a message $m \in M$ and outputs $t = H(k, m) \in T$ that for every PPT adversary $\mathcal{A}$:*

$$Pr\left[((m_0, m_1 \in M) \leftarrow \mathcal{A}(1^\kappa))(m_0 \neq m_1 \wedge H(k, m_0) = H(k, m_1))\right] \in Negl(1^\kappa)$$

Definition 13. *Given a sufficiently large security parameter κ, a* ***keyless collision resistant hash function*** *$H : 1^\kappa \times M \to T$ is an efficiently computable function from some message space $M = \{0,1\}^*$ to a digest space $T = \{0,1\}^\kappa$ that for every PPT adversary $\mathcal{A}$:*

$$Pr\left[((m_0, m_1 \in M) \leftarrow \mathcal{A}(1^\kappa))(m_0 \neq m_1 \wedge H(1^\kappa, m_0) = H(1^\kappa, m_1))\right] \in Negl(1^\kappa)$$

A.5 Generalized-Merkle-Tree Scheme

A Merkle-tree is a well-known cryptographic construction introduced by Merkle [48]. In essence, a Merkle tree is a construction that takes a list of entries E and constructs a binary tree which its leaves are the entries in E and each inner tree node is the result of applying a collision-resistant hash function (CRHF) over its child nodes. The root of the resulting *hash tree* uniquely identifies the list of entries (the tree leaves), in a secure way. Namely, the Merkle tree construction ensures *collision resistance* for its inputs, i.e., is a CRHF for strings of arbitrary length, even if the underlying hash function h is only a Fixed-Input-Length CRHF; note that this goal is also achieved by the Merkle-Damgard construction [20, 49].

There are numerous applications and variants of the Merkle tree construction, e.g., for blockchains and for PKI schemes [3, 21, 39]; many of these applications take advantage of two additional security properties:

Proof of Inclusion. The collision-resistance property of the Merkle tree allows a recipient to confirm that the root of the hash tree corresponds to a particular list of entries E; however, in many applications, we want to validate

that the list E *contains* a particular element e, without providing the entire list for the validation process - allowing for better efficiency, as well as, possibly, privacy. Merkle trees facilitate such efficient validation of proofs of inclusion of a particular element. (Merkle trees may also allow efficient proofs of inclusion for multiple elements, and privacy for other elements, under appropriate assumptions regarding the underlying hash function h; however, these properties are not required for our needs.)

Proof of consistency. Merkle trees can also be *consistent*; that is, given two Merkle trees t_1, t_2 that represent lists E_1, E_2 respectively, tree t_2 can be proven to be consistent with t_1 if the first $|E_1|$ items of E_2 are identical to E_1. This property is very useful, e.g., in applications that require *tamper-free, append-only* log of entries like Certificate Transparency. Namely, certificates can only be added to the list/tree and cannot be retroactively added/changed, and this can be easily proved to anyone.

Merkle trees are indeed applied extensively - however, insufficient attention was given to defining and proving their security properties. One exception is [21], which proved some security properties for a specific variant of the Merkle tree construction. We mostly follow and use their approach and construction, with a few changes. First, we define the aforementioned security properties: collision-resistance, proof-of-inclusion and proof-of-consistency, which differ from the (two) properties proven in [21]. Second, we present the construction as an implementation of a *Generalized Merkle Tree ($\mathcal{GMT}$)* scheme, allowing use of the $\mathcal{GMT}$ in protocols, regardless of the specific construction (implementation), allowing for future constructions with additional properties. That said, we emphasize that the construction described in §A.5.3 is almost identical to the one described in [21].

Definition 14. *A generalized Merkle tree scheme is a tuple*

$$\mathcal{GMT} = (\mathsf{GenTree}, \mathsf{GenIncProof}, \mathsf{VerIncProof}, \mathsf{GenConsProof}, \mathsf{VerConsProof})$$

which consists of the following algorithms:

- *A Merkle tree hash generation algorithm* $\mathcal{GMT}.\mathsf{GenTree}(\kappa, E) \to \varphi$*, which takes as input a security parameter* κ *and a an ordered set (list) of binary strings (entries)* E*, and outputs a Merkle tree root* φ*. We refer to* E *as the* tree leaves*.*
- *A proof of inclusion generation algorithm* $\mathcal{GMT}.\mathsf{GenIncProof}(\kappa, i, E) \to \sigma$*, which takes as input a security parameter* κ*, an index* i *and entries* E*, and outputs a string* σ*, which we refer to as a* Proof of Inclusion *of* $E(i)$ *in the output of* $\mathcal{GMT}.\mathsf{GenTree}(1^\kappa, E)$*.*
- *A proof of inclusion verification algorithm* $\mathcal{GMT}.\mathsf{VerIncProof}(\kappa, e, i, n, \varphi, \sigma) \to \{\top, \bot\}$*, which takes as input an entry* e*, an index* i*, a number* n*, a string* φ *and a string* σ*, and outputs* $\top$ *if* σ *is a valid proof of inclusion for entry* e *of index* i *from a list of entries of size* n *which its Merkle tree root is* φ*, or* $\bot$ *otherwise.*
- *A proof of consistency generation algorithm* $\mathcal{GMT}.\mathsf{GenConsProof}(\kappa, i, j, E) \to \sigma$*, which takes as input a security parameter* κ*, indexes* i, j *and entries* E

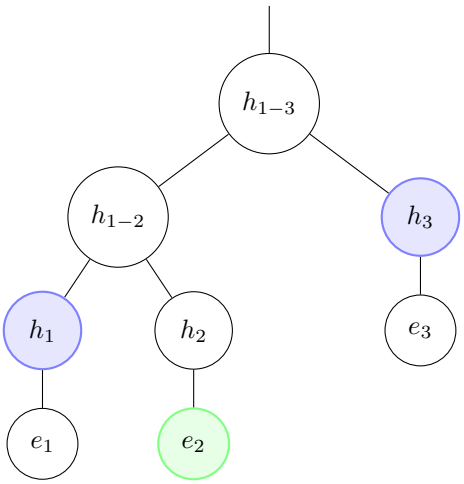

Fig. 1: Example Merkle hash-tree, representing a list of entries $E = \{e_1, e_2, e_3\}$. Every internal node is the hash of its children; in particular, for every i holds $h_i = h(e_i)$, and $h_{i-j)} = h(h_i||h_j)$. Each entry e_i can be proven to included in the tree. For example, to validate that entry e_2 is included in the tree hash h_{1-3}, the values of internal hash nodes required to validate the hash are h_1 and h_3.

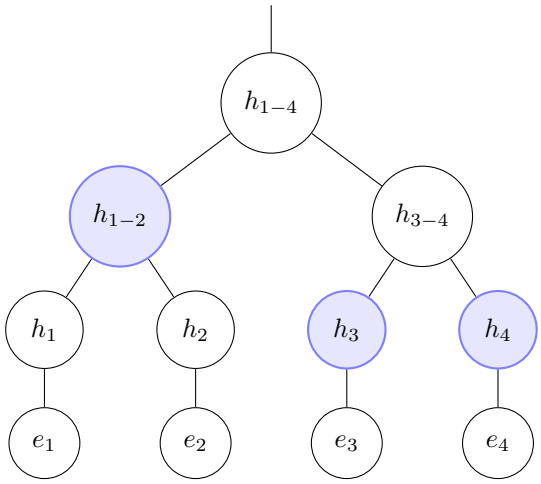

Fig. 2: Example of proof of consistency. Adding a new entry e_4 to the Merkle tree described in Fig. 1. After this addition, to prove consistency from tree root h_{1-3} (from Fig. 1) to h_{1-4}, the values of internal hash nodes required are (h_3, h_4, h_{1-2}). From h_{1-2} and h_3 we get h_{1-3}, and from h_3 and h_4 we get h_{3-4}, which with h_{1-2} gives h_{1-4}.

and outputs a proof of consistency σ from the Merkle root of $E[0:i]$ to the Merkle root of $E[0:j]$.

- *A proof of consistency verification algorithm* $\mathcal{GMT}.\mathsf{VerConsProof}(\kappa, i, j, \varphi_i, \varphi_j, \sigma) \rightarrow \{\top, \bot\}$*, which takes as input a security parameter κ, indexes i, j, Merkle tree root φ_i, a second Merkle tree root φ_j and a proof of consistency σ and outputs $\top$ if σ is a valid proof of consistency from the tree represented by φ_i to the tree represented by φ_j or $\bot$ otherwise.*

A.5.1 GMT Correctness properties

Definition 15. *A generalized Merkle tree scheme*

$$\mathcal{GMT} = (\mathsf{GenTree}, \mathsf{GenIncProof}, \mathsf{VerIncProof}, \mathsf{GenConsProof}, \mathsf{VerConsProof})$$

satisfies ***correctness of inclusion proofs*** *if:*

$$(\forall\ \kappa, E, 1 \geq i \leq |E|; \varphi \leftarrow \mathcal{GMT}.\mathsf{GenTree}(1^\kappa, E)) \Rightarrow$$
$$(\mathcal{GMT}.\mathsf{VerIncProof}(E[i], \varphi, \mathcal{GMT}.\mathsf{GenIncProof}(1^\kappa, i, E))) = \top$$

Definition 16. *A generalized Merkle tree scheme*

$$\mathcal{GMT} = (\mathsf{GenTree}, \mathsf{GenIncProof}, \mathsf{VerIncProof}, \mathsf{GenConsProof}, \mathsf{VerConsProof})$$

satisfies ***correctness of consistency proofs*** *if:*

$$(\forall\ \kappa, E, 1 \leq i \leq |E|; \varphi \leftarrow \mathcal{GMT}.\mathsf{GenTree}(1^\kappa)E) \Rightarrow$$
$$(\mathcal{GMT}.\mathsf{VerConsProof}(\mathcal{GMT}.\mathsf{GenTree}(1^\kappa, E[1:i]), \varphi,$$
$$\mathcal{GMT}.\mathsf{GenConsProof}(1^\kappa, i, E))) = \top$$

A.5.2 GMT Security properties

Definition 17. *A generalized Merkle tree scheme is* ***collision resistant****, if it is computationally hard to find two* different *ordered lists E, E' that have the same Merkle tree root. Namely, if for every PPT adversary $\mathcal{A}$ and a sufficiently large security parameter κ:*

$$Pr\begin{bmatrix}(E, E') \leftarrow \mathcal{A}(1^\kappa)(E \neq E' \wedge E, E' \in \{\{0,1\}^*\} \\ \wedge \mathcal{GMT}.\mathsf{GenTree}(E) = \mathcal{GMT}.\mathsf{GenTree}(E'))\end{bmatrix} \in Negl(1^\kappa)$$

Definition 18. *A generalized Merkle tree scheme* ***has secure proofs of inclusion****, if inclusion proofs can only be generated for the tree leaves. Namely, if for every PPT adversary $\mathcal{A}$, every list of entries E and a sufficiently large security parameter κ:*

$$Pr\left[\begin{array}{l}(e,\sigma) \leftarrow \mathcal{A}(1^\kappa)(e \notin E \wedge \\ \mathcal{GMT}.\mathsf{VerIncProof}(e, \mathcal{GMT}.\mathsf{GenTree}(E), \sigma) = \top)\end{array}\right] \in Negl(1^\kappa)$$

Definition 19. *A generalized Merkle tree scheme is* ***consistent****, if it is computationally hard to find a consistency proof to two ordered lists E, E' where E is not a subset of E'. Namely, if for every PPT adversary $\mathcal{A}$ and a sufficiently large security parameter κ:*

$$Pr\left[\begin{array}{l}(E,E',\sigma) \leftarrow \mathcal{A}(1^\kappa)(E \not\subseteq E' \wedge \\ \mathcal{GMT}.\mathsf{VerConsProof}(\mathcal{GMT}.\mathsf{GenTree}(E), \\ \mathcal{GMT}.\mathsf{GenTree}(E'), \sigma) = \top)\end{array}\right] \in Negl(1^\kappa)$$

A.5.3 The Merkle Tree Construction

Given a CRHF h, we now define the *Merkle tree* construction.

Algorithm 9 $\mathcal{GMT}_{h\in\mathcal{H}}.\mathsf{GenTree}(\kappa, E)$

1: **if** $|E| = 1$ **then return** $h(\kappa, 0 \mid\mid E[0])$
2: **else if** $|E| > 1$ **then**
3: $\quad i \leftarrow 2^{\left\lceil log_2\left(\frac{|E|}{2}\right)\right\rceil}$
4: $\quad$ **return** $h(\kappa, 1 \mid\mid \mathcal{GMT}_h.\mathsf{GenTree}(\kappa, E[0:i]) \mid\mid \mathcal{GMT}_h.\mathsf{GenTree}(\kappa, E[i:|E|]))$
5: **else return** $\perp$
6: **end if**

Algorithm 10 $\mathcal{GMT}_{h\in\mathcal{H}}.\mathsf{GenIncProof}(\kappa, i, E)$

1: **if** $|E| > 1$ **then**
2: $\quad j \leftarrow 2^{\left\lceil log_2\left(\frac{|E|}{2}\right)\right\rceil}$
3: $\quad$ **if** $i < j$ **then return** $\mathcal{GMT}_h.\mathsf{GenIncProof}(\kappa, i, E[0:j]) \mid\mid \mathcal{GMT}_h.\mathsf{GenTree}(\kappa, E[j:|E|])$
4: $\quad$ **else return** $\mathcal{GMT}_h.\mathsf{GenIncProof}(\kappa, i-j, E[j:|E|]) \mid\mid \mathcal{GMT}_h.\mathsf{GenTree}(\kappa, E[0:j])$
5: $\quad$ **end if**
6: **end if**
7: **return** $\perp$

Algorithm 11 $\mathcal{GMT}_{h\in\mathcal{H}}.\mathsf{VerIncProof}(\kappa, e, i, n, \varphi, \sigma)$

return $\varphi = \textsc{Root}_h(\kappa, e, i, n, \sigma)$

procedure $\textsc{Root}_h(\kappa, e, i, n, \sigma)$
  if $n = 1$ **then return** $h(0 \mid\mid e)$
  $j \leftarrow 2^{\lceil log_2\left(\frac{n}{2}\right)\rceil}$
  if $i < j$ **then**
    $\ell \leftarrow \textsc{Root}_h(\kappa, e, i, j, \sigma[0 : |\sigma| - 1])$
    $r \leftarrow \sigma[|\sigma| - 1]$
  else
    $\ell \leftarrow \sigma[|\sigma| - 1]$
    $r \leftarrow \textsc{Root}_h(\kappa, e, i - j, n - j, \sigma[0 : |\sigma| - 1])$
  end if
  return $h(\kappa, 1 \mid\mid \ell \mid\mid r)$
end procedure

Algorithm 12 $\mathcal{GMT}_{h\in\mathcal{H}}.\mathsf{GenConsProof}(\kappa, i, j, E)$

if $0 \leq i < j \leq |E|$ **then return** $\textsc{Proof}_h(\kappa, i, E[0 : j], \top)$
return $\perp$

procedure $\textsc{Proof}_h(\kappa, i, E, b)$
  if $i = |E| \ \wedge \ b = \perp$ **then return** $\mathcal{GMT}_h.\mathsf{GenTree}(\kappa, E[0 : i])$
  else
    $j \leftarrow 2^{\left\lceil log_2\left(\frac{|E|}{2}\right)\right\rceil}$
    if $i \leq j$ **then** **return** $\textsc{Proof}_h(\kappa, i, E[0 : j], b) \mid\mid \mathcal{GMT}_h.\mathsf{GenTree}(\kappa, E[j : |E|])$
    else **return** $\textsc{Proof}_h(\kappa, i - j, E[j : |E|], \perp) \mid\mid \mathcal{GMT}_h.\mathsf{GenTree}(\kappa, E[0 : j])$
    end if
  end if
end procedure

Algorithm 13 $\mathcal{GMT}_{h\in\mathcal{H}}.\mathsf{VerConsProof}(\kappa, i, j, \varphi_i, \varphi_j, \sigma)$

if i is a power of 2 **then** $\sigma \leftarrow \varphi_i \mid\mid \sigma$
$\varphi'_i \leftarrow Root0_h(\kappa, i, j, \sigma)$
$\varphi'_j \leftarrow Root1_h(\kappa, i, j, \sigma)$
return $((\varphi_i = \varphi'_i) \ \wedge \ (\varphi_j = \varphi'_j))$

procedure $Root0_h(\kappa, i, j, \sigma)$
    $k \leftarrow 2^{\lceil log_2(\frac{j}{2}) \rceil}$
    if $i < k$ **then return** $Root0_h(\kappa, i, k, \sigma[0 : |\sigma| - 1])$
    else if $i = k$ **then return** $\sigma[|\sigma| - 2]$
    else
        $\ell \leftarrow \sigma[|\sigma| - 1]$
        $r \leftarrow Root0_h(\kappa, i - k, j - k, \sigma[0 : |\sigma| - 1])$
        return $h(\kappa, 1 \mid\mid \ell \mid\mid r)$
    end if
end procedure
procedure $Root1_h(\kappa, i, j, \sigma)$
    if $|\sigma| = 2$ **then return** $h(\kappa, 1 \mid\mid \sigma[0] \mid\mid \sigma[1])$
    $k \leftarrow 2^{\lceil log_2(\frac{j}{2}) \rceil}$
    if $i < k$ **then**
        $\ell \leftarrow Root1_h(\kappa, i, k, \sigma[0 : |\sigma| - 1])$
        $r \leftarrow \sigma[|\sigma| - 1]$
    else
        $\ell \leftarrow \sigma[|\sigma| - 1]$
        $r \leftarrow Root1_h(\kappa, i - k, j - k, \sigma[0 : |\sigma| - 1])$
    end if
    return $h(\kappa, 1 \mid\mid \ell \mid\mid r)$
end procedure

B PoC-PKI: A Provably-Secure PKI

We now describe the PoC-PKI system, a provably-secure 'proof-of-concept' PKI scheme. PoC-PKI is designed for simplicity rather than efficiency or deployability. PoC-PKI provably meets all PKI requirements (see Appendix C for rigorous proofs and analysis).

B.1 High-Level Overview

In PoC-PKI, there is only one type of entities, which are called *authorities*, i.e., the set N is the set of authorities in PoC-PKI. Authorities issue certificates to clients using the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Issue}$ algorithm, which outputs the simplest form of a valid certificate in PoC-PKI, which is an *accountable* certificate, i.e., a certificate with the ACC attribute.

Similarly, authorities revoke certificates upon client's request, using the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Revoke}$ algorithm, which outputs a revoked certificate, i.e., the same inputted certificate but with the REV attribute. Both the ACC and the REV attributes are implemented using a secure signature scheme, used by the authority to generate a proof that the relevant certificate has these attributes. Namely, a certificate ψ has attribute *attr* if $\psi.\rho[attr].\sigma$ is a valid signature over $Core(\psi)$ and *attr* signed by $\psi.\rho[attr].\iota \in \mathsf{N}$ on local time $\psi.\rho[attr].clk$.

A client can request any authority to upgrade the client's accountable certificate, by adding to it the transparency attribute. If a client requests to upgrade an accountable certificate into a $\Delta_{\text{PoC-PKI}}$-transparent certificate, the authority uses the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Upgrade}$ algorithm to output an upgraded certificate, i.e., a certificate with the $\Delta\mathsf{TRA}$ attribute. Similarly, the same can be done when a client requests a $\Delta_{\text{PoC-PKI}}$-revocation transparency upgrade; the authority uses the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Upgrade}$ algorithm to output an upgraded certificate with the $\Delta\mathsf{ReTRA}$ attribute. Note that transparency in PoC-PKI *does not* involve pending certificates, and the outputted certificate is a *non-pending* upgraded certificate. The $\Delta\mathsf{TRA}$ and $\Delta\mathsf{ReTRA}$ attributes are implemented using proofs (like the ACC,REV attributes), but in addition, the authority immediately broadcasts the upgraded certificate to the rest of the authorities, so that they all know about the upgraded certificate before the $\Delta_{\text{PoC-PKI}}$ time period passes.

When a client requests to upgrade a certificate into an unequivocal certificate, the authority also use the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Upgrade}$ algorithm, but this time, the algorithm outputs a *pending* certificate, i.e., a certificate with the $\mathsf{EQ\text{-}P}$ attribute (pending unequivocal). The reason is that equivocation prevention is achieved in PoC-PKI using the active involvement of other authorities, and therefore, a pending certificate is generated until the upgrade process completes. Once the process is completed, the authority will upgrade the client's certificate into an unequivocal certificate, based on the information gathered from other authorities. However, this process may also fail, e.g., because during this process the authority learns about another (pending or not) unequivocal non-revoked certificate that conflicts with the pending certificate. If the process fails, the upgrading

authority never upgrades the pending certificate into an actual unequivocal non-pending certificate.

PoC-PKI implements this upgrade process using secure and robust *threshold signature* scheme $\mathcal{TS}$. Namely, the authority that issued the EQ-P pending certificate informs the rest of the authorities about the pending certificate, asking them to confirm whether they approve this upgrade request or not. Essentially, the only reason that an authority disapproves such request is if the authority is aware of some *other unequivocal, non-revoked* certificate that was issued for the same identifier for an overlapping period, i.e., a conflicting certificate (other criterias could also be used). If that is the case, the disapproving authority informs the upgrading authority about the existing certificate, thus providing the upgrading authority with a legitimate reason not to complete the upgrade. Otherwise, if such conflicting certificate does not exists, each authority approves the upgrade by using its share of the threshold-signing key and sends back a partial-signature for the certificate upgrade. Upon receiving a sufficient set of partial signatures, and not receiving any conflicting certificate, the authority generates and returns the certificate with the properly-threshold-signed EQ-P attribute.

Every authority in PoC-PKI can provide certificates with the aforementioned attributes. Relying parties are expected to ignore certificates or attributes where the signer is not authorized; however, the 'authorization' aspect, e.g., naming-constraints, is not part of the scheme and is left for the actual system that adopts PoC-PKI. Further, systems that use PoC-PKI can decide for themselves what type of certificates they are willing to support. That is to say, that although equivocation prevention is the strongest property suggested by PoC-PKI, systems can definitely accept and trust certificates which are 'only' accountable, transparent or pending-unequivocal. Of course, the system designers should take into consideration the proportional security guarantees. Furthermore, systems might consider using such certificates as 'temporary' certificates which might be considered less trusted, but can be useful until a certificate becomes unequivocal.

B.2 Model Function $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$

We define the model of PoC-PKI as

$$\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}(\xi) = \mathcal{M}^{\text{3-rounds}}_{\text{SecInit}}(\xi) \wedge \mathcal{M}^{|\mathsf{N_F}|\leq\lfloor(|\mathsf{N}|/3)\rfloor}(\xi) \wedge \mathcal{M}^{\text{COM}}_{\Delta_{com}}(\xi) \wedge \mathcal{M}^{\text{CLK}}_{\Delta_{clk}}(\xi)$$

where, $\mathcal{M}^{\text{3-rounds}}_{\text{SecInit}}$ is the secure initialization model described in §3.2.2, $\mathcal{M}^{|\mathsf{N_F}|\leq\lfloor(|\mathsf{N}|/3)\rfloor}$ is the fault model described in §3.2.1 which sets the number of entities that can arbitrarily misbehave to less than a third of all entities, $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$ is the communication model as described in §3.2.3, $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$ is the clock synchronization model as described in §3.2.4.

We use $\Delta_{\text{PoC-PKI}} = \Delta_{com} + \Delta_{clk}$ to denote the delay ensured by PoC-PKI. The reason for this value is that when an authority upgrades a certificate with transparency or revocation transparency, it immediately broadcasts the upgraded certificate to the rest of the entities. Since the $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ model ensures that the adversary cannot prevent these messages from being delivered, it

takes at most Δ_{com} time for the messages to arrive to other entities. However, entities' clocks might not be completely synchronized, which can add up to Δ_{clk} additional delay.

B.3 Local State Variables

Each entity has a local clock clk and a local state s. All entities store the following information:

- $s.\mathsf{N}$: the unique-identifiers of all entities.
- $s.\iota$: the entity's unique-identifier.
- $s.PrivInfo$: private (secret) information.
- $s.PubInfo_i$: the public information of entity $i \in \mathsf{N}$.
- $s.certs$: all the certificates known to the entity.

B.4 PoC-PKI Algorithms

B.4.1 Init Algorithm

The initialization algorithm PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Init}$ (Algorithm 14) consists of three rounds. In the first round (lines 1-6), each entity extracts from the input inp and store locally the set of entities $s.\mathsf{N}$ and its identifier $s.\iota$. Then, each entity generates a private/public signing key pair and encryption/decryption key pair. In line 3, the algorithm produces the public key attribution statement (see §3.3.1). The public keys are then sent to all other entities to be processed in the second round. In the second round (lines 7-13), each entity extract from the input the public keys sent by the other entities in the first round, and stores them locally. Then, a trusted designated entity generate group keys. This process could be alternatively implemented using a secure and distributed algorithm, which would eliminate the need for a trusted party; however, for simplicity, we pick entity number 1 (without loss of generality) to be the trusted entity in this implementation. Furthermore, entity number 1 needs to be trusted only during the secure initialization phase, and does not have to be trusted after that. Entity number 1 uses $\mathcal{TS}.\mathsf{Gen}$ to generate group verification key and $|\mathsf{N}|$ partial signature key secret shares tsk_i. Then, it encrypts the partial secret share tsk_i for every authority $i \in \mathsf{N}$ and sends to each entity its encrypted partial secret share along with the group verification key. In the third and final round, each authority stores the group verification key and decrypts and stores its encrypted partial secret group signing key.

B.4.2 Issue Algorithm

The certificate issuance algorithm PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Issue}$ (Algorithm 15) takes as input the certificate details: identity id, public information (incl. key) pub, and start, end dates sd, ed. The algorithm generates a signature σ using the $\mathcal{S}.\mathsf{Sign}$ algorithm over the inputted details (line 1) and generates a matching certificate ψ for the inputted details with the ACC attribute (accountability) and uses σ as

Algorithm 14 PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Init}()$

1: **if** $s.initCounter = 0$ **then**
2: $(s.PrivInfo.sk, s.PubInfo_{s.\iota}.vk) \leftarrow \mathcal{S}.\mathsf{Gen}(s.1^{\kappa})$
3: $out \leftarrow$ ('public key', $s.PubInfo_{s.\iota}.vk$)
4: $(s.PrivInfo.dk, s.PubInfo_{s.\iota}.ek) \leftarrow \mathcal{E}.\mathsf{Gen}(s.1^{\kappa})$
5: $s.initCounter \leftarrow 1$
6: $out \leftarrow \{(\text{'send'}, m = [PubInfo = s.PubInfo_{s.\iota}], j)\}_{j \in s.\mathsf{N} - s.\iota}$
7: **else if** $s.initCounter = 1$ **then**
8: $s.PubInfo_{inp.j} \leftarrow inp.m.PubInfo$
9: **if** $s.\iota = 1$ **then**
10: $(s.PubInfo.tvk, \{tsk_i\}_{i=1}^{|s.\mathsf{N}|}) \leftarrow \mathcal{TS}.\mathsf{Gen}(s.1^{\kappa}, |s.\mathsf{N}|, \lfloor |s.\mathsf{N}|/3 \rfloor)$
11: $E \leftarrow \{e_i = \mathcal{E}.\mathsf{Enc}(s.PubInfo_i.ek, tsk_i)\}_{i \in \mathsf{N}}$
12: $out \leftarrow \{(\text{'send'}, m = [tvk = s.PubInfo.tvk, tsk = E.e_j)], j)\}_{j \in s.\mathsf{N} - s.\iota}$
13: **end if**
14: $s.initCounter \leftarrow 2$
15: **else if** $s.initCounter = 2$ **then**
16: $s.PubInfo.tvk \leftarrow inp.m.tvk$
17: $s.PrivInfo.tsk \leftarrow \mathcal{S}.\mathsf{Dec}(s.PrivInfo.dk, inp.m.tsk)$
18: $s.initCounter \leftarrow 3$
19: **end if**

a proof of accountability (lines 2-3). The algorithms stores the newly generated certificate in its local state $s.certs$ (line 4) and outputs the accountable certificate ψ.

Algorithm 15 PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Issue}(id, pub, sd, ed)$

Comment: An honest authority invokes *issue* only if the client that request ownership over id is eligible for id and the authority is authorized to issue certificates for id.

// Generate a basic certificate
1: $data \leftarrow (id, pub, sd, ed, \mathsf{ACC}, clk)$
2: $\sigma = \mathcal{S}.\mathsf{Sign}(s.PrivInfo.sk, data)$
3: $\rho \leftarrow \{(\mathsf{ACC}, (\sigma, s.\iota, clk))\}$
4: $\psi \leftarrow (id, pub, sd, ed, \rho)$
// Add the new certificate to the local state
5: $s.certs \mathrel{+}= \psi$
6: **return** ψ

B.4.3 Audit Algorithm

The certificate audit algorithm PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Audit}$ (Algorithm 16) checks whether there are problems with inputted certificate ψ with respect to the inputted at-

tribute *attr*. When the *attr* attribute refers to accountability, revocation accountability or non-revocation accountability, the algorithm returns the output of the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.WasValid algorithm on ψ and *attr*, as it does not have any additional requirements besides the cryptographic correctness enforces by WasValid. For the rest of the attributes, WasValid is not enough, so the algorithm performs *additional* checks. When the *attr* attribute refers to Δ-transparency or $\Delta_{\text{PoC-PKI}}$-revocation transparency, the algorithm checks if ψ indeed appears in its local state, and if not, the algorithm outputs a matching IA (Indication of Accusation). When the *attr* attribute refers to equivocation prevention, the algorithm checks if ψ indeed appears in its local state, and if not, the algorithm outputs a matching IA (Indication of Accusation).

Algorithm 16 PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Audit($\psi, attr$)

1: $v \leftarrow$ PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.WasValid($\psi$, $attr$)
2: **if** $attr \in \{$ACC, REV, NREV, EQ-P$\}$ **then return** v
3: **if** $attr \in \{\Delta$TRA, ΔReTRA$\}$ $\wedge$ v $\wedge$ **then**
($\nexists \psi$' $\in s.certs$ *s.t.* ($Core(\psi) = Core(\psi')$ $\wedge$ PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.WasValid($\psi$', $attr$)))
4: **if** $s.PubInfo_{\psi.\rho[attr].\iota}.accusation = \perp$ **then** $s.PubInfo_{\psi.\rho[attr].\iota}.accusation \leftarrow clk$
5: **return** (IA, $\psi.\rho[attr].\iota$, $s.PubInfo_{\psi.\rho[attr].\iota}.accusation$)
6: **end if**
7: **return** $\perp$

B.4.4 Revoke Algorithm

PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Revoke (Algorithm 17). Revokes a valid certificate ψ prematurely. Returns a revoked-version ψ_r of ψ if it can be revoked or was already previously revoked, and $\perp$ otherwise. The algorithm first makes sure that the certificate is a valid certificate and that it is allowed to revoke it (line 1); in PoC-PKI, certificates can only be revoked by their issuers. Then, the algorithm checks if the certificate might have been already revoked. If this is the case, the revoked certificate is returned (line 2). If not, the algorithm revokes the certificate by adding to the certificate a signed revocation proof, and stores the revoked certificate in the local state (lines 3-8).

B.4.5 IsRevoked Algorithm

PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.IsRevoked (Algorithm 18). Checks whether a certificate was revoked or not. The algorithm is invoked over the issuer of the certificate, since if the certificate was revoked - the issuer of the certificate was the authority who revoked it. The algorithm first makes sure that the certificate is valid and that it was issued by the current executing authority (line 1). Then, it checks if there is a revoked version of this certificate in the local state, and if so, it returns the revoked certificate (line 2). Otherwise, the algorithm adds to the certificate

Algorithm 17 PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Revoke($\psi$)

// Verify that ψ was issued by the authority and ψ is a valid, not expired certificate
1: **if** $\psi.\rho[\mathsf{ACC}].\iota \neq s.\iota \ \vee$ PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.WasValid($\psi$) $\neq \top \ \vee \ \psi.ed < clk$ **then** return $\perp$
// If ψ was already revoked, return it
2: **if** $\exists \psi_r \in s.certs[\psi.id] \ s.t.$ PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.WasValid($\psi_r$, REV) $\wedge$ $Core(\psi) = Core(\psi_r)$ **then return** ψ_r
3: $\psi_r \leftarrow \psi$
// Revoke ψ
4: $data \leftarrow (Core(\psi), \mathsf{REV}, clk)$
5: $\sigma \leftarrow \mathcal{S}.\mathsf{Sign}(s.PrivInfo.sk, data)$
6: $\psi_r.\rho[\mathsf{REV}] \leftarrow (\sigma, s.\iota, clk)$
// Add ψ_r to the local state
7: $s.certs[\psi_r.id] \mathrel{+}= \psi_r$
8: **return** ψ_r

a signed proof that the certificate was not revoked until the current local time under the NREV attribute and outputs the certificate.

Algorithm 18 PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.IsRevoked($\psi$)

// Verify that ψ was issued by the authority and ψ is a valid, not expired certificate
1: **if** $\psi.\rho[\mathsf{ACC}].\iota \neq s.\iota \ \vee$ PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.WasValid($\psi$) $\neq \top \ \vee \ \psi.ed < clk$ **then** return $\perp$
// If ψ was already revoked, return it
2: **if** $\exists \psi_r \in s.certs[\psi.id] \ s.t.$ PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.WasValid($\psi_r$, REV) $\wedge$ $Core(\psi) = Core(\psi_r)$ **then** return ψ_r
3: ψ' $\leftarrow \psi$
// Add the non-revocation proof to ψ'
4: $data \leftarrow (Core(\psi), \mathsf{NREV}, clk)$
5: $\sigma = \mathcal{S}.\mathsf{Sign}(s.PrivInfo.sk, data)$
6: $\psi'.\rho[\mathsf{NREV}] \leftarrow (\sigma, s.\iota, clk)$
7: **return** ψ'

B.4.6 Upgrade Algorithm

PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Upgrade (Algorithm 19). Upgrades certificate ψ with attribute *attr*. The algorithm starts by making sure that the inputted certificate is a valid certificate (line 1). Then, the algorithm checks the local state whether an upgraded certificate with this attribute already exists. If so, the algorithm outputs the relevant certificate (line 2). Otherwise, the algorithm performs the upgrade based on the requested attribute. For transparency upgrades, the algorithm adds a relevant signed proof to the certificate and broadcasts the upgraded certificate (lines 4.1-4.1.4). For equivocation prevention upgrade, the algorithm generates a pending upgrade certificate by adding a signed proof to the certificate, and broadcasts the pending certificate (lines 4.2-4.2.4). The rest of the authorities

receive this pending certificate (using the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Incoming algorithm) and check whether they object to the upgrade request. If not, they send back a partial signature to the upgrading authority.

When the client returns with the pending certificate (line 4.3), the algorithm checks if the time for the upgrade process ($\psi.\rho[\text{EQ-P}].clk + \Delta$) has passed (line 4.3.1). If not, it means that the certificate is still pending, and the algorithm outputs the same pending certificate. If the time has expired, the algorithm *outputs* $\perp$*, since the upgrade failed*, along with the failure reason (line 4.3.2). Recall that line 2 of the algorithm checks whether an upgraded certificate exists in the local state. If the upgrade was successful, such certificate would have already exist in the local state (according to the implementation of the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Incoming algorithm). Therefore, since line 2 did not found such certificate, this means that the upgrade failed.

Algorithm 19 PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Upgrade($\psi, attr$)

```
    // Verify that ψ is a valid, not expired certificate
1:  if PoC-PKI^{S,E,TS}.WasValid(ψ) = ⊥ ∨ ψ.ed < clk then return ⊥
    // If there is already a matching pending or upgraded certificate, return it. A pending certificate
    is 'upgraded' to non-pending by the Incoming function - Upgrade does not need to do this.
2:  if ∃ψ' ∈ s.certs s.t. Core(ψ) = Core(ψ') ∧ PoC-PKI^{S,E,TS}.WasValid(ψ') then return ψ'
3:  ψ' ← ψ
4:  switch attr do
        // Transparency upgrade
5:      case ΔTRA ∨ ΔReTRA
            // Add the transparency proof to ψ'
6:          data ← (Core(ψ), attr, clk)
7:          σ = S.Sign(s.PrivInfo.sk, data)
8:          ψ'.ρ[attr] ← (σ, s.ι, clk)
9:          out ← {('send', m = [ψ'], j)}_{j∈N−s.ι}
        // Equivocation prevention upgrade for a non-pending certificate
10:     case (EQ-P ∧ PoC-PKI^{S,E,TS}.WasValid(ψ, EQ-P) = ⊥)
            // Add the PendEQ-P attr to ψ'
11:         data ← (Core(ψ), PendEQ-P, clk)
12:         σ = S.Sign(s.PrivInfo.sk, data)
13:         ψ'.ρ[EQ-P] ← (σ, s.ι, clk)
14:         out ← {('send', m = [ψ'], j)}_{j∈N−s.ι}
        // Non-equivocation upgrade for pending certificate
15:     case EQ-P ∧ PoC-PKI^{S,E,TS}.WasValid(ψ, EQ-P) = Pending
            // If upgrade time (Δ = 2) did not pass, return the pending certificate
16:         if clk < ψ.ρ[EQ-P].clk + Δ then return ψ
            // Upgrade time passed already yet still 'pending': failure
17:         return (⊥, s.certs[ψ.id].ρ[attr].failure)
    // Add the certificates to the local set of certificates
18: s.certs[ψ.id] += {ψ, ψ'}
19: return ψ'
```

B.4.7 WasValid Algorithm

PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.WasValid (see Algorithm 20). Checks if a certificate is valid, and optionally, whether it has an attribute *attr*. The algorithm first checks if the inputted certificate is a valid certificate (line 2). In PoC-PKI, a valid certificate is a non-expired certificate that was issued by an authority which is authorized to issue a certificate for the namespace that $\psi.id$ belongs to. In other words, the certificate must have a valid accountability (ACC) attribute. Hence, the algorithm verifies that authority that issued the accountability attribute is authorized using the Authorized algorithm (which is defined by the actual system that implements PoC-PKI) and that the proof of accountability is cryptographically valid. If there is no *attr* input (or $attr = \mathsf{ACC}$), then the algorithm outputs $\top$, since ψ is a valid certificate (line 3). Otherwise, the algorithm examines whether the certificate has the *attr* attribute.

The verification whether ψ has some specific attribute is done for each attribute accordingly. For the revocation accountability, Δ-transparency and Δ-revocation transparency attributes, the algorithm checks if ψ contains a relevant proof which is cryptographically valid (lines 5-5.1). The same check is performed for the non-revoked attribute (NREV), but in addition, the algorithm also ensures that the time in which the proof was issued comply with the *tms* value (lines 6-6.1). Finally, for the equivocation prevention attribute (EQ-P), the algorithm first check if the certificate is a pending certificate using the standard signature scheme (line 7.1). If not, the algorithm checks if ψ contains a valid threshold signature (line 7.2).

B.4.8 Incoming Algorithm

PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Incoming (see Algorithm 21). Handles all incoming messages. In PoC-PKI, there are three possible incoming messages: (1) a certificate broadcast, (2) a non-equivocation rejection, and (3) a non-equivocation approval. When a valid certificate arrives (line 1.1), it is added to the local state of certificates (line 1.1.1). If the arriving certificate is pending non-equivocation (line 1.1.2), the algorithm checks against the local state whether there is a conflicting certificate (line 1.1.3). Such conflicting certificate can be either an existing unequivocal certificate or a pending non-equivocation certificate. In case of a conflict, the algorithm prepares a response for the upgrading authority, to inform it about the conflicting certificate, i.e., about the upgrade request rejection (line 1.1.4). If there is no conflict, the algorithm prepares a partial signature approving the equivocation prevention upgrade (line 1.1.6). The actual response is sent in line 1.1.7. If the arriving message contains a conflicting certificate, i.e., upgrade rejection (line 1.2), store the conflicting certificate locally so it can be supplied to the client as an explanation why the upgrade failed (line 1.2.1). When the algorithm receives a partial signature (line 1.3), it stores the partial signature locally (line 1.3.1); when enough partial signature have arrived (line 1.3.2), the algorithm combines them using the threshold signature scheme's $\mathcal{TS}$.Combine algorithm (line 1.3.4). If the $\mathcal{TS}$.Combine algorithm was successful, the algorithm stores the upgraded certificate (line 1.3.5).

Algorithm 20 PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{WasValid}(\psi, pk, attr\ [, tms])$

$\chi \leftarrow \psi.\rho[\mathsf{ACC}]$

// Verify that $\chi.\iota$ (the issuer of ψ) is authorized to issue ψ and that the accountability proof is cryptographically valid. These are the basic validity requirements in PoC-PKI

if $\mathsf{Authorized}(\chi.\iota, \psi.id) \neq \top \ \vee \ tms < \psi.sd \ \vee \ tms > \psi.ed \ \vee$
$\mathcal{S}.\mathsf{Ver}(pk, (Core(\psi), \mathsf{ACC}, \chi.clk), \chi.\sigma) \neq \top$ **then**
  return $\bot$
end if

// If no attribute was supplied or the attribute is accountability, return true
if $attr = \bot \ \vee \ attr = \mathsf{ACC}$ **then** return $\top$

$\eta \leftarrow \psi.\rho[attr]$

// For the attributes that are implemented solely using proofs
if $attr \in \{\Delta\mathsf{TRA}, \Delta\mathsf{ReTRA}, \mathsf{REV}\}$ **then return** $\mathcal{S}.\mathsf{Ver}(pk, (Core(\psi), attr, \eta.clk), \eta.\sigma)$

// For the NREV attribute
if $attr = \mathsf{NREV}$ **then**
  // Check that the proof is cryptographically valid *and* that it is relevant to tms
  return $\mathcal{S}.\mathsf{Ver}(pk, (Core(\psi), attr, \eta.clk), \eta.\sigma) \ \wedge \ \psi.\rho[\mathsf{NREV}].clk \geq tms$
end if

// For the equivocation prevention attribute
if $attr = \mathsf{EQ\text{-}P}$ **then**
  // Check if ψ is pending
  if $\mathcal{S}.\mathsf{Ver}(pk, (Core(\psi), Pend\mathsf{EQ\text{-}P}, \eta.clk), \eta.\sigma)$ **then** return $Pending$
  // Certificate is not pending, check if the group proof is cryptographically valid
  return $\mathcal{TS}.\mathsf{Ver}(pk, (Core(\psi), \mathsf{EQ\text{-}P}), \eta.\sigma)$
end if
return $\bot$

Algorithm 21 PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .Incoming()

1: **for each** $m \in inp$ **do**

// If m is part of the secure initialization phase

2: **if** $s.initCounter < 3$ **then** PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .Init(m)

// If m is a broadcasted certificate

3: **if** $\psi \leftarrow m$ **s.t.** PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .WasValid(ψ) **then**

4: $s.certs[\psi.id] \mathrel{+}= \psi$

// If ψ is pending equivocation prevention

5: **if** PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .WasValid(ψ, EQ-P) $= Pending$ **then**

// If a conflicting certificate is known, return it to abort

6: **if** $\exists \psi' \in s.certs$ *s.t.* $\psi.id = \psi'.id \wedge$

$Core(\psi) \neq Core(\psi') \wedge$ PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .WasValid(ψ', EQ-P) $\neq \perp$ **then**

// Prepare a rejection response with the conflicting certificates

7: $res \leftarrow (\psi, \psi')$

// No conflicting certificate - approve the request for non-equivocation

8: **else**

// Prepare a partial signature approving the upgrade

9: $res \leftarrow (\psi, \sigma = \mathcal{TS}.\mathsf{Sign}(s.PrivInfo.tsk, (Core(\psi), \text{EQ-P})))$

10: **end if**

11: $out \leftarrow$ ('send', $res, \psi.\rho$[EQ-P].ι)

12: **end if**

// If m is an upgrade rejection

13: **else if** $(\psi, \psi') \leftarrow m$ *s.t.* PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .WasValid(ψ, EQ-P) $=$ $Pending \wedge$ PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .WasValid(ψ', EQ-P) $\neq \perp$ **then**

// Store the conflicting certificate

14: $s.certs[\psi.id].\rho$[EQ-P].$failure = \psi'$

// If m is a partial approval of a pending certificate which has not been rejected yet

15: **else if** $(\psi, \sigma) \leftarrow m$ *s.t.* PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .WasValid(ψ, EQ-P) $= Pending \wedge$ $s.certs[\psi.id].\rho$[EQ-P].$failure = \perp$ **then**

// Add new partial signature to the local state

16: $s.toUpgrade[\psi.id] \mathrel{+}= \sigma$

// Check to see if enough semi-signatures arrived

17: **if** $|s.toUpgrade[\psi.id]| < |\mathsf{N}| - \lfloor |\mathsf{N}|/3 \rfloor$ **then** continue

// Enough semi-proofs have arrived - try to combine them

18: $\psi' \leftarrow \psi$

19: $\psi'.\rho$[NEQ].$\sigma \leftarrow \mathcal{TS}.\mathsf{Combine}(s.toUpgrade[\psi.id])$

// Check if the upgrade was successful

20: **if** PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .WasValid(ψ', EQ-P) **then** $s.certs[\psi'.id] \mathrel{+}= \psi'$

21: **end if**

22: **end for**

The PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Time algorithm is not required in PoC-PKI, since in PoC-PKI there are no time-based events.

C Analysis of PoC-PKI

In this section, we provide reduction-based proofs showing that PoC-PKI achieves its safety properties, as described in §5.1. We first show that PoC-PKI achieves accountability, $\Delta_{\text{PoC-PKI}}$-transparency, revocation accountability, non-revocation accountability and $\Delta_{\text{PoC-PKI}}$-revocation transparency by reduction to the existential unforgeability of a secure signature scheme. We then show that PoC-PKI also achieves equivocation prevention by reduction to the $g-$existential unforgeability of a secure threshold signature scheme.

C.1 Proofs of Accountability, Revocation Accountability, Non-Revocation Accountability, $\Delta_{\text{PoC-PKI}}$-transparency and $\Delta_{\text{PoC-PKI}}$-revocation transparency

Proof Methodology. To prove that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves the attributes that are implemented using the secure signature scheme $\mathcal{S}$, we use the following methodology:

1. We first define a variation of PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ called $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ (see Def. 20), where a PPT oracle algorithm $OSign(sk,\cdot)$ is used to generate signatures using a secret key sk *instead* of entity $\iota \in \mathsf{N}$, where sk is the matching secret signing key of the verification key vk, see §C.1.1.
2. Then, we define a game called $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$, where we execute an adversary $\mathcal{A}$ with the $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ scheme, and ask $\mathcal{A}$ to output a message m and signature σ over m, signed using the public verification key vk, namely, without $\mathcal{A}$ knowing the matching signing key sk, nor $\mathcal{A}$ can use the oracle access to sign m, see §C.1.2.
3. We then formulate Lemma 1, showing that the existence of an adversary that 'wins' the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability means that $\mathcal{S}$ is not a secure signature scheme, see §C.1.3.
4. We then prove that if PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not achieves accountability, revocation accountability, $\Delta_{\text{PoC-PKI}}$-transparency or $\Delta_{\text{PoC-PKI}}$-revocation transparency, then we can construct an adversary that wins the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability, see §C.1.4.
5. Finally, we revisit Theorem 2 and complete its proof by combining steps $1-4$, see §C.1.5.

C.1.1 The $\overline{\text{PoC-PKI}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ scheme

We begin by defining a variation of the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ scheme, denoted as $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$; for brevity, where the identities of $\mathcal{S}$, $\mathcal{E}$ and $\mathcal{TS}$ are clear or irrelevant, we may use the shorthand $\overline{\text{PoC-PKI}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$.

Definition 20. *Let $\mathcal{S}$, $\mathcal{E}$ and $\mathcal{TS}$ be a signature, encryption and threshold-signature schemes, respectively, and let $(sk, vk) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$, for a given security parameter 1^κ. Given a PPT oracle $OSign(sk, \cdot)$, let $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ (abbreviated as $\overline{\text{PoC-PKI}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$) be a PKI scheme where one designated authority $\iota \in \mathsf{N}$ executes the $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ scheme with the following changes, and the rest of the authorities in N execute $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ without any changes:*

1. *The $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}.\mathsf{Init}$ algorithm is the same as $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Init}$ (Alg. 14), except for replacing line 2 of $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Init}$ with the following line:*
$$(s.PrivInfo.sk, s.PubInfo_{s.\iota}.vk) \leftarrow (nil, vk)$$
where vk is the public verification key of the sk signing key, given as input to $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$.
2. *In all the algorithms that use the signing key,* replace *the following code:*
$$\mathcal{S}.\mathsf{Sign}(s.PrivInfo.sk, data)$$
with the following code:
$$OSign(sk, data)$$
namely, sign data using the oracle access to the sign operation $\mathcal{S}.\mathsf{Sign}$.

C.1.2 The $\overline{\mathbf{Exp}}^{\mathit{Forge},\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ Game

We now define the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N})$ game:

1. Generate key pair $(sk, vk) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$.
2. Randomly choose an authority $\iota \xleftarrow{R} \mathsf{N}$.
3. Execute $\mathcal{A}$ with the $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ scheme, i.e.,
$$(\mathsf{N_F}, R, t, out_{\mathcal{A}}) \leftarrow \mathbf{Exec}_{\mathcal{A},\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}}(1^\kappa, \mathsf{N})$$
4. $\mathcal{A}$ outputs message m and signature σ, i.e., $(m, \sigma) \leftarrow out_{\mathcal{A}}$.
5. The experiment outputs 1 if:
   (a) $\mathcal{S}.\mathsf{Ver}(vk, m, \sigma) = \top$
   (b) $\mathcal{A}$ did not use the oracle access on m.
   (c) $\mathcal{A}$ satisfies model $\mathcal{M}$ (see Def. 1).
   (d) ι is an honest authority, i.e., $\iota \in \mathsf{N} - \mathsf{N_F}$

   Otherwise, the experiment outputs 0.

C.1.3 The Relation Between $\overline{\mathbf{Exp}}^{\mathit{Forge},\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ and the Security of the Signature Scheme $\mathcal{S}$

We now show that the existence of an adversary that 'wins' the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability means that $\mathcal{S}$ is not a secure signature scheme.

Lemma 1. *If there is a PPT adversary $\mathcal{A}$ that satisfies*

$$Pr\left[\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N})=1\right]\notin Negl(1^\kappa) \tag{8}$$

then $\mathcal{S}$ is not a secure signature scheme.

Proof. Assume to the contrary that such adversary $\mathcal{A}$ exists, yet $\mathcal{S}$ is a secure signature scheme.

Following §C.1.2, if $\mathcal{A}$ 'wins' the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability, then this means that $\mathcal{A}$ can output a message m and a valid signature σ over m, where $\mathcal{A}$ has only access to the verification key and oracle accesses to the signing key, without requesting the oracle to sign m.

Therefore, according to definition of existential unforgeability (Def. 8), the following holds for $\mathcal{A}$

$$Pr\left[\mathbf{Exp}^{EU}_{\mathcal{A},\mathcal{S}}(1^\kappa)=1\right]\notin Negl(1^\kappa) \tag{9}$$

thus contradicting the security of $\mathcal{S}$. □

C.1.4 Linking Accountability, Revocation Accountability, $\Delta_{\text{PoC-PKI}}$-transparency and $\Delta_{\text{PoC-PKI}}$-revocation transparency to the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}}$ Game

We now show that if PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures accountability under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then we can construct an adversary that wins in the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game.

Claim 1. If PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures accountability under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\mathsf{ACC}}$ such that

$$Pr\left[\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A}_{\mathsf{ACC}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N})=1\right]\notin Negl(1^\kappa) \tag{10}$$

Proof. From Definition 2, if PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures accountability under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\mathsf{ACC}}$ that satisfies

$$Pr\left[\begin{array}{l}(\mathsf{N_F},R,t,out_{\mathcal{A}_{\mathsf{ACC}}})\leftarrow\mathbf{Exec}_{\mathcal{A}_{\mathsf{ACC}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N})\\ \mathbf{Exp}^{\mathsf{ACC}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N},\mathsf{N_F},R,t,out_{\mathcal{A}_{\mathsf{ACC}}})=\top\end{array}\right]\notin Negl(1^\kappa) \tag{11}$$

Therefore, all that is left is to show that if Eq. 11 holds then Eq. 10 also holds.

First, according to the description of the security experiment $\mathbf{Exp}^{\mathsf{ACC}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ (Alg. 2), the return value of the experiment is TRUE only if, among other criteria, the following is TRUE:

$$\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{WasValid}(\psi, \mathsf{ACC}) \tag{12}$$

for a certificate ψ outputted by the adversary, where $\psi.\rho[\mathsf{ACC}].\iota$ (the issuer of the certificate), is an honest authority that did not issue ψ by executing the $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Issue}$ algorithm.

Second, according to the implementation of $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{WasValid}$, as described in Alg. 20, the algorithm executes

$$\mathcal{S}.\mathsf{Ver}(s.PubInfo_{\eta.\iota}.vk, (Core(\psi), \mathsf{ACC}, \eta.clk), \eta.\sigma) \tag{13}$$

for $\eta = \psi.\rho[\mathsf{ACC}]$.

Lastly, the only place in $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ where an honest authority ι computes its keys is in the $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Init}$ algorithm (Algorithm 14); specifically the sign/verify key pair is generated in line 2, using the $\mathcal{S}.\mathsf{Gen}$ algorithm. Furthermore, the signing key is only used in algorithms: $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Issue}$, $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Revoke}$, $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{IsRevoked}$ and $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Upgrade}$, and only with the $\mathcal{S}.\mathsf{Sign}$ algorithm; however, certificates can only be issued in PoC-PKI using the $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Issue}$ algorithm.

Thus, following Eq. 11, the value described in Eq. 12 must be TRUE, and as a result, Eq. 13 must also equal TRUE. Accordingly, with accordance to PoC-PKI's implementation, adversary $\mathcal{A}_{\mathsf{ACC}}$ is a PPT adversary that for a message $m = (Core(\psi), \mathsf{ACC}, \eta.clk)$ was able to generate a signature $\sigma = \eta.\sigma$ that is validated with non-negligible probability with the verification key $vk = s.PubInfo_{\eta.\iota}.vk$, without access to the signing key, and without ever having the honest authority ι sign m. Hence, such $\mathcal{A}_{\mathsf{ACC}}$ adversary satisfies Eq. 10. □

We now show that if $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures revocation accountability under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then we can construct an adversary that wins in the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game.

Claim 2. If PoC-PKI does not achieves revocation accountability under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\mathsf{REV}}$ such that

$$Pr\left[\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A}_{\mathsf{REV}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N}) = 1\right] \notin Negl(1^\kappa) \tag{14}$$

Proof. From Definition 2, if PoC-PKI does not ensures revocation accountability under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\mathsf{REV}}$ that satisfies

$$Pr\left[\begin{matrix}(\mathsf{N_F}, R, t, out_{\mathcal{A}_{\mathsf{REV}}}) \leftarrow \mathbf{Exec}_{\mathcal{A}_{\mathsf{REV}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N}) \\ \mathbf{Exp}^{\mathsf{REV}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}_{\mathsf{REV}}}) = \top\end{matrix}\right] \notin Negl(1^\kappa) \tag{15}$$

Therefore, all that is left is to show that if Eq 15 holds then Eq 14 also holds.

First, according to the description of the security experiment $\mathbf{Exp}^{\mathsf{REV}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ (Alg. 3), the return value of the experiment is TRUE only if, among other criteria, the following is TRUE:

$$\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \tag{16}$$

for a certificate ψ_r outputted by the adversary, where $\psi_r.\rho[\mathsf{REV}].\iota$ (the entity that revoked the certificate), is an honest authority that did not revoke ψ_r by executing the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Revoke algorithm.

Second, according to the implementation of PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.WasValid, as described in Alg. 20, the algorithm executes

$$\mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota}, (Core(\psi_r), \mathsf{REV}, \eta.clk), \eta.\sigma) \tag{17}$$

for $\eta = \psi_r.\rho[\mathsf{REV}]$.

Lastly, the only place in PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ where an honest authority ι computes its keys is in the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Init algorithm (Algorithm 14); specifically the sign/verify key pair is generated in line 2, using the $\mathcal{S}$.Gen algorithm. Furthermore, the signing key is only used in algorithms: PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Issue, PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Revoke, PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.IsRevoked and PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Upgrade, and only with the $\mathcal{S}$.Sign algorithm; however, certificates can only be have the non-revocation accountability attribute in PoC-PKI using the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.IsRevoked algorithm.

Thus, following Eq. 19, the value described in Eq. 20 must be TRUE, and as a result, Eq. 21 must also equal TRUE. Accordingly, with accordance to PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$'s implementation, adversary $\mathcal{A}_{\mathsf{NREV}}$ is a PPT adversary that for a message $m = (Core(\psi), \mathsf{NREV}, \eta.clk)$ was able to generate a signature $\sigma = \eta.\sigma$ that is validated with non-negligible probability with the verification key $vk = s.PubInfo_{\eta.\iota}.vk$, without access to the signing key, and without ever having the honest authority $\psi.\rho[\mathsf{NREV}].\iota$ sign m. Hence, such $\mathcal{A}_{\mathsf{NREV}}$ adversary satisfies Eq. 18. □

We now show that if PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures non-revocation accountability under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then we can construct an adversary that wins in the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game.

Claim 3. If PoC-PKI does not achieves non-revocation accountability under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\mathsf{NREV}}$ such that

$$Pr\left[\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A}_{\mathsf{NREV}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N}) = 1\right] \notin Negl(1^\kappa) \tag{18}$$

Proof. From Definition 2, if PoC-PKI does not ensures non-revocation accountability under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\mathsf{NREV}}$ that satisfies

$$Pr\left[\begin{array}{l}(\mathsf{N_F}, R, t, out_{\mathcal{A}_{\mathsf{NREV}}}) \leftarrow \mathbf{Exec}_{\mathcal{A}_{\mathsf{NREV}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N}) \\ \mathbf{Exp}^{\mathsf{NREV}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}_{\mathsf{NREV}}}) = \top\end{array}\right] \notin Negl(1^\kappa) \tag{19}$$

Therefore, all that is left is to show that if Eq 19 holds then Eq 18 also holds.

First, according to the description of the security experiment $\mathbf{Exp}^{\mathsf{NREV}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ (Alg. 4), the return value of the experiment is TRUE only if, among other criteria, the following is TRUE:

$$\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{WasValid}(\psi, \mathsf{NREV}) \tag{20}$$

for a certificate ψ outputted by the adversary, where $\psi.\rho[\mathsf{NREV}].\iota$ (the entity that issued the certificate's non-revocation), is an honest authority.

Second, according to the implementation of $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{WasValid}$, as described in Alg. 20, the algorithm executes

$$\mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota}, (Core(\psi_r), \mathsf{NREV}, \eta.clk), \eta.\sigma) \tag{21}$$

for $\eta = \psi.\rho[\mathsf{NREV}]$.

Lastly, the only place in $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ where an honest authority ι computes its keys is in the $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Init}$ algorithm (Algorithm 14); specifically the sign/verify key pair is generated in line 2, using the $\mathcal{S}.\mathsf{Gen}$ algorithm. Furthermore, the signing key is only used in algorithms: $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Issue}$, $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Revoke}$, $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{IsRevoked}$ and $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Upgrade}$, and only with the $\mathcal{S}.\mathsf{Sign}$ algorithm; however, certificates can only be revoked in PoC-PKI using the $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Revoke}$ algorithm.

Thus, following Eq. 15, the value described in Eq. 16 must be TRUE, and as a result, Eq. 17 must also equal TRUE. Accordingly, with accordance to $\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}$'s implementation, adversary $\mathcal{A}_{\mathsf{REV}}$ is a PPT adversary that for a message $m = (Core(\psi_r), \mathsf{REV}, \eta.clk)$ was able to generate a signature $\sigma = \eta.\sigma$ that is validated with non-negligible probability with the verification key $vk = s.PubInfo_{\eta.\iota}.vk$, without access to the signing key, and without ever having the honest authority ι sign m. Hence, such $\mathcal{A}_{\mathsf{REV}}$ adversary satisfies Eq. 14. □

We now show that if PoC-PKI does not ensures $\Delta_{\text{PoC-PKI}}$-transparency under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then we can construct an adversary that wins in the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game.

Claim 4. If PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures $\Delta_{\text{PoC-PKI}}$-transparency under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\Delta\mathsf{TRA}}$ such that

$$Pr\left[\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A}_{\Delta\mathsf{TRA}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N})=1\right]\notin Negl(1^\kappa) \tag{22}$$

Proof. From Definition 2, if PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures $\Delta_{\text{PoC-PKI}}$-transparency under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\Delta\mathsf{TRA}}$ that satisfies

$$Pr\left[\begin{matrix}(\mathsf{N_F},R,t,out_{\mathcal{A}_{\Delta\mathsf{TRA}}})\leftarrow\mathbf{Exec}_{\mathcal{A}_{\Delta\mathsf{TRA}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N})\\ \mathbf{Exp}^{\Delta\mathsf{TRA}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N},\mathsf{N_F},R,t,out_{\mathcal{A}_{\Delta\mathsf{TRA}}})=\top\end{matrix}\right]\notin Negl(1^\kappa) \tag{23}$$

Therefore, all that is left is to show that if Eq 23 holds then Eq 22 also holds.

First, according to the description of the security experiment $\mathbf{Exp}^{\Delta\mathsf{TRA}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ (Alg. 5), the return value of the experiment is TRUE only if, among other criteria, the following is TRUE:

$$\begin{aligned}\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{WasValid}(\psi,\Delta\mathsf{TRA})\wedge\\ R.out[t],R.out[t\text{-}1]\neq\text{IA}\end{aligned} \tag{24}$$

for a certificate ψ outputted by the adversary, and there exists two honest authorities that are not aware of ψ although they should, and they cannot indicate any problem with ψ nor with the authority that issued $\Delta\mathsf{TRA}$ for ψ.

Second, according to the implementation of PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{WasValid}$, as described in Alg. 20, the algorithm executes

$$\mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota},(Core(\psi),\Delta\mathsf{TRA},\eta.clk),\eta.\sigma) \tag{25}$$

for $\eta=\psi_r.\rho[\Delta\mathsf{TRA}]$.

Third, the only place in PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ where an honest authority ι computes its keys is in the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Init}$ algorithm (Algorithm 14); specifically the sign/verify key pair is generated in line 2, using the $\mathcal{S}.\mathsf{Gen}$ algorithm. Furthermore, the signing key is only used in algorithms: PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Issue}$, PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Revoke}$, PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{IsRevoked}$ and PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Upgrade}$, and only with the $\mathcal{S}.\mathsf{Sign}$ algorithm; however, certificates can only be upgraded with the $\Delta\mathsf{TRA}$ attribute in PoC-PKI using the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Upgrade}$ algorithm. We emphasize that an honest authority that upgrades a $\Delta_{\text{PoC-PKI}}$-transparent certificate ψ using the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Upgrade}$ algorithm, informs all the other authorities about ψ.

Lastly, according to PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Audit}$ (Alg. 16), if an honest authority receives a cryptographically valid $\Delta_{\text{PoC-PKI}}$-transparent certificate that it is not aware of although it should, it outputs an IA (Indicator of Accusation).

Thus, following Eq. 23, the value described in Eq. 24 must be TRUE, and as a result, Eq. 25 must also equal TRUE. However, if Eq. 24 is TRUE, this means

that the authority that endorsed $\Delta\mathsf{TRA}$ for ψ was honest, because otherwise, the honest authorities in Eq. 25 would have outputted an IA. Accordingly, with accordance to PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$'s implementation, adversary $\mathcal{A}_{\Delta\mathsf{TRA}}$ is a PPT adversary that for a message $m = (\mathit{Core}(\psi), \Delta\mathsf{TRA}, \eta.\mathit{clk})$ was able to generate a signature $\sigma = \eta.\sigma$ that is validated with non-negligible probability with the verification key $vk = s.\mathit{PubInfo}_{\eta.\iota}.vk$, without access to the signing key, and without ever having the honest authority $\psi.\rho[\Delta\mathsf{TRA}].\iota$ sign m. Hence, such $\mathcal{A}_{\Delta\mathsf{TRA}}$ adversary satisfies Eq. 22.

□

We now show that if PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures $\Delta_{\text{PoC-PKI}}$-revocation transparency under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then we can construct an adversary that wins in the $\overline{\mathbf{Exp}}^{\mathit{Forge},\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game.

Claim 5. If PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures $\Delta_{\text{PoC-PKI}}$-revocation transparency under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\Delta\mathsf{ReTRA}}$ such that

$$Pr\left[\overline{\mathbf{Exp}}^{\mathit{Forge},\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A}_{\Delta\mathsf{ReTRA}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N}) = 1\right] \notin \mathit{Negl}(1^\kappa) \tag{26}$$

Proof. From Definition 2, if PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not ensures $\Delta_{\text{PoC-PKI}}$-revocation transparency under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\Delta\mathsf{ReTRA}}$ that satisfies

$$Pr\left[\begin{array}{l}(\mathsf{N_F}, R, t, \mathit{out}_{\mathcal{A}_{\Delta\mathsf{ReTRA}}}) \leftarrow \mathbf{Exec}_{\mathcal{A}_{\Delta\mathsf{ReTRA}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N}) \\ \mathbf{Exp}^{\Delta\mathsf{ReTRA}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N},\mathsf{N_F},R,t,\mathit{out}_{\mathcal{A}_{\Delta\mathsf{ReTRA}}}) = \top\end{array}\right] \notin \mathit{Negl}(1^\kappa) \tag{27}$$

Therefore, all that is left is to show that if Eq 27 holds then Eq 26 also holds.

First, according to the description of the security experiment $\mathbf{Exp}^{\Delta\mathsf{ReTRA}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ (Alg. 6), the return value of the experiment is TRUE only if, among other criteria, the following is TRUE:

$$\begin{aligned}\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{WasValid}(\psi, \Delta\mathsf{ReTRA}) \wedge \\ \text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \wedge \\ \mathit{Core}(\psi) = \mathit{Core}(\psi_r) \wedge \\ \psi_r.\rho[\mathsf{REV}].\iota \in \mathsf{N} - \mathsf{N_F}\end{aligned} \tag{28}$$

for certificates ψ,ψ_r outputted by the adversary, where ψ_r is the revoked version of the $\Delta_{\text{PoC-PKI}}$-revocation transparent certificate ψ, which was revoked by an honest authority, without two honest entities being aware of ψ_r, although they should be.

Second, according to the implementation of PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{WasValid}$, as described in Alg. 20, the algorithm executes

$$\mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota}, (Core(\psi), \Delta\mathsf{ReTRA}, \eta.clk), \eta.\sigma) \tag{29}$$

for $\eta = \psi_r.\rho[\Delta\mathsf{ReTRA}]$.

Third, the only place in PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ where an honest authority ι computes its keys is in the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Init}$ algorithm (Algorithm 14); specifically the sign/verify key pair is generated in line 2, using the $\mathcal{S}.\mathsf{Gen}$ algorithm. Furthermore, the signing key is only used in algorithms: PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Issue}$, PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Revoke}$, PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{IsRevoked}$ and PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Upgrade}$, and only with the $\mathcal{S}.\mathsf{Sign}$ algorithm; however, the only place to revoke a certificate is via the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Revoke}$ algorithm and the only way to upgrade a certificate into a $\Delta_{\text{PoC-PKI}}$-revocation transparent certificate is via the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Upgrade}$ algorithm.

Lastly, when the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Revoke}$ algorithm revokes a $\Delta_{\text{PoC-PKI}}$-revocation transparent certificate, it notifies the rest of the authorities, and when the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Upgrade}$ algorithm upgrades a revoked certificate into $\Delta_{\text{PoC-PKI}}$-revocation transparent certificate it notifies the rest of the authorities.

Thus, following Eq. 27, the value described in Eq. 28 must be TRUE, and as a result, Eq. 29 must also equal TRUE. Accordingly, with accordance to PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$'s implementation, adversary $\mathcal{A}_{\Delta\mathsf{ReTRA}}$ is a PPT adversary that for a message $m = (Core(\psi), \Delta\mathsf{ReTRA}, \eta.clk)$ was able to generate a signature $\sigma = \eta.\sigma$ that is validated with non-negligible probability with the verification key $vk = s.PubInfo_{\eta.\iota}.vk$, without access to the signing key, and without ever having the honest authority ι sign m. Hence, such $\mathcal{A}_{\Delta\mathsf{ReTRA}}$ adversary satisfies Eq. 26.

□

C.1.5 Completing the Proof

Now, we revisit Theorem 2 presented in §5.1, and complete its proof.

Theorem 2. *Let $\mathcal{S}$ be an existentially-unforgeable signature scheme. Then* PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves accountability, revocation accountability, non-revocation accountability, $\Delta_{\text{PoC-PKI}}$-transparency, and $\Delta_{\text{PoC-PKI}}$-revocation transparency under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$.*

Proof. The proof for all four properties is essentially identical; we present the argument for *accountability* and later discuss the (trivial) adaptions for the other three properties.

Assume, therefore, that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not achieve *accountability*, and we will show that this implies that $\mathcal{S}$ is not a secure signature scheme. According to Claim 1, this means there exists a PPT adversary $\mathcal{A}$ that wins the $\overline{\text{---}}\mathbf{Exp}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability. Note that Claim 1 refers to this adversary as $\mathcal{A}_{\mathsf{ACC}}$; the argument follows by substituting $\mathcal{A}$ in Eq. (10).

Similarly, from Claims 2-5, if PoC-PKI does not achieve *revocation accountability, $\Delta_{\text{PoC-PKI}}$-transparency, or $\Delta_{\text{PoC-PKI}}$-revocation transparency*, then there a PPT adversary that wins the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability.

However, Lemma 1 shows that if there exists a PPT adversary $\mathcal{A}$ that wins the $\overline{\mathbf{Exp}}^{Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability, then $\mathcal{S}$ is not a secure signature scheme. □

C.2 Proof of Non-Equivocation

Proving that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves equivocation prevention is different from proving the other properties, because PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ prevents equivocation using both a secure encryption scheme $\mathcal{E}$ and a robust threshold signature scheme $\mathcal{TS}$. This requires a few adjustments to the proof methodology, as we now discuss.

Proof methodology. To prove that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves equivocation prevention, we use the following methodology:

1. We define a variation of PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ called $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ where equivocation prevention relies solely on the threshold scheme $\mathcal{TS}$, rather than also relying on a secure encryption scheme (Def. 21), see §C.2.1.
2. Then, we show that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ is secure against conflicting certificates, see §C.2.2.
3. We then define the $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game in §C.2.3 and link the security of a threshold signature scheme $\mathcal{TS}$ with the $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game in §C.2.4.
4. We show that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ ensures equivocation prevention under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ in §C.2.5
5. Finally, we show that the security argument also holds for the original PoC-PKI scheme, see Theorem 3 in §C.2.6.

Rationale behind the proof methodology. The rationale behind this methodology can be viewed as a 'divide and conquer' approach that allows us to present the proof in a simplified manner. Since both encryption and threshold signature schemes are used to achieve equivocation prevention, the aforementioned proof methodology separates the two by defining the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ scheme, where encryption is not used. We can then prove that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ ensures equivocation prevention, by reduction to the security of the threshold signature scheme; since $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ does not use encryption, this resembles the proof of properties that rely on standard signature schemes, e.g., accountability, as described in Theorem 2, except that here we reduce to the security of threshold signatures rather than to the security of a standard signature scheme. The fact that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ only slightly varies from PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$, allows us to prove that if we add the

encryption back to the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ scheme, thus ending up the with the original PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ scheme, the security argument that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves equivocation prevention still holds, as long as the encryption scheme is secure.

C.2.1 The $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ Scheme

We start by defining a variation of the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$, called $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$.

Definition 21. *Let $\mathcal{S}$,$\mathcal{E}$ and $\mathcal{TS}$ be a signature, encryption and threshold-signature schemes, respectively. Given a PPT oracle $OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)$, let $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ (abbreviated as $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}$) be a PKI scheme identical to* PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$*, except for the following changes:*

1. *In the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$.*Init *(Alg. 14), replace lines 10-12 with the following code:*

$$out \leftarrow \{(\text{`}send\text{'}, m = [(tvk, \mathcal{E}.\mathsf{Enc}(PubInfo_i.ek, \text{`}0\text{'}))], j)\}_{i\in S.\mathsf{N}-s.\iota} \tag{30}$$

2. *In the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$.*Incoming *algorithm, replace the following line of code:*

$$res \leftarrow (\psi, \sigma = \mathcal{TS}.\mathsf{Sign}(S.PrivInfo.tsk, (Core(\psi), \mathsf{EQ\text{-}P}))) \tag{31}$$

with the following line of code:

$$res \leftarrow (\psi, \sigma = OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(s.\iota, (Core(\psi), \mathsf{EQ\text{-}P}))) \tag{32}$$

*namely, generate proof σ by signing data using the oracle access to the sign operation $\mathcal{TS}$.*Sign*.*

Note the two modifications that happen in $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ as opposed to PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$. First, instead of delivering each authority $\iota \in \mathsf{N}$ its matching share of the threshold signature scheme tsk_ι, the authorities receive a 'useless' string ('0'). Second, instead of using the individual signing algorithm $\mathcal{TS}$.Sign in the Incoming algorithm, the scheme uses the oracle $OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)$. These two modifications essentially eliminate the part that the encryption scheme $\mathcal{E}$ plays in equivocation prevention, hence, in $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$, equivocation prevention is implemented solely using the threshold signature scheme $\mathcal{TS}$.

C.2.2 Proving that $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ is Secure Against Equivocation

We now show that $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ is secure against conflicting (equivocating) certificates, i.e., honest authorities would not sign conflicting certificates. Namely, when an honest authority is aware of a valid certificate ψ with the EQ-P attribute, it will not partially sign any other certificate ψ' with the same identifier ($\psi.id = \psi'.id$) that its validity period overlaps with the validity period of ψ, since these two certificates are in conflict.

Claim 6. Let N be a set of entities and let f be the number of compromised entities in N. Let $\mathcal{TS}$ be a (t,n)-threshold-signature scheme where $n = |\mathsf{N}| > 2f$ as the number of shares, and the threshold t is defined as $t = |\mathsf{N}| - f - 1$. If the $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ scheme uses $\mathcal{TS}$, then no PPT adversary can *abuse* $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ to generate two conflicting certificates ψ,ψ' with the non-equivocation attribute.

Proof. The only place in $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ where authorities generate partial signatures is in line 9 of the $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$.Incoming algorithm (Alg. 21), where an entity generates a share for the signature proof of a non-equivocal certificate ψ. However, this line is executed only if the check in line 6 is satisfied, i.e., there is no conflicting certificate ψ' in the *s.certs* repository. In other words, if line 6 is satisfied, it ensures that there is no certificate ψ' (valid or pending) in *s.certs* with the same identifier but different public information that has the EQ-P attribute. Therefore, each honest authority would only execute line 9, i.e., generate their partial group-signature, for *either* ψ *or* ψ', but never *for both.*

Let n_ψ ($n_{\psi'}$) denote the number of honest authorities partially-signing ψ (resp., ψ'). Then:

$$n_\psi + n_{\psi'} \leq |\mathsf{N}| - f \tag{33}$$

Assume, without loss of generality, that $n_\psi \geq t + 1 = |\mathsf{N}| - f$, i.e., there are enough signature-shares from honest authorities to combine into a valid certificate ψ with the EQ-P attribute. Following Eq. 33:

$$n_{\psi'} \leq |\mathsf{N}| - f - n_\psi = 0 \tag{34}$$

hence, the total number of shares of signatures for ψ' is at most f (i.e., only from the malicious authorities). Since following Def. 11, at least $t + 1$ partial signatures are required to be combined into a valid group signature, and since $t+1 = |\mathsf{N}| - f > 2f - f = f$, then f is not enough partial signatures to combine into a valid non-equivocal certificate upgrade for ψ'. □

C.2.3 The $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ Game

We now define the $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N})$ game:

1. Generate keys $(tvk, \{tsk_i\}_{i\in\mathsf{N}}) \leftarrow \mathcal{TS}.\mathsf{Gen}(1^\kappa, |\mathsf{N}|, \lfloor|\mathsf{N}|/3\rfloor)$.
2. Execute $\mathcal{A}$ with the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ scheme, i.e.,

$$(\mathsf{N_F}, R, t, out_{\mathcal{A}}) \leftarrow \mathbf{Exec}_{\mathcal{A},\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}}(1^\kappa, \mathsf{N})$$

3. $\mathcal{A}$ outputs message m and signature σ, i.e., $(m, \sigma) \leftarrow out_{\mathcal{A}}$.
4. The experiment outputs 1 if:
   (a) $\mathcal{TS}.\mathsf{Ver}(tvk, m, \sigma) = \top$
   (b) $\mathcal{A}$ did not perform more than $\lfloor|\mathsf{N}|/3\rfloor$ oracle requests for message m.
   (c) $\mathcal{A}$ satisfies model $\mathcal{M}$ (see Def. 1).
   Otherwise, the experiment outputs 0.

C.2.4 The Relation Between $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ and the Security of the Threshold Signature Scheme $\mathcal{TS}$

We now show that the existence of an adversary that 'wins' the $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability means that $\mathcal{TS}$ is not a secure threshold signature scheme.

Lemma 2. *If there is a PPT adversary $\mathcal{A}$ that satisfies*

$$Pr\left[\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N}) = 1\right] \notin Negl(1^\kappa) \tag{35}$$

then $\mathcal{TS}$ is not a $g-$existentially unforgeable threshold signature scheme.

Proof. Assume to the contrary that such adversary $\mathcal{A}$ exists, yet $\mathcal{TS}$ is a $g-$existentially unforgeable threshold signature scheme.

Following §C.2.3, if $\mathcal{A}$ 'wins' the $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability, then this means that $\mathcal{A}$ can output a message m and a valid signature σ over m, where $\mathcal{A}$ has only access to the verification key and no more than $\lfloor|\mathsf{N}|/3\rfloor$ oracle requests for message m.

Therefore, according to the definition of $g-$existential unforgeability (Def. 11), the following holds for $\mathcal{A}$

$$Pr\left[\mathbf{Exp}^{EU}_{\mathcal{TS},\mathcal{A}}(1^\kappa, |\mathsf{N}|, \lfloor|\mathsf{N}|/3\rfloor) = 1\right] \notin Negl(1^\kappa) \tag{36}$$

thus contradicting the $g-$existential unforgeability of $\mathcal{TS}$. □

C.2.5 $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ Ensures Equivocation Prevention

We now show that if $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ does not ensures equivocation prevention under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then we can construct an adversary that wins in the $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game.

Claim 7. If $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ does not ensures equivocation prevention under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\mathsf{EQ\text{-}P}}$ such that

$$Pr\left[\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N})=1\right]\notin Negl(1^\kappa) \tag{37}$$

Proof. From Definition 2, if $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ does not ensures equivocation prevention under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\mathsf{EQ\text{-}P}}$ that satisfies

$$Pr\left[\begin{array}{l}(\mathsf{N_F},R,t,out_{\mathcal{A}_{\mathsf{EQ\text{-}P}}})\leftarrow\mathbf{Exec}_{\mathcal{A}_{\mathsf{EQ\text{-}P}},\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}}(1^\kappa,\mathsf{N})\\ \mathbf{Exp}^{\mathsf{EQ\text{-}P}}_{\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}}(1^\kappa,\mathsf{N},\mathsf{N_F},R,t,out_{\mathcal{A}_{\mathsf{EQ\text{-}P}}})=\top\end{array}\right]\notin Negl(1^\kappa) \tag{38}$$

Therefore, all that is left is to show that if Eq. 38 holds then Eq. 37 also holds.

Since $\mathcal{A}$ 'wins' in the equivocation prevention security experiment (Eq. 38), then following the security experiment (Alg. 8), $\mathcal{A}$ managed to produce two valid conflicting certificates ψ,ψ' with the equivocation prevention attribute $\mathsf{EQ\text{-}P}$, namely:

$$\begin{array}{l}\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}.\mathsf{WasValid}(\psi,\mathsf{EQ\text{-}P})=\\ \overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}.\mathsf{WasValid}(\psi',\mathsf{EQ\text{-}P})=\top\end{array} \tag{39}$$

Hence, following Eq. 39 and the implementation of $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}.\mathsf{WasValid}$ described in Alg. 20, we get that:

$$\begin{array}{rcl}\mathcal{TS}.\mathsf{Ver}(tvk,(Core(\psi),\mathsf{EQ\text{-}P}),\psi.\rho[\mathsf{EQ\text{-}P}].\sigma)&\equiv&\\ \mathcal{TS}.\mathsf{Ver}(tvk,(Core(\psi'),\mathsf{EQ\text{-}P}),\psi'.\rho[\mathsf{EQ\text{-}P}].\sigma)&\equiv&\top\end{array} \tag{40}$$

Since $\mathcal{A}_{\mathsf{EQ\text{-}P}}$ can generate two conflicting certificates ψ,ψ' as described in Eq. 40 with non-negligible probability, and following Claim 6 that ψ' (without loss of generality) was not 'honestly' generated by $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ honest authorities, it shows that adversary $\mathcal{A}_{\mathsf{EQ\text{-}P}}$ is a PPT adversary that is able to generate a message $m=(Core(\psi'),\mathsf{EQ\text{-}P})$ and signature $\sigma=\psi'.\rho[\mathsf{EQ\text{-}P}].\sigma$ over m with only the knowledge of the public group verification key $v=tvk$ and up to t oracle accesses on m, and therefore, following Def. 11:

$$Pr\left[\mathbf{Exp}^{EU}_{\mathcal{TS},\mathcal{A}_{\mathsf{EQ\text{-}P}}}(1^\kappa,\mathsf{N},t)=1\right]\in Negl(1^\kappa)$$

thus $\mathcal{A}_{\mathsf{EQ\text{-}P}}$ contradicts the unforgeability of $\mathcal{TS}$.

Therefore, $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ achieves equivocation prevention. □

We next argue that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ ensures equivocation prevention.

Lemma 3. *$\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ ensures equivocation prevention under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, assuming that $\mathcal{S}$ is a secure signature scheme, $\mathcal{E}$ is a secure encryption scheme and $\mathcal{TS}$ is a robust threshold signature scheme.*

Proof. Assume to the contrary that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ does not ensures equivocation prevention under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$. We will show that this implies that $\mathcal{TS}$ is not an unforgeable threshold signature scheme. According to Claim 7, this means there exists a PPT adversary $\mathcal{A}_{\mathsf{EQ\text{-}P}}$ that wins the $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A}_{\mathsf{EQ\text{-}P}},\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}}$ game with non-negligible probability. However, Lemma 2 shows that if there exists a PPT adversary $\mathcal{A}_{\mathsf{EQ\text{-}P}}$ that wins the $\overline{\mathbf{Exp}}^{\mathcal{TS}-Forge,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A}_{\mathsf{EQ\text{-}P}},\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}}$ game with non-negligible probability, then $\mathcal{TS}$ is not an unforgeable threshold signature scheme. □

C.2.6 Proving That (Original) PoC-PKI Also Achieves Equivocation Prevention

We complete our proof with the last phase of our proof methodology. We already showed that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ achieves equivocation prevention. To prove that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ also achieves equivocation prevention, we need to show that the fact that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ uses encryption to achieve equivocation prevention does not provide any advantage to the adversary in comparison to $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$.

To that end, we define the following indistinguishability game $\overline{\mathbf{Exp}}^{CPA-IND}_{\mathcal{A},\mathcal{E}}(1^\kappa)$:

1. The game randomly chooses $b \in \{0,1\}$.
2. If $b = 0$, we execute $\mathcal{A}$ with the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ scheme, and if $b = 1$, we execute $\mathcal{A}$ with the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ scheme.
3. $\mathcal{A}$ outputs b' $\in \{0,1\}$.
4. The game outputs 1 if $b = b$', otherwise 0.

Now, we revisit Theorem 3 presented in §5.1, and complete the proof that PoC-PKI achieves equivocation prevention.

Theorem 3. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves equivocation prevention under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, provided that $\mathcal{TS}$ is $(t \leq \mathcal{M}^{|\mathsf{N_F}|\leq\lfloor(n/3)\rfloor})$-existentially unforgeable, and $\mathcal{E}$ is CPA-indistinguishable.*

Proof. The only difference between the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ and the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ schemes is that the adversary has an advantage in PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$, because it also receives the encrypted secret information generated by PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.$\mathsf{Init}$ using the secure encryption scheme $\mathcal{E}$. Lemma 3 shows that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ ensures equivocation prevention under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, since $\mathcal{TS}$ is secure; we now show that security in the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ scheme also holds.

Assume to the contrary that although $\mathcal{E}$ is secure; there exists an adversary $\mathcal{A}$ that negates the claim that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ ensures equivocation prevention under the $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ model, namely:

$$Pr\left[\begin{array}{l}(\mathsf{N_F},R,t,out_{\mathcal{A}_{\mathsf{EQ\text{-}P}}})\leftarrow\mathbf{Exec}_{\mathcal{A}_{\mathsf{EQ\text{-}P}},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N})\\ \mathbf{Exp}^{\Delta\mathsf{ReTRA}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa,\mathsf{N},\mathsf{N_F},R,t,out_{\mathcal{A}_{\mathsf{EQ\text{-}P}}})=\top\end{array}\right]\notin Negl(1^\kappa) \tag{41}$$

Since the only difference between PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ and $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ is the use of $\mathcal{E}$ to encrypt the individual secret information, it means that $\mathcal{A}$ uses this advantage to win the experiment.

Consider an adversary $\mathcal{A}$' that simulates $\mathcal{A}$ in the aforementioned $\overline{\mathbf{Exp}}^{CPA-IND}_{\mathcal{A},\mathcal{E}}(1^\kappa)$ indistinguishability game, and outputs $b'=1$ if $\mathcal{A}$ wins the $\mathbf{Exp}^{\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathsf{EQ\text{-}P}}(1^\kappa,\mathsf{N})$ experiment (since we conclude it is an execution with the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ scheme, where $\mathcal{A}$ has an advantage), and outputs $b'=0$ otherwise (since it is probably an execution with the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign_{\{tsk_i\}_{i\in\mathsf{N}}}(i,\cdot)}_{\mathsf{N},tvk}$ scheme). Consequently, if such $\mathcal{A}$ exists, then we are able to construct $\mathcal{A}$' that wins the $\overline{\mathbf{Exp}}^{CPA-IND}_{\mathcal{A},\mathcal{E}}(1^\kappa)$ experiment with a non-negligible probability, thus contradicting the indistinguishability of $\mathcal{E}$.

Therefore, PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ also achieves equivocation prevention. □

C.3 Proofs of VAS, NF, NFA and UFAT

Lemma 4. *Let $\mathcal{S}$ be an existentially-unforgeable signature scheme. Then* PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves Verifiable Attribution of Statements (VAS) for $\mathcal{S}$ under model* $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$.

Proof. Assume to the contrary, that there exists an adversary $\mathcal{A}$ that 'wins' in the verifiable attribution of statements experiment $\mathbf{Exp}^{\text{VAS}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$, yet $\mathcal{S}$ is a secure signature scheme.

First, following the definition of honest entities from the execution process (Algorithm 1), $\mathcal{A}$ cannot change the execution of PoC-PKI algorithms by honest entities.

Second, following the convention described in § 3.3.1, in every PoC-PKI algorithm where the $\mathcal{S}.\mathsf{Sign}$ algorithm is used, the algorithm 'automatically' produces an attribution statement of the signed data.

Therefore, with accordance to the definition of $\mathbf{Exp}^{\text{VAS}}_{\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ from Req 1, $\mathcal{A}$ is capable of producing a message m and a signature σ over m signed by an

honest authority ι, *without* executing $\mathcal{S}.\mathsf{Sign}$ on ι with message m, with non-negligible probability.

However, if such adversary $\mathcal{A}$ exists, then according to definition of existential unforgeability (Def. 8), the following holds for $\mathcal{A}$

$$Pr\left[\mathbf{Exp}^{EU}_{\mathcal{A},\mathcal{S}}(1^\kappa) = 1\right] \notin Negl(1^\kappa) \tag{42}$$

thus contradicting the security of $\mathcal{S}$. □

Lemma 5. *Let $\mathcal{S}$ be an existentially-unforgeable signature scheme. Then* PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves Verifiable Attribution of Statements Non-frameability (NF), No False Accusations (NFA) and Use First-Accuse Time (UFAT) under model* $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$.

Proof. NF holds trivially since PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}.V_p$ always return $\perp$ (for any inputs).

The only accusation statement generated in PoC-PKI is via the Audit algorithm, when a $\Delta_{\text{PoC-PKI}}$-transparent certificate is inputted which the entity is supposed to be aware of, yet it does not. According to the CT$_{comp}$ $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Upgrade}$ algorithm, an honest entity always informs all other entities immediately after upgrading a certificate with the $\Delta\mathsf{TRA}$ attribute. Therefore, since according to $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ sent messages arrive to their respective destinations within bounded time which the $\Delta_{\text{PoC-PKI}}$ time takes into consideration, honest entities will always receive all $\Delta_{\text{PoC-PKI}}$-transparent certificates from honest entities *within the* $\Delta_{\text{PoC-PKI}}$ *time*, and therefore, NFA also holds.

UFAT also trivially holds, since in PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}.\mathsf{Audit}$ always stores the clock of the first accusation and outputs it in case of accusation. □

Theorem 1. *Let $\mathcal{S}$ be an existentially-unforgeable signature scheme. Then* PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves Verifiable Attribution of Statements (VAS) for $\mathcal{S}$, Non-frameability (NF), No False Accusations (NFA) and Use First-Accuse Time (UFAT) under model* $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$.

Proof. See Lemmas 4-5. □

D CT$_{comp}$: Certificate Transparency Complemented

In this section we provide a formal description of CT$_{comp}$, which is our best effort to complete the missing specifications of Certificate Transparency as defined in RFC6962, in the simplest possible way, to create a well-defined protocol whose properties can be analyzed (See Section E). Obviously, the analysis can be easily adapted to support improved, more efficient and/or complex variants of CT.

D.1 Mapping CT into the PKI Framework

We design CT$_{comp}$ by mapping the CT specification from RFC 6962 into the PKI security framework. We use pseudo-code to describe the main functionalities,

simplifying some low-level details and APIs defined in the specification. Since the specification does not provide all the necessary details to produce a formal protocol description, we use CT_{comp} to fill in the gaps to produce a formal, complete protocol description.

In CT_{comp} there are three types of roles: certificate authorities, loggers and monitors, represented respectively by the set $R = \{CA, L, M\}$. Loggers keep public logs of certificates issued by different CAs, and monitors validate that logs are published consistently. Hence, the set of entities in CT_{comp} is defined as $\mathsf{N} = \bigcup_{r \in R} \mathsf{N}_r$, where N_r is the subset of entities with role $r \in R$. For simplicity, we assume that each entity has only one role.

Note that R does not contain clients, since they are irrelevant with respect to the security properties. Clients are subjects of certificates, who want to ensure that there are no misissued certificates for identifiers they own, i.e., identifiers endorsed in their certificates as well as relying parties, who want to validate a specific certificate before they use it. Similarly, R does not contain *auditors*, which, as defined in the specifications, are not a separate entity but rather a way to offload auditing functionality from certificate subjects and relying parties. Clients, both certificate subjects and relying parties, simply rely on CAs, loggers and monitors for ensuring that the system achieves its claimed properties.

Operation-wise, each logger in CT_{comp} use generalized Merkle tree to produces one signed tree hash (STH) per one maximum merge delay (MMD) period. If no new certificates were added since the previous STH was produced, then the logger re-signs the most recent STH. All loggers use the same MMD, which is consistent with the current deployment of CT. Each honest monitor maintains a full copy of each log it watches and after each MMD period, it fetches the new STH along with all the newly added certificates. Then, the monitor ensures that the new STH complies with the updated copy of the log it maintains.

CT_{comp} employs a naive gossip approach, where every MMD period all monitors gossip all the new certificates and STHes they have learned since the previous gossip.

D.2 Model Function $\mathcal{M}^{\boldsymbol{CT_{comp}}}_{\boldsymbol{\Delta_{com}}, \boldsymbol{\Delta_{clk}}}$

We define the model of CT_{comp} as

$$\mathcal{M}^{CT_{comp}}_{\Delta_{com}, \Delta_{clk}}(\xi) = \mathcal{M}^{\text{2-rounds}}_{\text{SecInit}}(\xi) \wedge \mathcal{M}^{\text{COM}}_{\Delta_{com}}(\xi) \wedge \mathcal{M}^{\text{CLK}}_{\Delta_{clk}}(\xi) \wedge \mathcal{M}^{Auth}_{Mapping}(\xi)$$

where, $\mathcal{M}^{\text{2-rounds}}_{\text{SecInit}}$ is the secure initialization model (§ 3.2.2), $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$ is the communication model (§ 3.2.3), $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$ is the clock synchronization (§ 3.2.4), and $\mathcal{M}^{Auth}_{Mapping}$ enforces monitor-logger mapping as we discuss next. Note that $\mathcal{M}^{\text{COM}}_{\Delta_{com}}$ defines a bounded-delay communication model although RFC6962 states that some of the algorithms are asynchronous. However, those algorithms definitely use time-outs; the authors apparently meant to say the algorithms are *non-blocking*. Similarly, $\mathcal{M}^{\text{CLK}}_{\Delta_{clk}}$ assumes a bounded-drift clock synchronization, although such synchronization is not explicitly stated in the specifications.

In CT_{comp}, each monitor is assigned loggers to watch, as part of the secure initialization phase. Since this monitor-logger mapping is determined by the adversary, the model must make sure that the adversary does not abuse this privilege. In CT_{comp}, this could be abused by the adversary in the $\Delta_{CT_{comp}}$-transparency requirement, where the adversary is required to output a $\Delta_{CT_{comp}}$-transparent certificate and honest monitors which are not aware of ψ - although they should. However, the adversary could take advantage of this monitor-logger mapping capabilities into outputting a $\Delta_{CT_{comp}}$-transparent certificate which was signed by a logger which is not monitored by m, thus winning the $\mathbf{Exp}^{\Delta\mathsf{TRA}}$ game. Therefore, the model enforces an authentic mapping by making sure that the output of the adversary in the $\mathbf{Exp}^{\Delta\mathsf{TRA}}_{\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}}$ game complies with the monitor-logger mapping. This is captured by the following model function:

$$\mathcal{M}^{Auth}_{Mapping}(1^{\kappa},\mathsf{N},\mathsf{N_F},R,t,out_{\mathcal{A}}) = \begin{bmatrix}\forall(\Delta\mathsf{TRA},\psi,\iota) \leftarrow out_{\mathcal{A}}\ \exists t' \in \{1,\ldots,|\mathsf{N}|\} : \\ i[t'] = \iota\ \wedge\ \psi.\rho[\Delta\mathsf{TRA}].\iota \in inp[t']\end{bmatrix}$$

We consider the Δ delay used in CT_{comp} as $\Delta_{CT_{comp}} = 4\Delta_{com} + 2\Delta_{clk}$. $\Delta_{CT_{comp}}$ accounts for the time it takes to include the certificate in the log ($MMD \leq \Delta_{com}$), plus the time it takes for other entities to learn about new certificates (Δ_{com}), plus the time it takes to send a gossip message (Δ_{com}) and receive a gossip message (Δ_{com}). On top of that, because the clocks might be skewed by at most Δ_{clk}, we add one Δ_{clk} for when entities learn of new certificates and another Δ_{clk} for the gossip.

D.3 CT_{comp} PKI Implementation

D.3.1 Local State Variables

Each entity has a local clock clk and a local state S. All entities, regardless of their role, store the following information:

- $s.\mathsf{N}$: the unique-identifiers of all entities.
- $s.\iota$: the entity's unique-identifier.
- $s.role$: the entity's role.
- $s.PrivInfo$: private (secret) information.
- $s.PubInfo_i$: the public information of entity $i \in \mathsf{N}$.

In addition, entities also store role-specific information. Specifically, each CA $c \in \mathsf{N}_{CA}$ stores:

- $s.certs$: all the certificates issued/revoked by the CA.

Each monitor $m \in \mathsf{N}_M$ stores:

- $s.loggers \subseteq \mathsf{N}_L$: the loggers watched by m.
- $s.log_i$: the monitored information about logger $i \in s.loggers$, where:
  - $s.log_i.version$ is the version number of the logger's latest STH.
  - $s.log_i.STH$ is the latest STH.
  - $s.log_i.entries$ are all the certificates represented by the STH.
  - $s.log_i.accusations$ are all the accusations when the logger did not send updated information in time.

- $s.log_i.msgs$ are all messages received about this log, either from the logger or through gossip.

Lastly, each logger $\ell \in \mathsf{N}_L$ stores:

– $s.entries$: *ordered* list of certificates currently logged in the Merkle tree.
– $s.toAppend$: all newly added certificates not yet logged.

D.3.2 Init Algorithm

The initialization algorithm $CT_{comp}.\mathsf{Init}$ (Algorithm 22) consists of two rounds. In the first round (lines 1-7), each entity extract from the input inp and store locally the set of entities $s.\mathsf{N}$, its identifier $s.\iota$, its role $s.role$, and monitors also take the subset of loggers they need to monitor $s.loggers$. Then, each entity generate a private/public signing key pair and a public key for hashing. The public keys are then sent to all other entities to be processed in the second round. In the second round (lines 8-11), each entity extract from the input the public keys sent by the other entities in the first round, and store them locally.

Algorithm 22 $CT_{comp}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Init}(1^\kappa)$

1: **if** $s.initCounter = 0$ **then**
2: $(s.role, s.loggers) \leftarrow inp.m$
3: $(s.PrivInfo.sk, s.PubInfo_{s.\iota}.vk) \leftarrow \mathcal{S}.\mathsf{Gen}(s.1^\kappa)$
4: $out \leftarrow (\text{'public key'}, s.PubInfo_{s.\iota}.vk)$
5: $s.PubInfo_{s.\iota}.hk \leftarrow \{0,1\}^n$
6: $s.initCounter \leftarrow 1$
7: $out \leftarrow \{(\text{'send'}, m = [PubInfo = s.PubInfo_{s.\iota}], j)\}_{j \in s.\mathsf{N} - s.\iota}$
8: **else if** $s.initCounter = 1$ **then**
9: $s.PubInfo_{inp.j} \leftarrow inp.m.PubInfo$
10: $s.initCounter \leftarrow 2$
11: **end if**

D.3.3 Issue Algorithm

The certificate issuance algorithm $\mathrm{CT}_{comp}.\mathsf{Issue}$ (Algorithm 23) is used by a CA to produce an *accountable* certificate ψ. Other authorities do not issue certificates. The algorithm uses $\mathcal{S}.\mathsf{Sign}$ to sign the mapping between the identifier id and some public information pub. An honest CA only issues a certificate if the requesting client is eligible for the specific id. After the certificate is produced, the CA adds it to its local state and returns it to the requesting client. We do not model the interactions of CAs and their clients. In the context of the X.509 web PKI, an accountable certificate ψ can be viewed as a valid X.509 certificate and can be used as such.

Algorithm 23 $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Issue}(id, pub, sd, ed)$

Comment: An honest authority invokes Issue only if the client is eligible for the id it requests.

// Check if invoked by a CA
1: **if** $s.\iota \notin \mathsf{N}_{CA}$ **then return** $\perp$
// Generate a basic certificate
2: $data \leftarrow (id, pub, sd, ed, \mathsf{ACC}, clk)$
// Generate an accountable certificate
3: $\sigma = \mathcal{S}.\mathsf{Sign}(s.PrivInfo.sk, data)$
4: $\rho \leftarrow \{(\mathsf{ACC}, (\sigma, s.\iota, clk))\}$
5: $\psi \leftarrow (id, pub, sd, ed, \rho)$
// Add the new certificate to the local state
6: $s.certs \mathrel{+}= \psi$
7: **return** ψ

D.3.4 Upgrade Algorithm

The upgrade algorithm $\mathrm{CT}_{comp}.\mathsf{Upgrade}$ (Algorithm 24) needs to handle only one scenario. Namely, the upgrade of an accountable certificate to a $\Delta_{CT_{comp}}$-transparent certificate, i.e., a certificate with an attribute $attr = \Delta\mathsf{TRA}$. A $\Delta_{CT_{comp}}$-transparent certificate is a certificate that has been logged by one of the loggers, that is, a certificate for which an SCT has been issued and a proof of inclusion can be produced.

Since certificates can only logged by loggers, the loggers are also the only entities that can upgrade certificates. However, logging a certificate is not an atomic operation, and therefore, cannot be completed immediately; this is a direct result of only periodically updating the tree and generating new STHes. Thus, when a request to upgrade a valid, accountable certificate arrives, loggers create a $\Delta_{CT_{comp}}$-transparency *pending* certificate, and when the next STH will be generated, the pending certificate will be upgraded into a $\Delta_{CT_{comp}}$-transparent certificate. Hence, When $\mathrm{CT}_{comp}.\mathsf{Upgrade}$ is invoked on an accountable but not yet pending $\Delta_{CT_{comp}}$-transparency or $\Delta_{CT_{comp}}$-transparent certificate ψ, the logger immediately returns a pending $\Delta_{CT_{comp}}$-transparency certificate ψ_p, which is ψ with an SCT, the logger's promise to log the certificate within its MMD. After each MMD-defined period of time, the logger adds new pending certificates to its log, produces a new Merkle Tree Hash (MTH), and finally signs it producing an STH (Signed Tree Hash), which serves as a commitment to the new version of the log. When $\mathrm{CT}_{comp}.\mathsf{Upgrade}$ is invoked on ψ_p after a new STH was produced, the logger returns an upgraded $\Delta_{CT_{comp}}$-transparent certificate, which includes an SCT and a proof of inclusion.

D.3.5 Revoke Algorithm

The certificate revocation algorithm $\mathrm{CT}_{comp}.\mathsf{Revoke}$ (Algorithm 25) is used by an eligible CA (the one who issued the certificate) to revoke a certificate ψ, which is valid, not expired or not already revoked at the time of the revocation request. If

Algorithm 24 $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Upgrade}(\psi, attr)$

// Check that invoked by a logger, ψ is valid, and the upgrade attribute is $\Delta\mathsf{TRA}$
1: **if** $s.role \neq$ 'L' $\vee\ CT_{comp}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{WasValid}(\psi, \mathsf{ACC}) \neq \top\ \vee attr \neq \Delta\mathsf{TRA}$ **then return** $\perp$

// If there is already a matching upgraded (transparent) certificate, return it.
2: **if** $\exists\psi' \in s.entries\ s.t.\ Core(\psi) = Core(\psi')\ \wedge\ CT_{comp}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{WasValid}(\psi', \Delta\mathsf{TRA})$ **then return** ψ'

// If there is a matching pending certificate and no new STH yet, return it.
3: **if** $\exists\psi' \in s.toAppend\ s.t.\ Core(\psi) = Core(\psi') \wedge CT_{comp}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{WasValid}(\psi', \Delta\mathsf{TRA}) = Pending \wedge s.STH.clk < \psi'.\rho[Pend\Delta\mathsf{TRA}].clk$ **then return** ψ'

// Otherwise, issue a pending certificate
4: **return** Produce_Pending(ψ)

5: **procedure** PRODUCE_PENDING(ψ)
    // Generate SCT and add it as a proof
6: $data \leftarrow (Core(\psi), Pend\Delta\mathsf{TRA}, clk)$
7: $SCT = (s.\iota, clk, data, \mathcal{S}.\mathsf{Sign}(s.PrivInfo.sk, data))$
8: $\psi.\rho[Pend\Delta\mathsf{TRA}] \leftarrow SCT$
    // Store ψ to be included in the next STH
9: $s.toAppend \mathrel{+}= \psi$
10: **return** ψ
11: **end procedure**

the certificate was already revoked, the algorithm outputs the revoked certificate ψ_r. Otherwise, the algorithm revokes the certificate by signing the revocation statement and outputting the certificate with the signed revocation statement. The signature is generated using the signing algorithm $\mathcal{S}.\mathsf{Sign}$. Note that both X.509 and CT do not explicitly instruct the CA to sign upon revocation, however, this simple extension is necessary to ensure accountability of performing the revocation operation.

Algorithm 25 $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Revoke}(\psi)$

// Verify that ψ was issued by the executing CA and ψ is a valid, not expired certificate
1: **if** $\psi.\rho[\mathsf{ACC}].\iota \neq s.\iota\ \vee\ CT_{comp}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{WasValid}(\psi) \neq \top\ \vee\ \psi.ed < clk$ **then return** $\perp$

// If ψ was already revoked, return it
2: **if** $\exists\psi_r \in s.certs[\psi.id]\ s.t.\ CT_{comp}.\mathsf{WasValid}(\psi_r, \mathsf{REV})\ \wedge\ Core(\psi) = Core(\psi_r)$ **then return** ψ_r
3: $\psi_r \leftarrow \psi$
// Revoke ψ
4: $data \leftarrow (Core(\psi), \mathsf{REV}, clk)$
5: $\sigma \leftarrow \mathcal{S}.\mathsf{Sign}(s.PrivInfo.sk, data)$
6: $\psi_r.\rho[\mathsf{REV}] \leftarrow (\sigma, s.\iota, clk)$
// Add ψ_r to the local state
7: $s.certs[\psi_r.id] \mathrel{+}= \psi_r$
8: **return** ψ_r

D.3.6 IsRevoked Algorithm

The check revocation status algorithm CT_{comp}.IsRevoked (Algorithm 26) first verifies that the inputted certificate is eligible to be examined, namely that it is cryptographically valid, did not expire and that it was issued by the executing entity. If the certificate was already revoked, then the algorithm outputs the revoked version of the certificate. If the certificate is indeed valid and was not revoked, the algorithm outputs the certificate with the not-revoked (NREV) attribute, which contains a signed proof that the certificate is not revoked until the current time clk.This is essentially equivalent to the operation of OCSP.

Algorithm 26 $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$.IsRevoked($\psi$)

// Verify that ψ was issued by the authority and ψ is a valid, not expired certificate
1: **if** $\psi.\rho[\mathsf{ACC}].\iota \neq s.\iota \ \vee \ CT_{comp}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{WasValid}\psi \neq \top \ \vee \ \psi.ed < clk$ **then return** $\perp$
// If ψ was already revoked, return it
2: **if** $\exists \psi_r \in s.certs[\psi.id] \ s.t. \ CT_{comp}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \ \wedge \ Core(\psi) = Core(\psi_r)$ **then return** ψ_r
// Add the non-revocation proof to ψ'
3: $\psi' \leftarrow \psi$
4: $data \leftarrow (Core(\psi), \mathsf{NREV}, clk)$
5: $\sigma = \mathcal{S}.\mathsf{Sign}(s.PrivInfo.sk, data)$
6: $\psi'.\rho[\mathsf{NREV}] \leftarrow (\sigma, s.\iota, clk)$
7: **return** ψ'

D.3.7 WasValid Algorithm

The validation algorithm CT_{comp}.WasValid (Algorithm 27) verifies two requirements: 1) the current time is between the certificate's validity period, and that 2) the accountability proof is cryptographically valid. These requirements are enough for the accountability property, see lines 1-5. For the revoke property REV and the not-revoked property NREV, the algorithm verifies that the proofs are valid, see lines 6-7. For transparency (lines 8-11), the algorithm first check if the certificate is pending. If not, the algorithm verifies that the certificate is included in the public log. If ψ is indeed transparent, then it contains the root of the Merkle tree, the signature over the root signed by the logger and the necessary proof to check the path from ψ to the tree root.

D.3.8 Audit Algorithm

The $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$.Audit algorithm (Algorithm 28) checks whether there are problems with the inputted certificate ψ with regards to the inputted attribute $attr$. In CT_{comp}, only monitors are required to support the algorithm. Namely, the algorithm checks whether there were accusations against the logger that issued the $attr$ attribute for ψ. If such accusations exist, the algorithm outputs the first accusation. Alternatively, if the attribute is $\Delta_{CT_{comp}}$-transparency, then

Algorithm 27 $CT_{comp}.\mathsf{WasValid}(\psi, pk, attr\ [, tms])$

$\chi \leftarrow \psi.\rho[\mathsf{ACC}]$
// Verify that the accountability proof is cryptographically valid and that the certificate has not expired
if $tms < \psi.sd\ \vee\ tms > \psi.ed\ \vee\ \mathcal{S}.\mathsf{Ver}(pk, (Core(\psi), \mathsf{ACC}, \chi.clk), \chi.\sigma) \neq \top$ **then**
    return $\perp$
end if
// If no attribute was supplied or the attribute is accountability, return true
if $attr = \perp\ \vee\ attr = \mathsf{ACC}$ **then return** $\top$
$\eta \leftarrow \psi.\rho[attr]$
// For the REV and NREV properties, check that the proof is cryptographically valid
if $attr \in \{\mathsf{REV}, \mathsf{NREV}\}$ **then return** $\mathcal{S}.\mathsf{Ver}(pk, (Core(\psi), attr, \eta.clk), \eta.\sigma)$
// For transparency - check inclusion
if $attr = \Delta\mathsf{TRA}$ **then**
    // Check if the certificate is pending
    if $\mathcal{S}.\mathsf{Ver}(pk, (Core(\psi), Pend\Delta\mathsf{TRA}, \eta.clk), \eta.\sigma)$ **then return** $Pending$
    // Check that the certificate is indeed included in the log
    return $\mathcal{S}.\mathsf{Ver}(pk, \eta.sth.mth.hash, \eta.sth.mth.sigma) \wedge$
        $\mathcal{GMT}.\mathsf{VerIncProof}(\kappa, Core(\psi), \eta.index, \eta.sth.mth.len, \eta.sth.mth.proof, \sigma)$
end if
return $\perp$

the algorithm checks whether ψ appears in relevant monitored log. If not, the algorithm outputs a proof of misbehavior, by outputting ψ and the messages received by the logger, so the entity that verifies the proof can see that no such certificate was delivered to executing monitor.

Algorithm 28 $CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}.\mathsf{Audit}(attr, \psi)$

if $s.role \neq$ 'M' **then return** $\perp$
$isValid \leftarrow CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}.\mathsf{WasValid}(\psi, attr)$
if $s.log_{\psi.\rho[attr].\iota}.accusations \neq \perp$ **then**
    return $s.log_{\psi.\rho[attr].\iota}.accusations[0]$
else if $isValid\ \wedge\ attr = \Delta\mathsf{TRA}\ \wedge\ clk \geq \psi.\rho[\Delta\mathsf{TRA}].clk + \Delta\ \wedge$
        $\nexists\psi' \in s.log_{\psi.\rho[\Delta\mathsf{TRA}].\iota}.entries\ s.t.\ Core(\psi) = Core(\psi')$ **then**
    return $(error =$ 'misbehavior', $\psi, s.log_{\psi.\rho[\Delta\mathsf{TRA}].\iota}.msgs)$
end if
return $isValid$

D.3.9 Time Algorithm

The algorithm $\mathrm{CT}_{comp}.\mathsf{Time}$ (Algorithm 29) handles time-based events and is implemented according to the needs of each of CT's entities. This algorithm defines the periodically invoked operations of loggers and monitors. We do not

define such operations for CAs in this algorithm since CAs operations to issue, upgrade or revoke certificates are handled using the corresponding algorithms.

The main operation for a logger is to periodically append certificates to the tree such that it obeys its stated MMD. When it is time to update the log, the logger appends new certificates to the tree, it saves the current STH and its timestamp, then it updates the tree by appending new certificates, and produces a new MTH and STH. Once the log is updated, for each newly added certificate, the logger upgrades it to a transparent certificate by adding the $\Delta\mathsf{TRA}$ attribute and the corresponding proof of inclusion of the certificate in the log.

The main operation for a monitor is to periodically contact all loggers it watches and obtain the new STH and newly added certificates, and to gossip this information to all other monitors. For simplicity, we rely on a simple but inefficient gossip protocol where everyone gossips with everyone by sending the same message to each monitor.

D.3.10 Incoming Algorithm

Algorithm $\mathrm{CT}_{comp}.\mathsf{Incoming}$ (Algorithm 30) handles incoming messages and is implemented according to the needs of each of the CT's entities. We model only the messages necessary to provide the security properties and note that additional messages would be needed to accommodate clients and additional functionalities.

Loggers handle one message *get-entries (start)* to produce a set of certificates currently in the tree starting at index *start* as well as the latest STH. Monitors handle two incoming messages, a response to the *get-entries* messages and gossip messages. When a monitor receives a response to *get-entries*, it checks it by verifying the signature on the MTH , checks the reported MTH against the set of current and new certificates, checks the new certificates for any invalid or suspicious ones, and finally saves the newly received information. When a monitor receives gossip messages, it compares the incoming gossiped data against the local state and reports any discrepancies.

E CT_{comp} Analysis

E.1 Proof of Accountability, Revocation Accountability and Non-Revocation Accountability

We start with accountability and revocation accountability.

Theorem 4. *CT_{comp} $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ satisfies security requirements* accountability, revocation accountability *and* non-revocation accountability, *under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, under the same assumptions as in Theorem 6.*

Proof. Valid certificates in CT_{comp} $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ can only be issued using CT_{comp} $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Issue}$ (Alg. 23), certificates can only be revoked using CT_{comp} $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Revoke}$ (Alg. 25) and non-revocation accountability can only be produced using CT_{comp}

Algorithm 29 $CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}$.Time()

```
    // If current entity is a logger
1:  if s.role = 'L' then LOGGER_TIME()
    // If current entity is a monitor
2:  if s.role = 'M' then MONITOR_TIME()

3:  procedure LOGGER_TIME()
        // Every MMD
4:      if time to append new certificates then
            // Append all new certificates from s.toAppend
5:          s.entries += s.toAppend
            // Update tree and produce a new STH
6:          s.MTH ← (clk, len = |s.entries|, hash = GMT.GenTree(s.entries))
7:          s.root ← (s.ι, mth = s.MTH, σ = S.Sign(S.PrivInfo.sk, s.MTH))
            // Add a proof of inclusion to each newly added certificate
8:          j ← 1
9:          for each ψ ∈ {s.entries ∩ s.toAppend} do
10:             ψ.ρ[ΔTRA] ← (sth = s.root, index = |s.entries| + j,
                                clk = s.MTH.clk, proof = GMT.GenIncProof(Core(ψ), s.entries))
11:             j ← j + 1
12:         end for
13:         s.toAppend = ⊥
14:     end if
15: end procedure
16: procedure MONITOR_TIME()
        // Every MMD
17:     if time to retrieve log changes then
18:         m = {type = 'get-entries', start=|s.log_i.entries|}
19:         out ← {('send', m, j)}_{j∈s.loggers}
20:     end if
21:     if time for logger ℓ ∈ s.loggers to send the newly logged certificates and STH passed then
22:         out ← (IA, ℓ, clk)
23:         s.log_ℓ.accusations += out
24:     end if
25: end procedure
```

Algorithm 30 $CT_{comp}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Incoming}()$

for each message $msg \in inp$ **do**
  if $s.initCounter < 2$ **then** PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Init($msg$)
  if $s.role =$ 'L' **then** Logger_Incoming(msg)
  if $s.role =$ 'M' **then** Monitor_Incoming(msg)
end for

procedure Logger_Incoming(msg)
  if $msg.m.type =$ 'get-entries' **then**
    $m = (type = \text{'entries'}, entries = s.certs[msg.m.start : |s.entries| - 1],$
      $sth = s.root, from = msg.m.start, to = |s.entries| - 1)$
    $out \leftarrow$ ('send', $m, msg.j$)
  end if
end procedure
procedure Monitor_Incoming(msg)
  switch $msg.m.type$ **do**
    case *'gossip-entries'*: Handle_Gossip_Entries(msg)
    case *'notify-error'*: Handle_Notify_Error(msg)
    case *'entries'*: Handle_Get_Entries(msg)
end procedure

$^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$.IsRevoked (Alg. 26). The implementation of the CT$_{comp}$ $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$.Issue, CT$_{comp}$ $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$.Revoke and CT$_{comp}$ $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$.IsRevoked algorithms is *identical* to the implementation of the matching algorithms in PoC-PKI, i.e., PoC-PKI.Issue (Alg. 15), PoC-PKI.Revoke (Alg. 17) and PoC-PKI.IsRevoked (Alg. 18), with only one minor difference; that is, in CT$_{comp}$ $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$, only CAs can issue certificates, and therefore, the first line of CT$_{comp}$.Issue verifies that the entity that executes the algorithm is indeed a CA. Other than that, the algorithms are identical. However, this line has no impact on the security of the system, and only needed since in CT$_{comp}$ $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ all entities have the same role. Therefore, we avoid repetition, and refer to the proof methodology described in Appendix C.1, which also applies here, and proves that CT$_{comp}$ $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ ensures accountability and revocation accountability. □

E.2 Proof of $\boldsymbol{\Delta_{CT_{comp}}}$-transparency

Proof Methodology. To prove that CT$_{comp}$ $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ achieves the attributes that are implemented using the secure signature scheme $\mathcal{S}$, we use the following methodology:

1. We first define a variation of CT$_{comp}$ $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ called $\overline{\text{CT}_{\text{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ (see Def. 22), where a PPT oracle algorithm $OSign(sk,\cdot)$ is used to generate signatures using a secret key sk *instead* of entity $\iota \in \mathsf{N}$, where sk is the matching secret signing key of the verification key vk, see Section E.2.1.

Algorithm 31 $CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}$.Incoming() Cont.

procedure HANDLE_NOTIFY_ERROR(msg)
  if $msg.m.error =$ *'Equivocation'* **then** Handle dishonest CA
  if $msg.m.error =$ *'Bad MTH'* $\vee$ *'Conflicting logs'* **then** Handle dishonest logger
end procedure
procedure HANDLE_GOSSIP_ENTRIES(msg)
  if $\mathcal{S}.\mathsf{Ver}(s.PubInfo_{msg.j}.vk, msg.m.STH.MTH, msg.m.STH.\sigma) \neq \top$ **then return** $\perp$
  $s.log_{msg.m.logger}.msgs \mathrel{+}= msg.m$
  if $|s.log_i.entries| < msg.m.to$ **then**
    $s.log_i.entries \mathrel{+}= m.entries[|s.log_i.entries| - msg.m.start : msg.m.to - msg.m.start]$
    $s.log_{msg.j}.STH \leftarrow \mathcal{GMT}.\mathsf{GenTree}(sec, s.log_{msg.j}.entries)$
  end if
  if $msg.m.sth \neq s.log_{msg.j}.STH$ **then**
    $m = (type=$*'notify-error'*$, data = (msg.m, s.log[msg.m.data.id]),$ error = 'Conflicting logs')
    $out \leftarrow \{(\text{'send'}, m, j)\}_{j \in s.\mathsf{N} - s.\iota}$
  end if
end procedure
procedure HANDLE_GET_ENTRIES(msg)
  if $\mathcal{S}.\mathsf{Ver}(s.PubInfo_{msg.j}.vk, msg.m.STH.MTH, msg.m.STH.\sigma) \neq \top$ **then return** $\perp$
  $s.log_{msg.j}.msgs \mathrel{+}= msg.m$
  if $\mathcal{GMT}.\mathsf{GenTree}(1^\kappa, s.log_i.entries \mid\mid msg.m.entries) \neq msg.m.sth$ **then**
    $m = (type=$*'notify-error'*$, data = msg.m,$ error = 'Bad MTH')
    $out \leftarrow \{(\text{'send'}, m, j)\}_{j \in s.\mathsf{N} - s.\iota}$
  else ▷ Check for equivocation
    for $\psi \in msg.m.entries$ **do**
      if $\exists \psi' \in s.certs$ *s.t.* $Core(\psi) = Core(\psi')$ **then**
        $m = (type=$*'notify-error'*$, id = s.\iota, data = (msg.m, \psi, \psi'),$ error = 'Equivocation')
        $out \leftarrow \{(\text{'send'}, m, j)\}_{j \in s.\mathsf{N} - s.\iota}$
      end if
    end for
    $m = (type=$*'gossip-entries'*$, id = s.\iota, data = msg.m)$
    $out \leftarrow \{(\text{'send'}, m, j)\}_{j \in s.\mathsf{N} - s.\iota}$
  end if
  $s.log_i.entries \mathrel{+}= msg.m.entries$
  $s.log_i.STH = msg.m.sth$
end procedure

2. Then, we define a game called $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$, where we execute an adversary $\mathcal{A}$ with the $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ scheme, and ask $\mathcal{A}$ to output a message m and signature σ over m, signed using the public verification key vk, namely, without $\mathcal{A}$ knowing the matching signing key sk, nor $\mathcal{A}$ can use the oracle access to sign m, see Section E.2.2.
3. We then formulate Lemma 6, showing that the existence of an adversary that 'wins' the $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability means that $\mathcal{S}$ is not a secure signature scheme, see Section E.2.3.
4. We then prove that if $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ does not achieves $\Delta_{CT_{comp}}$-transparency, we can construct an adversary that wins the $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}$ game with non-negligible probability, see Section E.2.4.
5. After that, we combine steps $1-4$, see Section E.2.5.
6. Finally, we show that the security argument also holds for the original $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ scheme, see Theorem 5 in Section E.2.6.

E.2.1 The $\overline{\text{CT}_{\text{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ Scheme

We now define a variation of the $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ scheme called $\overline{\text{CT}_{\text{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$.

Definition 22. *Let $\mathcal{S}$, $\mathcal{GMT}$ and H be a signature, generalized Merkle tree and CRHF schemes, respectively, and let $(sk, vk) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$, for a given security parameter 1^κ. Given a PPT oracle $OSign(sk,\cdot)$, $\overline{\text{CT}_{\text{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ is a PKI scheme which is identical to the implementation of the $CT_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ scheme with the following changes:*

1. *The $\overline{\text{CT}_{\text{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}.\mathsf{Init}$ algorithm is the same as $CT_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Init}$ (Alg. 22), except for replacing line 2 of $CT_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Init}$ with the following line:*

$$(s.PrivInfo.sk, s.PubInfo_{s.\iota}.vk) \leftarrow (nil, vk)$$

   where vk is the public verification key of $\mathcal{S}$, given as input to $\overline{\text{CT}_{\text{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$.
2. *In all the algorithms that use the signing key,* replace *the following code:*

$$\mathcal{S}.\mathsf{Sign}(s.PrivInfo.sk, data)$$

   with the following code:

$$OSign(sk, data)$$

   namely, generate proof by signing data using the oracle access to the sign operation $\mathcal{S}.\mathsf{Sign}$.
3. *In the $\overline{\text{CT}_{\text{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}.\mathsf{Time}$ algorithm,* replace *the following code:*

$$\mathcal{GMT}.\mathsf{GenTree}(s.entries)$$

   with the following code:

$$s.entries$$

Note that the security of $\overline{\mathrm{CT}_{\mathrm{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ relies solely on the security of the signature scheme used, as opposed to the (original) $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ scheme which *also* relies on the security of the generalized Merkle tree and CRHF used.

E.2.2 The $\overline{\mathbf{Exp}}^{\boldsymbol{CT_{comp}-Forge,\mathcal{M}}}_{\boldsymbol{\mathcal{A},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}}$ Game

We now define the $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}}_{\mathcal{A},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}(1^\kappa,\mathsf{N})$ game:

1. Generate key pair $(sk,vk) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$.
2. Randomly choose an authority $\iota \xleftarrow{R} \mathsf{N}$.
3. Execute $\mathcal{A}$ with the $\overline{\mathrm{CT}_{\mathrm{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ scheme, i.e.,
$$(\mathsf{N_F}, R, t, out_{\mathcal{A}}) \leftarrow \mathbf{Exec}_{\mathcal{A},\overline{\mathrm{CT}_{\mathrm{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}}(1^\kappa,\mathsf{N})$$
4. $\mathcal{A}$ outputs message m and signature σ, i.e., $(m,\sigma) \leftarrow out_{\mathcal{A}}$.
5. The experiment outputs 1 if:
   (a) $\mathcal{S}.\mathsf{Ver}(vk,m,\sigma) = \top$
   (b) $\mathcal{A}$ did not use the oracle access on m.
   (c) $\mathcal{A}$ satisfies model $\mathcal{M}$ (see Def. 1).
   (d) ι is an honest authority, i.e., $\iota \in \mathsf{N} - \mathsf{N_F}$
   Otherwise, the experiment outputs 0.

E.2.3 The Relation Between $\overline{\mathbf{Exp}}^{\boldsymbol{CT_{comp}-Forge,\mathcal{M}}}_{\boldsymbol{\mathcal{A},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}}$ and the Security of the Signature Scheme $\mathcal{S}$

We now show that the existence of an adversary that 'wins' the $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}}_{\mathcal{A},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}$ game with non-negligible probability means that $\mathcal{S}$ is not a secure signature scheme.

Lemma 6. *If there is a PPT adversary $\mathcal{A}$ that satisfies*
$$Pr\left[\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}}_{\mathcal{A},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}{}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}(1^\kappa,\mathsf{N}) = 1\right] \notin Negl(1^\kappa) \tag{43}$$
then $\mathcal{S}$ is not a secure signature scheme.

Proof. Assume to the contrary that such adversary $\mathcal{A}$ exists, yet $\mathcal{S}$ is a secure signature scheme.

Following Section E.2.2, if $\mathcal{A}$ 'wins' the $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}}_{\mathcal{A},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}{}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$ game with non-negligible probability, then this means that $\mathcal{A}$ can output a message m and a valid signature σ over m, where $\mathcal{A}$ has only access to the verification key and oracle accesses to the signing key, without requesting the oracle to sign m.

Therefore, according to definition of existential unforgeability (Def. 8), the following holds for $\mathcal{A}$
$$Pr\left[\mathbf{Exp}^{EU}_{\mathcal{A},\mathcal{S}}(1^\kappa) = 1\right] \notin Negl(1^\kappa) \tag{44}$$
thus contradicting the security of $\mathcal{S}$. □

E.2.4 Linking $\Delta_{CT_{comp}}$-transparency to the $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}$ Game

We now show that if $\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$ does not ensures $\Delta_{CT_{comp}}$-transparency under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, then we can construct an adversary that wins in the $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}$ game.

Claim 8. If $\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$ does not ensures $\Delta_{CT_{comp}}$-transparency under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\Delta\mathsf{TRA}}$ such that

$$Pr\left[\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A}_{\Delta\mathsf{TRA}},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}(1^\kappa,\mathsf{N})=1\right]\notin Negl(1^\kappa) \tag{45}$$

Proof. From Definition 2, if $\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$ does not ensures $\Delta_{CT_{comp}}$-transparency under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, then there exists a PPT adversary $\mathcal{A}_{\Delta\mathsf{TRA}}$ that satisfies

$$Pr\left[\begin{array}{l}(\mathsf{N_F},R,t,out_{\mathcal{A}_{\Delta\mathsf{TRA}}})\leftarrow\mathbf{Exec}_{\mathcal{A}_{\Delta\mathsf{TRA}},\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}}(1^\kappa,\mathsf{N})\\ \mathbf{Exp}^{\Delta\mathsf{TRA}}_{\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}}(1^\kappa,\mathsf{N},\mathsf{N_F},R,t,out_{\mathcal{A}_{\Delta\mathsf{TRA}}})=\top\end{array}\right]\notin Negl(1^\kappa) \tag{46}$$

Therefore, all that is left is to show that if Eq 46 holds then Eq 45 also holds.

First, according to the description of the security experiment $\mathbf{Exp}^{\Delta\mathsf{TRA}}_{\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}}$ (Alg. 5), the return value of the experiment is TRUE only if, among other criteria, the following is TRUE:

$$\begin{aligned}\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}.\mathsf{WasValid}(\psi,\Delta\mathsf{TRA})\ \wedge\\ R.out[t],R.out[t\text{-}1]\neq\mathrm{IA}\end{aligned} \tag{47}$$

for a certificate ψ outputted by the adversary, and there exists two honest authorities that are not aware of ψ although they should, and they cannot indicate any problem with ψ nor with the authority that issued $\Delta\mathsf{TRA}$ for ψ.

Second, according to the implementation of $\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}.\mathsf{WasValid}$, as described in Alg. 20, the algorithm executes

$$\mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota},(Core(\psi),\Delta\mathsf{TRA},\eta.clk),\eta.\sigma) \tag{48}$$

for $\eta=\psi_r.\rho[\Delta\mathsf{TRA}]$.

Lastly, the only place in $\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$ where an honest authority ι computes its keys is in the $\overline{\mathrm{CT}_{\mathrm{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}.\mathsf{Init}$ algorithm (Algorithm 14); specifically the sign/verify key pair is generated in line 2, using the $\mathcal{S}.\mathsf{Gen}$ algorithm.

Furthermore, the signing key is only used in algorithms: $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$.Issue, $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$.Revoke, $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$.IsRevoked and $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$.Upgrade, and only with the $\mathcal{S}$.Sign algorithm.

Thus, following Eq. 46, the value described in Eq. 47 must be TRUE, and as a result, Eq. 48 must also equal TRUE. Accordingly, with accordance to $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$'s implementation, adversary $\mathcal{A}_{\Delta\mathsf{TRA}}$ is a PPT adversary that for a message $m = (Core(\psi), \Delta\mathsf{TRA}, \eta.clk)$ was able to generate a signature $\sigma = \eta.\sigma$ that is validated with non-negligible probability with the verification key $vk = s.PubInfo_{\eta.\iota}.vk$, without access to the signing key, and without ever having the honest authority ι sign m. Hence, such $\mathcal{A}_{\Delta\mathsf{TRA}}$ adversary satisfies Eq. 45. □

E.2.5 Completing the Proof that $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$ Ensures $\Delta_{CT_{comp}}$-transparency

Lemma 7. *$\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$ ensures $\Delta_{CT_{comp}}$-transparency under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$.*

Proof. Assume to the contrary that $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$ does not ensures $\Delta_{CT_{comp}}$-transparency; we will show that this implies that $\mathcal{S}$ is not a secure signature scheme. According to Claim 1, this means there exists a PPT adversary $\mathcal{A}$ that wins the $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}}$ game with non-negligible probability. However, Lemma 6 shows that if there exists a PPT adversary $\mathcal{A}$ that wins the $\overline{\mathbf{Exp}}^{CT_{comp}-Forge,\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}}$ game with non-negligible probability, then $\mathcal{S}$ is not a secure signature scheme. □

E.2.6 Proving That (Original) $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ Also Ensures $\Delta_{CT_{comp}}$-transparency

We complete our proof with the last phase of our proof methodology. We already showed that $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$ ensures $\Delta_{CT_{comp}}$-transparency under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$. To prove that $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ also ensures equivocation prevention under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, we need to show that the fact that $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ uses generalized Merkle tree to ensure $\Delta_{CT_{comp}}$-transparency does not provide any advantage to the adversary in comparison to $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$.

To that end, we define the following indistinguishability game $\overline{\mathbf{Exp}}^{CPA-IND}_{\mathcal{A},\mathcal{E}}(1^\kappa)$:

1. The game randomly chooses $b \in \{0,1\}$.
2. If $b = 0$, we execute $\mathcal{A}$ with the $\overline{\text{CT}_{\text{comp}_{\mathsf{N},\iota,vk}}}^{OSign(sk,\cdot)}$ scheme, and if $b = 1$, we execute $\mathcal{A}$ with the $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ scheme.
3. $\mathcal{A}$ outputs b' $\in \{0,1\}$.
4. The game outputs 1 if $b = b$', otherwise 0.

Now, we revisit Theorem 3 presented in Section 5.1, and complete the proof that PoC-PKI ensures equivocation prevention.

Theorem 5. *$CT_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ satisfies security requirement $\Delta_{CT_{comp}}$-transparency under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, under the same assumptions as in Theorem 6.*

Proof. The only difference between the $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ and the $\overline{\mathrm{CT}_{\mathrm{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ schemes is that the adversary has an advantage in $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$, because the generalized Merkle tree scheme $\mathcal{GMT}$ is used to hash the sorted list of certificates logged by the logger. Lemma 3 shows that $\overline{\mathrm{CT}_{\mathrm{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ ensures $\Delta_{CT_{comp}}$-transparency under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, since $\mathcal{S}$ is secure; we now show that security in the $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ scheme also holds.

Assume to the contrary that although $\mathcal{GMT}$ is secure; there exists an adversary $\mathcal{A}$ that negates the claim that $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ ensures $\Delta_{CT_{comp}}$-transparency under the $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$ model, namely:

$$Pr\left[\begin{array}{l}(\mathsf{N_F}, R, t, out_{\mathcal{A}}) \leftarrow \mathbf{Exec}_{\mathcal{A},CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}(1^{\kappa}, \mathsf{N}) \\ \mathbf{Exp}^{\Delta\mathsf{TRA}}_{CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}(1^{\kappa}, \mathsf{N}, \mathsf{N_F}, R, t, out_{\mathcal{A}}) = \top\end{array}\right] \notin Negl(1^{\kappa}) \quad (49)$$

Since the only difference between $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ and $\overline{\mathrm{CT}_{\mathrm{comp}}}^{OSign(sk,\cdot)}_{\mathsf{N},\iota,vk}$ is the use of $\mathcal{E}$ to encrypt the individual secret information, it means that $\mathcal{A}$ uses this advantage to win the experiment.

Since $\Delta_{CT_{comp}} > MMD$, and following the implementation of $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Time}$ and $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Incoming}$ that state that every MMD time loggers update their Merkle tree root to accommodate the new certificate from the previous update, we know that after $\psi.\rho[\Delta\mathsf{TRA}].clk$ there was at least one Merkle tree update. Thus, either the logger includes ψ in that Merkle tree update, or not.

Since $\mathcal{A}$ wins with non-negligible probability in the $\mathbf{Exp}^{\Delta\mathsf{TRA}}_{\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}}$ game, then according to the game's description (Req. 5), two honest entities received ψ (a $\Delta_{CT_{comp}}$-transparent certificate) which they were unaware of, yet they cannot present IA or proof of misbehavior.

Since the honest authorities are unaware of ψ, then the logger could not have included ψ in the tree update. However, if it did not included ψ in the tree, then according to $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Audit}$, would have output a valid proof of misbehavior, by outputting the *signed* updated tree root that does not include ψ along with the logger's signature of ψ. Thus, since neither of these options is possible, this negates the possibility that such adversary exists.

Therefore, $\mathrm{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ also ensures $\Delta_{CT_{comp}}$-transparency under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$. □

E.3 Proof of VAS, NF, NFA, and UFAT

Lemma 8. *$CT_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ achieves Verifiable Attribution of Statements (VAS) for $\mathcal{S}$ under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, assuming security of $\mathcal{S}$, $\mathcal{GMT}$, and $\mathcal{H}$, under their respective definitions.*

Proof. Assume to the contrary, that there exists an adversary $\mathcal{A}$ that 'wins' in the verifiable attribution of statements experiment $\mathbf{Exp}^{\text{VAS}}_{CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}$, yet $\mathcal{S}$ is a secure signature scheme.

First, following the definition of honest entities from the execution process (Algorithm 1), $\mathcal{A}$ cannot change the execution of CT_{comp} algorithms by honest entities.

Second, following the convention described in §3.3.1, in every CT_{comp} algorithm where the $\mathcal{S}.\mathsf{Sign}$ algorithm is used, the algorithm 'automatically' produces an attribution statement of the signed data.

Therefore, with accordance to the definition of $\mathbf{Exp}^{\text{VAS}}_{CT^{\mathcal{S},\mathcal{GMT},\mathcal{H}}_{comp}}$ from Req 1, $\mathcal{A}$ is capable of producing a message m and a signature σ over m signed by an honest authority ι, *without* executing $\mathcal{S}.\mathsf{Sign}$ on ι with message m, with non-negligible probability.

However, if such adversary $\mathcal{A}$ exists, then according to definition of existential unforgeability (Def. 8), the following holds for $\mathcal{A}$

$$Pr\left[\mathbf{Exp}^{EU}_{\mathcal{A},\mathcal{S}}(1^{\kappa}) = 1\right] \notin Negl(1^{\kappa}) \tag{50}$$

thus contradicting the security of $\mathcal{S}$. □

Lemma 9. *$CT_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ achieves Verifiable non-frameability (NFA) under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, assuming security of $\mathcal{S}$, $\mathcal{GMT}$, and $\mathcal{H}$, under their respective definitions.*

Proof. The only misbehavior proof generated in CT_{comp} is via the Audit algorithm, when a $\Delta_{CT_{comp}}$-transparent certificate is inputted which the entity is supposed to be aware of, yet it does not. According to $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Time}$ and $\text{CT}_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}.\mathsf{Incoming}$, an honest logger always inform its monitors of new logged certificates, i.e., $\Delta_{CT_{comp}}$-transparent certificates, every MMD. Therefore, since according to $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$ messages that contain new $\Delta_{CT_{comp}}$-transparent certificates arrive to the respective monitors with Δ_{com} at most, honest monitors will always receive all newly logged certificates from honest loggers. □

Theorem 6. *$CT_{comp}{}^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ achieves Verifiable Attribution of Statements (VAS) for $\mathcal{S}$, Non-frameability (NF), No False Accusations (NFA) and Use First-Accuse Time (UFAT) under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$, assuming security of $\mathcal{S}$, $\mathcal{GMT}$, and $\mathcal{H}$, under their respective definitions.*

Proof. See Lemmas 8-9. □

E.4 Properties That CT_{comp} Does Not Achieves

Theorem 7. *CT_{comp} $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ does not ensure Δ-revocation transparency under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$.*

Proof. CT_{comp} does not have a *built-in* mechanism, e.g., Revocation Transparency, to ensure that a revoked certificate is known to all the monitor that watch a specific logger. □

Theorem 8. *CT_{comp} $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ does not ensure $\Delta-$equivocation detection under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$. Consequently, CT_{comp} $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ does not ensure equivocation prevention under model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$.*

Proof. Recall the definition of $\Delta-$equivocation detection (Requirement 11): Adversary $\mathcal{A}$ wins in the Δ-equivocation detection experiment $\mathbf{Exp}^{\mathsf{\Delta EQ\text{-}D}}_{\mathcal{P}}$ if it produces two valid, non-revoked certificates ψ, ψ' for the same identifier ($\psi.id = \psi'.id$) and for overlapping validity periods which both have the $\mathsf{\Delta EQ\text{-}D}$ property, where each certificate has different public information ($\psi.pub \neq \psi'.pub$), yet none of the entities in N was able to detect the equivocation before the Δ time of the $\mathsf{\Delta EQ\text{-}D}$ property has passed.

Consider a scenario where two loggers L_1, L_2 do not share an honest monitor, which model $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$ does not prohibits. In such scenario, logger L_1 can store a certificate ψ that conflicts with certificate ψ' which is stored on L_2. However, since there is no shared honest monitor that monitors both L_1 and L_2, there is no guarantee that at some point a monitor would detect this conflict. Moreover, even if the monitors that monitor L_1 gossip with the monitors of L_2, but they only gossip for consistency, i.e., the tree roots and not the actual certificates, then the conflict might not be detected. Hence, under the $\mathcal{M}^{CT_{comp}}_{\Delta_{com},\Delta_{clk}}$ model, $\mathsf{\Delta EQ\text{-}D}$ is not achieved.

Since equivocation can occur in CT_{comp} $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$, then CT_{comp} $^{\mathcal{S},\mathcal{GMT},\mathcal{H}}$ does not ensures equivocation prevention. □