Provable Security for PKI Schemes

It is possible to build a cabin with no foundations,

***but not a lasting building.* - Eng. Isidor Goldreich [28].**

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, USA
[3] Dept. of Computer Science, Trinity College, Hartford, CT, USA

Abstract. Public Key Infrastructure (PKI) schemes have significantly evolved since X.509, with more complex goals, e.g., transparency, ensuring security even against corrupt issuers. However, their security properties are still not rigorously defined or established. This is alarming, as PKIs are the basis for security of many critical systems, and security concerns exists even for well known and widely deployed PKI schemes, e.g., Certificate Transparency (CT) [36].
We present the first rigorous security specifications for PKI schemes, with properties such as *transparency, revocation transparency* and *non-equivocation*. Our goal is a general framework, applicable to most PKI designs. We apply it to CT as well as PoC-PKI, a 'proof-of-feasibility' PKI scheme, which has a significantly different design than CT, to demonstrate the flexibility of the framework. Lastly, we present theorems we proved for these two schemes.

1 Introduction

Public Key Infrastructure (PKI) provides an essential foundation to applications of public key cryptography, and crucial for security in open networks and systems. Since its introduction in 1988, the deployment of PKI was dominated by the X.509 standard [16]. X.509 was widely deployed by many protocols and systems. The most widespread is likely the TLS/SSL protocol [44], used to secure connections between web server and browser. This 'web-PKI' is critical for the secure use of the Web - and hence, a lucrative target for attackers, more than any other PKI deployment so far.

Unfortunately, the web-PKI deployment has inherent weaknesses. In particular, any CA is trusted to issue certificates for any domain [21]. This makes CAs a prime target for attackers. Over the years, we have seen many failures of this trusted-CA approach. For example, hackers stole the master keys of CAs [18,43] 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 [22]. 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 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. The proposals and designs include *Certificate Transparency (CT) [35, 36], Enhanced-CT [46], Sovereign Key [23], CONIKS [42], AKI [32], PoliCert [52], ARPKI [8], DTKI [55], CoSi [50, 51], IKP [39], CertCoin [26], PB-PKI [7], Catena [53]* and *CertLedger [34]*.

These designs have goals beyond those of X.509 - and, the goals and designs alike are significantly more complex than X.509. However, so far, these goals - and, often, even the designs - have not been rigorously defined; certainly, there are no proofs of security in most cases. This is alarming, as most practical applications of cryptography involve certificates and PKI; and the extensive efforts to prove security of cryptographic protocols, may be foiled when implemented over an unproven, and not even well defined, PKI. The concerns are even greater, considering the wide use and importance of PKIs, and the fact that attacks against PKI are not only a theoretical threat and have been done in practice.

Even for the much simpler X.509 PKI, there is no definition or proof of security; however, this may not be as critical, since for X.509, both definitions and proof are quite straightforward. Few works [9,15] prove security of cryptographic protocols based on parts of the X.509 design; however, there are signifcant aspects of X.509 they ignore, e.g., revocation. Works introducing PKIs with security against corrupt adversary typically offer only informal security arguments, with the following exceptions. Definitions and analysis for the logging properties of CT are presented in [17, 20]. ARPKI [8] and DTKI [55] provide automated symbolic analysis for their core, system-specific properties. No work defined (or analyzed) the generic PKI security properties.

Indeed, defining and proving security for PKI schemes is a non-trivial challenge, especially for schemes with defenses against corrupt CAs (and other entities). Another challenge is that PKI proposals vary greatly - even in the parties involved and in the communication and attack models. However, without definitions and proofs, there is a risk of using insecure schemes, and, consequently, of insecure deployments and implementations of PKI schemes. Furthermore, the lack of proper definitions and proofs makes it challenging to build (provably) secure systems, which depend on PKI schemes and their advanced features, and

to improve, compare and select a PKI scheme that best fits a specific application or scenario.

The lack of a precise framework for security of PKI schemes also makes it hard to evaluate and compare schemes, and to define new security properties, e.g, pertaining to privacy. It also prevents a modular design of schemes that achieve such new and advanced properties, by provable reductions to simpler, already-analyzed schemes.

We present the *PKI framework*, with well-defined correctness, safety and liveness requirements, allowing for reduction-based proofs of security for PKI schemes. The framework supports a wide range of PKI schemes, from X.509 to advanced, improved-security PKI schemes.

The PKI framework focuses on *defending against corrupt CAs*. We explicitly 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 [40], subsequently extended by [11,38], and others [10,29,31,37,47,56]. Our framework focuses on the complementary issue of dealing with parties that may arbitrarily misbehave by ensuring that their misbehavior can be discovered or prevented.

To define the framework, we reviewed and analyzed a number of existing PKI schemes and applications, to ensure that the framework reflects current PKIs and applications while remaining flexible to accommodate future PKIs, uses and extensions. We present game-based definitions, for the basic security requirements for PKI schemes designed for possible faulty CAs, namely, *accountability*, *Δ-transparency*, *non-equivocation* (detection and prevention), *revocation accountability* and *Δ-revocation transparency*. We map these requirements to existing PKIs in Table 1.

PKI is a fundamental part of most cryptographic systems; the surge of works on PKI in the recent years may indicate that more schemes, models and goals, may be identified in the future. The PKI framework is designed to support such flexibility, allowing different communication, synchronization and adversary models, and making it easy to define additional requirements. We also expect future works to investigate different variations and extensions of the framework itself, ranging from additional requirements, e.g., privacy, to challenges such as provably-secure compositions. Such variations were explored for other fundamental cryptographic mechanisms, such as encryption schemes.

To make sure that the PKI properties we identified are, in fact, feasible, and that our framework is sufficiently general and flexible to support different PKI schemes, we present the theorems we proved for two different PKI schemes. The first is a 'Proof of Concept' PKI scheme we developed and call PoC-PKI; the second is CT_{comp}, which is essentially *Certificate Transparency (CT)* with (simplified) specification of some essential missing details. We describe PoC-PKI and prove its security in the full version of this paper [4], and describe and analyse

CT, including CT_{comp}, in concurrent work [1]. The design of CT is very different from that of PoC-PKI, with different roles, adversary model and more, allowing us to fine-tune the framework to ensure it can support diverse schemes.

The PKI framework facilitate the definition of the adversary *model* for each PKI schemes, including the *entity-faults, communication and synchronization* assumptions, which may differ significantly between different PKI schemes. The interactions between the adversary and the different entities are defined using a precise *execution algorithm* (Alg. 1), and the properties are proven as long as the adversary operates within the assumed - and well defined - adversary model. The execution algorithm and adversary models, may be applicable for other applied, distributed cryptographic protocols. In particular, to analyze the CT and PoC-PKI PKI schemes, we present models for $f(n)$*-corruption-faults, bounded-delay-authenticated communication* and *bounded drift clocks*. These and similar models are implicitly assumed by many practical cryptographic systems; the execution algorithm and adversary models may be applicable therefore to other cryptographic systems, beyond PKI schemes. In particular, other models, e.g., passive adversaries, may be defined in similar manner.

The paper is **organized** as follows. Section 2 reviews the PKI landscape with respect to the requirements we identify in our framework and summarizes the related work. Section 3 presents the execution model of the framework and Section 4 presents the PKI framework and its correctness, safety and liveness requirements. Section 5 discusses how the framework can be applied in practice. We conclude and discuss future work in Section 6.

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, and then use these properties to compare proposed PKI schemes (see Table 1). See Section 4.4 for the game-based definitions.

2.1 'Basic' PKI Safety Properties

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)* [19] and *online certificate status protocol (OCSP)* [48]. 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.

2.2 Beyond X.509: security against trusted-yet-corrupted CAs

In Section 2.1, we discussed accountability and revocation accountability, the two 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 [6, 30, 33, 45].

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 [54], or the EFF SSL Observatory [24] 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 the 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 [12].

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 [42], 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* 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 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 Section 4; we also mention two additional properties, privacy, discussed informally in Section 2.2, and global name-spaces. Details of the evaluation of each scheme are included in the full version [4].

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

System [reference]	Safety requirements						Additional req.	
	ACC	ΔTRA	ΔEQ-D	EQ-P	ReACC	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 [53]	⊙[7]	⊙	⊙	⊙	⊙[7]	⊙	n/s	✓
CertCoin [26]	n/s	⊙	⊙	⊙	n/s	⊙	n/s	✓
PB-PKI [7]	n/s	⊙	⊙	⊙	n/s	⊙	⊙	✓
CoSi [51]	◐	⊙	⊙	⊙	n/s	n/s	n/s	✓
Enhanced-CT [46] DTKI [55] [3]	◐	⊙	⊙	n/s	◐	⊙	n/s	✓
AKI [32]	◐	⊙	⊙	n/s	◐	⊙	⊙	✓
CONIKS [42]	◐	n/s	⊙	n/s	◐	⊙	⊙	✗
ARPKI [8] [4]	◐	⊙	⊙	⊙	◐	⊙	n/s	✓
CertLedger [34]	◐	⊙	⊙	⊙	⊙	⊙	n/s	✓
Certificate Transparency (CT) [36]	◐	⊙[5]	⊙[5]	n/s	◐	n/s[6]		✓
CT_{comp} [1] (CT completed)	●	●	●	n/s	●	n/s	n/s	✓
PoC-PKI [4] (full version of 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 certificate-status freshness. [3]DTKI has symbolic proofs of some aspects. [4]ARPKI has symbolic proofs of some aspects. [5]Proofs of logging properties in [17,20]. [6]CT is extended to include revocation transparency in [35].

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 Section 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 [3], 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 and revocation accountability as 'intuitively true' for all systems, except for CertCoin, Catena, PB-PKI, and CoSi. Accountability and revocation accountability 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. [20] 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 [17], 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 [41]. 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 more complex to 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. However, we separated between properties which are not-supported, and properties which are claimed to be supported using some security arguments. Note that several 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 [35].

Finally, security requirements for cryptographic schemes often consider only *safety requirements*; however, in practice, *liveness* is an essential property - which is not always ensured. Liveness implies that operations terminate within bounded time, or eventually; e.g., whenever a certificate is issued (or revoked), the process will terminate. In the case of PKI schemes, liveness usually seems easy to achieve, but it is difficult to formalize, analyze and prove. Indeed, we believe that liveness 'intuitively holds' for *all* proposed PKI schemes as indicated in Table 1, however, *none* of the proposals identify liveness as a goal, and certainly none define it rigorously or prove that it holds. We correct this situation by presenting a liveness requirement that covers processing certificates, and specifically in this case, issuing, revoking and 'upgrading' certificates; see Section 4.5.

3 Entities and Execution Model

In this section, we define the execution algorithm for the framework we propose; see the next section for details specific to PKI schemes, such as algorithms specific to PKI schemes.

The execution algorithm is flexible, and applicable not just to PKI schemes but also to other systems, with different goals and functionalities. For example, the execution is invoked for a given scheme, denoted $\mathcal{P}$; this may be a PKI scheme - but could also be any other scheme - the execution is not PKI specific.

The flexibility of the execution algorithm also extends to support for a variety of network, synchronization and adversary models. In particular, the execution model allows synchronous, partially synchronous or asynchronous communication models, and adversarial models ranging from passive eavesdropper to active adversaries who can modify messages and launch man-in-the-middle attacks.

3.1 Entities, Authorities and basic operations

In PKI, there are two types of entities: *authorities* and *clients*, also referred to as *relying parties*. Clients rely on the authorities to obtain and manage their certificates and for any other certificate-related requests and queries. 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. For simplicity and generality, the model treats all authorities equally, although this does not prevent implementations where authorities have specific roles (e.g., CA, RA, logger, auditor, etc.). When a system makes such distinctions between authorities, they are captured in the execution model as an output value $Out_{\mathcal{A}}$,(see Algorithm 1). We denote the set of all authorities in the system as $\mathsf{N} = \{1, \ldots, n\}$, where n is the number of all authorities.

Authorities may interact with one another to perform certain actions. Each authority has a local clock Clk and a local state S, e.g. containing issued certificates, and the following information: *i)* $S.\iota$: the unique-identifier of each entity, *ii)* $S.PrivInfo$: private (secret) information, and *iii)* $S.PubInfo$: public information.

Each authority supports several operations specific to PKI schemes as detailed in Section 4. Below we define three operations which are not PKI-specific and are performed locally by each authority: Gen, Time and $\mathsf{Incoming}$.

- $\mathsf{Gen}(1^\kappa) \rightarrow (PrivInfo, PubInfo)$: The Gen algorithm allows to initialize the authority and generate the necessary set up information (e.g., cryptographic keys) for each individual authority. The algorithm takes as input a security parameter 1^κ and outputs private information $PrivInfo$ and public information $PubInfo$.
- $\mathsf{Time}(S, Clk) \rightarrow (S', \{m_i\}_{i\in\mathsf{N}}, out)$: The Time algorithm performs operations which are time-dependent. The algorithm takes as input a local state S and local clock Clk. The algorithm outputs the modified state S', a set of messages $\{m_i\}_{i\in\mathsf{N}}$ for other authorities and output out.
- $\mathsf{Incoming}(S, Clk, \{\widetilde{m}_i\}_{i\in\mathsf{N}}) \rightarrow (S', \{m_i\}_{i\in\mathsf{N}}, out)$: The $\mathsf{Incoming}$ algorithm process and handles incoming messages from other authorities. The algorithm takes as input a local state S, a local clock Clk and a set of messages $\{\widetilde{m}_i\}_{i\in\mathsf{N}}$. The algorithm outputs the modified state S', a set of messages $\{m_i\}_{i\in\mathsf{N}}$, and output out.

In addition to these three local operations, we define a *system initiation* operation, $\mathsf{GroupGen}$, which generates common public information, such as a system-wide public key or a set of 'root' public keys. The $\mathsf{GroupGen}$ operation is invoked as a part of our execution model; to implement this in a real system, $\mathsf{GroupGen}$ could be run by a trusted third party, or using an appropriate multi-party computation protocol. The $\mathsf{GroupGen}$ algorithm takes as input a security parameter 1^κ, the set of authorities N and a set of public information $\{PubInfo_i\}_{i\in\mathsf{N}}$. $\mathsf{GroupGen}$ has only one output, the public group information $PubInfo$. Namely, $\mathsf{GroupGen}(1^\kappa, \mathsf{N}, \{PubInfo_i\}_{i\in\mathsf{N}}) \rightarrow PubInfo$.

3.2 Execution Model

The execution model is defined by the $\mathbf{Exec}_{\mathcal{A},\mathcal{P}}$ algorithm (see Algorithm 1), where $\mathcal{A}$ and $\mathcal{P}$ represent the adversary and the specific implementation of a PKI system, respectively. This algorithm takes as input a security parameter 1^κ and the set of authorities N. The **Exec** algorithm begins with an *initialization phase* (lines 1-7). First (line 1), the adversary selects the set of faulty authorities $\mathsf{N_F}$. The local state of each authority is initialized using Gen (line 2) but the adversary is allowed to override the state for each authority in $\mathsf{N_F}$ (line 3). We note that the local clock of each authority is initialized by the adversary as the specific clock synchronization rules must conform to the restrictions expressed in the *model predicate* $\mathcal{M}$ (see Section 3.3). Then (line 4), $\mathsf{GroupGen}$ algorithm is used to generate the public information $PubInfo$, known by all authorities. Finally (line 7), the adversary is provided with all public information and invoked to set the inputs for the first round of the execution phase. The *execution phase* (lines 8-9) is a loop of *rounds*, where each round is labeled with an incremental round number t. In each round, each authority $i \in \mathsf{N}$ handles three events. First (line 8.2.1), we invoke one of the functions of $\mathcal{P}$, as selected by the adversary; the function $\mathcal{P}.Alg_i^t$ is selected by the variable Alg_i^t, which the adversary sets in line 7 (initialization) or afterwards line 8.3 (execution). This is intended to invoke the algorithms in PKI which are not run by the execution process directly (i.e., all but Gen, $\mathsf{GroupGen}$, Time, and $\mathsf{Incoming}$). Second (line 8.2.2), we invoke the $\mathcal{P}.\mathsf{Incoming}$ algorithm to handle incoming messages $\widetilde{m}_{i,j}^{t-1}$ from other authorities $j \in \mathsf{N}$ (in previous round $t-1$). Next (line 8.2.3), we invoke the $\mathcal{P}.\mathsf{Time}$ algorithm to handle time-based events. Each of the algorithms is provided with access to the current local state and clock of the appropriate authority, and after it is finished executing, it outputs a modified state for that authority as well as a set of messages for other authorities (possibly empty). We complete the round by invoking the adversary (line 8.3), who receives as input all the messages sent in this round $\{m_{i,j}\}_{i,j\in\mathsf{N}}$, and determines which messages would be received in the next round $\{\widetilde{m}_{i,j}^t\}_{i,j\in\mathsf{N}}$; this allows for an active adversary, or to enforce $\widetilde{m}_{i,j}^t = m_{i,j}$ for eavesdropping adversary. The execution rounds repeat until the adversary decides to abort. When the adversary aborts the execution (line 9), the execution concludes, and outputs (line 11) four values: $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R]$, where t is the number of rounds in the execution, $\mathsf{N_F}$ is the set of faulty authorities, $Out_{\mathcal{A}}$ is the transcript of adversary's choices, and R is the *run*, namely, the 'transcript' of the execution (line 10).

3.3 Fault, Communication and Synchronization Models

Our execution model is general, flexible and suitable for schemes designed for different adversary models. We define a *model predicate* $\mathcal{M}$ over each execution run, which returns *true* if, and only if, the run conforms to the intended adversarial model. Namely, given a set of input parameters $\xi = \{\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R\}$ which provides the execution details, the model $\mathcal{M}$ is defined as:

$$\mathcal{M}(\xi) = \{\mathcal{M}^{FAULT}(\xi) \wedge \mathcal{M}^{COM}(\xi) \wedge \mathcal{M}^{SYNC}(\xi)\}$$

where $\mathcal{M}^{FAULT}$ is the fault model function, $\mathcal{M}^{COM}$ is the communication model function, and $\mathcal{M}^{SYNC}$ is the synchronization model function.

This approach allows each system to precisely define its adversarial model by specifying the details of the fault, communication, and synchronization models. While many systems use common and well-understood assumptions (e.g., a honest majority for the fault model, a synchronous network for the communication model, or fully synchronized clocks for the synchronization model), others may need to define their own variants and being able to do so in a formal way is necessary. In this work, we apply this framework to our proof-of-concept system PoC-PKI that follows a standard adversarial model but also to Certificate Transparency which requires different assumptions. We present the model predicate $\mathcal{M}$ for PoC-PKI and CT in Section 5. Below we provide formal descriptions of some of the standard fault, communication and synchronization model functions.

Faults Model Function $\mathcal{M}^{FAULT}$. The fault model function specifies which sets of authorities the adversary may control. One commonly used faults model limits only the *size* of the set of faulty authorities, as a function f of the total number of authorities. We refer to this particular faults model as $\mathcal{M}_f^{NumF}$, where $f : \mathbb{N} \to \mathbb{N}$ bounds the number of faulty authorities as a function of the total number of authorities, namely:

$$\mathcal{M}_f^{NumF}(\xi) = \{TRUE \; iff \; |\xi.\mathsf{N_F}| \leq f\left(|\xi.\mathsf{N}|\right)\} \tag{1}$$

In particular, our proof-of-concept PKI, PoC-PKI, uses $\mathcal{M}_f^{NumF}$ with f defined as: $f(n) = \lfloor (n/3) \rfloor$. Namely, the faulty entities can arbitrarily misbehave so long the adversary controls less than a third of all authorities. The way that we let the adversary control the faulty authorities is by allowing the adversary to control their public (and private) information, see line 3 of the execution, and also to modify messages they send and receive, as we define below; this essentially allows adversary to 'act as' all of these authorities.

Communication Model Function $\mathcal{M}^{COM}$. Below we present a bounded-delay communication model with authenticated channels function $\mathcal{M}_{\Delta_{com}}^{BD-COM}$, where every message received by honest entity was sent at most Δ_{com} earlier (captured by $\mathcal{M}_{\Delta_{com}}^{BD-RCV}$) and every message sent by an honest entity is received at most Δ_{com} later (captured by $\mathcal{M}_{\Delta_{com}}^{RC-SENT}$).

$$\mathcal{M}_{\Delta_{com}}^{BD-RCV}(\xi) = \{ \; \Big((\forall i, j \in \xi.\mathsf{N} - \xi.\mathsf{N_F}, \tau \in \mathbb{N})\xi.R.\tilde{m}_{i,j}^{\tau} \neq \varnothing \Rightarrow$$
$$\exists \tau' \; s.t. \; Clk_i^{\tau} - \Delta_{com} \leq Clk_j^{\tau'} < Clk_i^{\tau} \; \wedge \; \xi.R.\tilde{m}_{i,j}^{\tau} = \xi.R.m_{i,j}^{\tau'}\Big)\}$$

$$\mathcal{M}_{\Delta_{com}}^{BD-SENT}(\xi) = \{ \; \Big((\forall i, j \in \xi.\mathsf{N} - \xi.\mathsf{N_F}, \tau \in \mathbb{N})\xi.R.m_{i,j}^{\tau} \neq \varnothing \Rightarrow$$
$$\exists \tau' \; s.t. \; Clk_i^{\tau} < Clk_j^{\tau'} \leq Clk_i^{\tau} + \Delta_{com} \; \wedge \; \xi.R.\tilde{m}_{i,j}^{\tau'} = \xi.R.m_{i,j}^{\tau}\Big)\}$$

Finally, the communication model function is defined as:

$$\mathcal{M}_{\Delta_{com}}^{BD-COM}(\xi) = \{\mathcal{M}_{\Delta_{com}}^{BD-RCV}(\xi) \wedge \mathcal{M}_{\Delta_{com}}^{BD-SENT}(\xi)\}$$

Synchronization Model Function $\mathcal{M}^{SYNC}$. The following synchronization model function $\mathcal{M}^{BD-SYNC}_{\Delta_{clk}}$ defines a bounded-drift clock synchronization model, i.e., the time difference between the local clocks of all entities is at most Δ_{clk}. As a special case ($\Delta_{clk} = 0$), this function defines a model where the local clocks are fully synchronized, i.e., there is no difference between entities' clocks.

$$\mathcal{M}^{BD-SYNC}_{\Delta_{clk}}(\xi) = \{\ \left((\forall i, j \in \xi.\mathsf{N}, \tau \in \mathbb{N}) | \xi.R.Clk_i^\tau - \xi.R.Clk_j^\tau| \leq \Delta_{clk}\right)\ \}$$

Algorithm 1 $\mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$

// Adversary picks state, set of faulty authorities
1: $\left(\mathsf{N_F}, S^0_{\mathcal{A}}, \{Clk^0_i\}_{i\in\mathsf{N}}\right) \leftarrow \mathcal{A}(\mathsf{N}, 1^\kappa)$

// Initialize local variables for all authorities
2: $\forall i \in \mathsf{N} : S^0_i \leftarrow \perp, \left(S^0_i.PrivInfo, S^0_i.PubInfo\right) \leftarrow \mathcal{P}.\mathsf{Gen}_i\,(1^\kappa)$

// Adversary sets each faulty entity's *PrivInfo* and *PubInfo*
3: $\left(S^0_{\mathcal{A}}, \{(S^0_i.PrivInfo, S^0_i.PubInfo)\}_{i\in\mathsf{N_F}}\right) \leftarrow \mathcal{A}\left(\{S^0_{\mathcal{A}}, S^{\prime 0}_i.PrivInfo\}_{i\in\mathsf{N_F}}, \{S^{\prime 0}_j.PubInfo\}_{j\in\mathsf{N}}\right)$

// Generate public group information
4: $PubInfo \leftarrow \left(\mathcal{P}.\mathsf{GroupGen}(1^\kappa, \mathsf{N}, \{S^{\prime 0}_i.PubInfo\}_{i\in\mathsf{N}}), \{S^{\prime 0}_i.PubInfo\}_{i\in\mathsf{N}}\right)$

5: $\forall i \in \mathsf{N} : S^0_i.PubInfo \leftarrow PubInfo$

// Initialize the round indicator t
6: $t \leftarrow 0$

// Invoking the adversary before the first round
7: $[\{Alg^1_i, Inp^1_i, Clk^1_i\}_{i\in\mathsf{N}}, \{\widetilde{m}^0_{i,j}\}_{i,j\in\mathsf{N}}, S^1_{\mathcal{A}}] \leftarrow\ \mathcal{A}(S^0_{\mathcal{A}}, PubInfo)$

// Execution phase:
8: **repeat**

8.1: $t \leftarrow t + 1$

8.2: $\forall i \in \mathsf{N}$:

// Invoke instructions for the current round
8.2.1: $\left(S_1, \{m^1_{i,j}\}_{j\in\mathsf{N}}, Out_1\right) \leftarrow \mathcal{P}.Alg^t_i\left(S^{t-1}_i, Clk^t_i, Inp^t_i\right)$

8.2.2: $\left(S_2, \{m^2_{i,j}\}_{j\in\mathsf{N}}, Out_2\right) \leftarrow \mathcal{P}.Incoming\left(S_1, Clk^t_i, \{\widetilde{m}^{t-1}_{i,j}\}_{j\in\mathsf{N}}\right)$

8.2.3: $\left(S^t_i, \{m^3_{i,j}\}_{j\in\mathsf{N}}, Out_3\right) \leftarrow \mathcal{P}.Time\left(S_2, Clk^t_i\right)$

// Aggregate all outputted values during round t into a single set
8.2.4: $Out^t_i \leftarrow Out_1 \cup Out_2 \cup Out_3$, $(\forall j) m_{i,j} = \bigcup_{\ell=1}^3 m^\ell_{i,j}$

// Adversary sees and controls messages, and selects operations for next round
8.3: $[\{Alg^{t+1}_i, Inp^{t+1}_i, Clk^{t+1}_i\}_{i\in\mathsf{N}}, \{\widetilde{m}^t_{i,j}\}_{i,j\in\mathsf{N}}, S^{t+1}_{\mathcal{A}}, Abort, Out_{\mathcal{A}}] \leftarrow\ \mathcal{A}(\{m_{i,j}\}_{i,j\in\mathsf{N}}, S^t_{\mathcal{A}})$

// Repeat until the adversary indicates termination via the *Abort* indicator
9: **until** $Abort \neq \perp$

// Output:
10: $R \leftarrow \left\{ \left\{ Alg^{\hat{t}}_i, Inp^{\hat{t}}_i, Out^{\hat{t}}_i, Clk^{\hat{t}}_i \right\}_{i\in\mathsf{N}}, \left\{ m^{\hat{t}}_{i,j}, \widetilde{m}^{\hat{t}}_{i,j} \right\}_{i,j\in\mathsf{N}} \right\}_{\hat{t}\in\{1\ldots t\}}$

// The algorithm outputs the number of rounds t, the set of faulty authorities $\mathsf{N_F}$, the transcript $Out_{\mathcal{A}}$ detailing the adversary's choices, and the transcript R of all execution details.
11: Return $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R]$

4 The PKI Framework

In this section, we describe the PKI framework. We first provide an overview of the framework and then define the correctness, safety and liveness requirements.

4.1 Certificates and Functionalities

A PKI scheme $\mathcal{P}$ associates an identity identifier id with some public information pub. The association of (id, pub) is achieved through a *certificate*, and authorities issue certificates, add attributes to certificates, or revoke certificates in response to clients' requests using algorithms PKI defines (see Section 4.2). A certificate ψ contains:

$$\psi = (id, pub, sd, ed, \rho)$$

where:

- $\psi.id$: identifier of the entity for which ψ was issued.
- $\psi.pub$: public information associated with $\psi.id$, e.g., a cryptographic public key.
- $\psi.sd$: start of certificate validity period.
- $\psi.ed$: end of certificate validity.
- $\psi.\rho$: certificate's attributes and details.
  - $\psi.\rho[attr].\sigma$: 'attestation' that ψ has attribute $attr$.
  - $\psi.\rho[attr].\iota$: identity of the authority who attests for $attr$.
  - $\psi.\rho[attr].clk$: the local time when the $attr$ attribute was added to ψ.

When a client wishes to obtain a certificate to associate some id with some pub, she contacts some authority $\iota \in \mathsf{N}$. Authority ι verifies the legitimacy of the client's request, and if warranted, produces a certificate ψ using the $\mathcal{P}.\mathsf{Issue}$ algorithm. We stress that this verification process is not prescribed by this framework. We assume that each implementation of PKI will specify this verification process and define any additional constraints on CAs (e.g., naming constraints). The $\mathcal{P}.\mathsf{WasValid}$ algorithm can be used to check if a given certificate is valid. This algorithm does not depend on any state so it can be used by any part in the PKI, and each implementation can define its specific $\mathcal{P}.\mathsf{WasValid}$ to reflect its notion of a certificate validity. Authorities also support the $\mathcal{P}.\mathsf{Query}$ algorithm, which outputs a list of all *valid* certificates issued for some id, if such certificates are known to the authority.

Once a certificate is issued, it can be then *upgraded* using the $\mathcal{P}.\mathsf{Upgrade}$ algorithm to receive an *attribute*, i.e., an 'added-on', signed 'endorsement' of a certificate, such as *signed certificate timestamp (SCT)* in the Certificate Transparency PKI [2]. We denote the entire set of possible attributes as $\mathsf{AttrSet}$ and each implementation may define its own $\mathsf{AttrSet}$, providing for flexibility and customization. We define $\mathsf{AttrSet} = \{\mathsf{ACC}, \Delta\mathsf{TRA}, \Delta\mathsf{EQ\text{-}D}, \mathsf{EQ\text{-}P}, \mathsf{REV}, \Delta\mathsf{ReTRA}\}$ to support the safety requirements of accountability, Δ-transparency, equivocation detection, equivocation prevention, revocation accountability, and Δ-revocation transparency, respectively. Each attribute reflects some property of a certificate and carries a specific meaning in terms of its security. For example, a certificate with the ACC attribute is *accountable*, that is, the issuing authority can be

identified. A Δ-*transparent* (ΔTRA attribute) certificate can be assumed to be known to all honest authorities in the system after some point in time defined by Δ while an *unequivocal* (EQ-P attribute) certificate carries a guarantee that another valid unequivocal certificate for the same identifier will not be issued unless the initial one was expired or revoked. We view revocation as an attribute too and discuss it next.

An authority may invalidate a certificate before its expiration date using the $\mathcal{P}$.Revoke algorithm, which generates a revoked certificate ψ_r, i.e., a certificate that has a revocation attribute added that indicates the level of revocation: (REV for accountable revocation and ΔReTRA for Δ-transparent revocation, or both). As before, each implementation should define its specific revocation rules. An expired certificate need not be revoked. The $\mathcal{P}$.IsRevoked algorithm is used to check the revocation status of a certificate: if ψ is revoked, then $\mathcal{P}$.IsRevoked returns its revoked version ψ_r; if ψ is not revoked, then $\mathcal{P}$.IsRevoked returns a certificate with the NR attribute that contains a proof of non-revocation until the current time; in any other case, e.g., if the state of the certificate is not determined or known, the algorithms returns $\perp$.

Pending Certificates. In our execution model, all PKI algorithms return an *immediate* response once invoked. However, there are three algorithms that might not be able to immediately return a *final* response: the $\mathcal{P}$.Issue algorithm, the $\mathcal{P}$.Revoke algorithm, and the $\mathcal{P}$.Upgrade algorithm. These algorithms might not be solely depended on local operations; for example, they might require interaction with other authorities before they are able to decide whether to issue/revoke/upgrade a certificate. Therefore, we introduce the notion of *pending* certificates. Namely, when one of these three algorithms is unable to produce the immediate *final* (expected) response, the algorithm produces a pending certificate ψ_p instead, which contains a *commitment* that specifies *when* the final response will be ready with respect to a finite, system parameter Δ. At some time after the Δ time period has passed, the client requests the authority for the final response. The authority executes the $\mathcal{P}$.Upgrade algorithm with the pending certificate ψ_p as an input, and delivers to the client the output of $\mathcal{P}$.Upgrade, which is the *final* output. The final output is either the 'real' certificate, i.e., the expected *non-pending* upgraded certificate, or, the output is $\perp$, 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, the final response can only be obtained using the $\mathcal{P}$.Upgrade algorithm.

4.2 PKI Algorithms

Each scheme $\mathcal{P}$ defined in the PKI framework provides the following algorithms:

$$\mathcal{P} = (\mathsf{Gen}, \mathsf{GroupGen}, \mathsf{Issue}, \mathsf{Upgrade}, \mathsf{WasValid}, \\ \mathsf{Query}, \mathsf{Revoke}, \mathsf{IsRevoked}, \mathsf{Time}, \mathsf{Incoming})$$

The PKI framework follows the execution model described in Section 3, which defines Gen and GroupGen (for key generation), Time (for handling time-based events) and Incoming (for handling incoming messages).

We start by describing the *stateless* WasValid algorithm.

WasValid$(\psi, [attr, tms]) \rightarrow \top / Pending / \bot$: The algorithm takes as input a certificate ψ, optional attribute $attr$ and an optional timestamp tms. If ψ is a valid certificate and has the $attr$ attribute (or if no $attr$ is provided), 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 some point in time defined by 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 $\bot$.

WasValid is *stateless*, hence, it may be run by everyone, including relying parties, the adversary, and in particular, can be used in security games to enforce validity requirements. All other algorithms always receive as input the local state S and current local clock clk, and output the modified state S', output messages $\{m_i\}_{i \in \mathsf{N}}$, and output Out. To avoid clutter, we *abuse notation* and do not explicitly write these inputs and outputs, but only other inputs and outputs, which are unique to each algorithm. We now describe the remaining algorithms: Issue, Upgrade, Revoke, IsRevoked and Query.

Issue$(id, pub, sd, ed) \rightarrow \psi / \psi_p / \bot$: 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 after sd and before 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 $\bot$.

Upgrade$(\psi, attr, [\alpha]) \rightarrow \psi' / \psi_p / \bot$: The algorithm takes as input a certificate ψ, an upgrade attribute $attr$ and additional optional information α. If the upgrade request is valid, the algorithm outputs an upgraded certificate ψ' based on ψ with the $attr$ attribute. If the algorithm requires further operations, the algorithm outputs a pending certificate ψ_p. If the input is a pending certificate ψ_p, then after the Δ-defined time, the algorithm returns a final version ψ of ψ_p. If the upgrade fails, the algorithm returns $\bot$ and provide an explanation why in the output parameter Out.

Revoke$(\psi) \rightarrow \psi_r / \psi_p / \bot$: The algorithm takes as input a certificate ψ, and outputs a revoked certificate ψ_r, a pending-revoked certificate ψ_p, or failure indicator $\bot$.

IsRevoked$(\psi) \rightarrow \psi' / \psi_r / \bot$: 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 NR attribute. If ψ was already revoked, the algorithm returns the revoked certificate ψ_r. In any other case, the algorithm returns $\bot$.

Query$(id) \rightarrow \{\psi\} / \bot$: The algorithm takes as input an identity id and returns the set of certificates $\{\psi\}$ that are associated with id. If such certificates do not exist, the algorithm returns $\bot$.

4.3 Correctness Requirements

We next describe the PKI correctness requirements for issuing, revoking, upgrading, and the revocation status check operation. The requirements specify that if an operation does not fail (return $\perp$), then it should produce the requested output as specified by each algorithm. We first state that the results of the issue operation is a 'correct' certificate, namely a certificate with the requested *core*. Given a certificate ψ, its *core* is $Core(\psi) = (\psi.id, \psi.pub, \psi.sd, \psi.ed)$, i.e., the entire certificate except for its attributes. For convenience, we denote $Core(\perp) = \perp$.

Requirement 1. *PKI scheme $\mathcal{P}$ satisfies correctness of certificate issuance if:*

$$(\forall\ id, pub, sd, ed; \psi \leftarrow \mathcal{P}.\mathsf{Issue}(id, pub, sd, ed)\ s.t.\ \psi \neq \perp) \Rightarrow$$
$$[(\mathcal{P}.\mathsf{WasValid}(\psi) \neq \perp) \wedge (\psi.id = id) \wedge (\psi.pub = pub)]$$

Requirement 2. *PKI scheme $\mathcal{P}$ satisfies correctness of certificate revocation if:*

$$(\forall\ \psi\ ;\ \psi_r \leftarrow \mathcal{P}.\mathsf{Revoke}(\psi)\ s.t.\ Core(\psi) = Core(\psi_r)) \Rightarrow$$
$$\mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \neq \perp$$

Requirement 3. *PKI scheme $\mathcal{P}$ satisfies correctness of certificate upgrade if:*

$$\big(\forall\ \psi, attr, \alpha; \psi' \leftarrow \mathcal{P}.\mathsf{Upgrade}(\psi, attr, \alpha)\ s.t.$$
$$Core(\psi) = Core(\psi')\big)\ \Rightarrow\ \mathcal{P}.\mathsf{WasValid}(\psi', attr) \neq \perp$$

Correctness of certificate revocation status should return either a revoked or a non-revoked certificate, with the same core. The proof of non-revocation is added to the certificate in the dedicated NR attribute.

Requirement 4. *PKI scheme $\mathcal{P}$ satisfies correctness of certificate revocation status if:*

$$(\forall\ \psi\ ;\ \psi' \leftarrow \mathcal{P}.\mathsf{IsRevoked}(\psi)\ s.t.\ Core(\psi) = Core(\psi')) \Rightarrow$$
$$\mathcal{P}.\mathsf{WasValid}(\psi', \mathsf{REV})\ \vee \mathcal{P}.\mathsf{WasValid}(\psi', \mathsf{NR})$$

4.4 Safety Requirements

PKI schemes, as many other protocols, have two kinds of security requirements, safety and liveness, which we now define for PKI. For each security requirement ξ, we define a corresponding experiment $\mathbf{SecExp}^{\xi,\mathcal{M}}_{\mathcal{A},\mathcal{P}}$, where model $\mathcal{M}$ is as described in subsection 3.3, and use it to define the requirement using the following 'generic' definition.

Definition 1. *PKI scheme $\mathcal{P}$* satisfies a security requirement ξ under model $\mathcal{M}$*, if for every PPT adversary $\mathcal{A}$ and for every set* N *holds:*

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

Algorithm 2 $\mathbf{SecExp}^{\mathsf{ACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$

// Execute the adversary
1: $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$
// Did $\mathcal{A}$ follow the model?
2: **if** $\mathcal{M}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \perp$ **then** Return $\perp$
// Extract algorithms and inputs from R
3: $\left\{Alg_i^{\hat{t}}, Inp_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
// Extract from the adversary's output
4: $(\psi, \iota) \leftarrow Out_{\mathcal{A}}$
5: Return:
// ψ is a valid accountable certificate
5.1: $\mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{ACC}) \ \wedge$
// ψ was issued by honest authority ι
5.2: $\iota = \psi.\rho[\mathsf{ACC}].\iota \ \wedge \ \iota \in \mathsf{N} - \mathsf{N_F} \ \wedge$
// However, ι did not issue ψ
5.3: $\nexists \ \hat{t} \ s.t. \ Alg_\iota^{\hat{t}} = \mathcal{P}.\mathsf{Issue} \ \wedge$
$Inp_\iota^{\hat{t}} = (\psi.id, \psi.pub, \psi.sd, \psi.ed)$

Algorithm 3 $\mathbf{SecExp}^{\mathsf{ReACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$

// Execute the adversary
1: $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$
// Did $\mathcal{A}$ follow the model?
2: **if** $\mathcal{M}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \perp$ **then** Return $\perp$
// Extract algorithms and inputs from R
3: $\left\{Alg_i^{\hat{t}}, Inp_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
// Extract from the adversary's output
4: $(\psi_r, \iota) \leftarrow Out_{\mathcal{A}}$
5: Return:
// ψ_r is a valid revoked certificate
5.1: $\mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \ \wedge$
// ψ_r was issued by an honest authority
5.2: $\iota = \psi_r.\rho[\mathsf{ACC}].\iota \ \wedge \ \iota \in \mathsf{N} - \mathsf{N_F} \ \wedge$
// However, $\mathcal{P}.\mathsf{Revoke}$ was not invoked on ι
5.3: $\nexists \ \hat{t} \ s.t. \ Alg_\iota^{\hat{t}} = \mathcal{P}.\mathsf{Revoke} \ \wedge$
$Inp_\iota^{\hat{t}} = \psi \ \wedge Core(\psi) = Core(\psi_r)$

Requirement 5. *Accountability (*ACC*). Adversary $\mathcal{A}$ wins in the accountability experiment* $\mathbf{SecExp}^{\mathsf{ACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$ *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.*

The $\mathbf{SecExp}^{\mathsf{ACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$ game initializes the experiment by calling the **Exec** algorithm, which simulates the adversary. In return, the **Exec** algorithm outputs the number of rounds t simulated in **Exec**, the set of corrupted authorities $\mathsf{N_F}$, the transcript of adversary's choices $Out_{\mathcal{A}}$, and R, the transcript of the simulation (lines 1,3). The experiment verifies that the run followed the model $\mathcal{M}$ (line 2); if not, it aborts and outputs $\perp$. $\mathcal{M}$ takes as input parameter $\xi = \{\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R\}$. For a run which follows the model, we 'parse' R to find the operations and inputs $\left\{Alg_i^{\hat{t}}, Inp_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}}$ (line 3). From $Out_{\mathcal{A}}$ (line 4), we extract the certificate returned by the adversary (ψ) and the selected honest authority (ι).

The adversary *wins* if ψ is a valid accountable certificate issued by the honest authority ι (lines 5.1-5.2), yet ι was never instructed to execute the $\mathcal{P}.\mathsf{Issue}$ algorithm along with the inputs $\psi.id$, $\psi.pub$, $\psi.sd$, $\psi.ed$ (line 5.3).

Requirement 6. *Revocation accountability (*ReACC*). Adversary $\mathcal{A}$ wins in the revocation accountability experiment* $\mathbf{SecExp}^{\mathsf{ReACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$ *if it produces a valid revoked certificate ψ_r issued by an honest authority $\psi_r.\rho[\mathsf{ACC}].\iota$, where $\psi_r.\rho[\mathsf{ACC}].\iota$ did* not *revoke ψ_r. See Algorithm 3.*

Algorithm 4 $\mathbf{SecExp}^{\Delta\mathsf{TRA},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa,\mathsf{N})$

$[t,\mathsf{N_F},Out_{\mathcal{A}},R] \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa,\mathsf{N})$
if $\mathcal{M}(\mathsf{N},\mathsf{N_F},Out_{\mathcal{A}},R)=\perp$ **then** Return $\perp$
$\left\{Alg_i^{\hat{t}},Inp_i^{\hat{t}},Out_i^{\hat{t}},Clk_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
$(\psi,\iota) \leftarrow Out_{\mathcal{A}}$
Return:
// ψ is a valid transparent certificate
5.1: $\mathcal{P}.\mathsf{WasValid}(\psi,\Delta\mathsf{TRA}) \wedge$
// Both ι and $\psi.\rho[\Delta\mathsf{TRA}].\iota$ are honest
5.2: $\iota,\psi.\rho[\Delta\mathsf{TRA}].\iota \in \mathsf{N}-\mathsf{N_F} \wedge$
// ι is not aware of ψ although it should
5.3: $Clk_\iota^t \geq \psi.\rho[\Delta\mathsf{TRA}].clk+\Delta \wedge$
5.4: $\left[Alg_\iota^t,Inp_\iota^t\right]=[\mathcal{P}.\mathsf{Query},\psi.id] \wedge$
5.5: $\nexists\psi' \in Out_\iota^t \; s.t. \; Core(\psi)=Core(\psi')$

Algorithm 5 $\mathbf{SecExp}^{\Delta\mathsf{ReTRA},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa,\mathsf{N})$

$[t,\mathsf{N_F},Out_{\mathcal{A}},R] \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa,\mathsf{N})$
if $\mathcal{M}(\mathsf{N},\mathsf{N_F},Out_{\mathcal{A}},R)=\perp$ **then** Return $\perp$
$\left\{Alg_i^{\hat{t}},Inp_i^{\hat{t}},Out_i^{\hat{t}},Clk_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
$(\psi_r,x,\hat{t},\iota) \leftarrow Out_{\mathcal{A}}$
Return:
// ψ_r is a valid revoked certificate
5.1: $\mathcal{P}.\mathsf{WasValid}(\psi_r,\mathsf{REV}) \wedge$
// ψ_r claims revocation transparency
5.2: $\mathcal{P}.\mathsf{WasValid}(\psi_r,\Delta\mathsf{ReTRA}) \wedge$
// Both ι and $\psi_r.[\Delta\mathsf{ReTRA}].\iota$ are honest
5.3: $\iota,\psi_r.[\Delta\mathsf{ReTRA}].\iota \in \mathsf{N}-\mathsf{N_F} \wedge$
// ι is not aware of ψ_r although it should
5.4: $\left[Alg_\iota^t,Inp_\iota^t\right]=[\mathcal{P}.\mathsf{Query},\psi_r.id] \wedge$
5.5: $(\nexists\psi \in Out_\iota^t \; s.t. \; Core(\psi_r)=Core(\psi) \wedge$
$\mathcal{P}.\mathsf{WasValid}(\psi,\mathsf{REV})) \wedge$
5.6: $Clk_\iota^{\hat{t}} \geq \psi_r.\rho[\Delta\mathsf{ReTRA}].clk+\Delta$

The $\mathbf{SecExp}^{\mathsf{ReACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa,\mathsf{N})$ game executes adversary $\mathcal{A}$ using **Exec**, verifies that $\mathcal{A}$ followed the model $\mathcal{M}$, and extracts the execution details as described in $\mathbf{SecExp}^{\mathsf{ACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$.

The adversary wins the game if two requirements are met. First, ψ_r must be a valid revoked certificate and the authority that revoked ψ_r must be an honest authority (lines 5.1-5.2). The second requirement is that the authority that is claimed to revoke ψ_r did not revoke it (line 5.3).

Requirement 7. *Δ-Transparency (ΔTRA). Adversary $\mathcal{A}$ wins in the Δ-transparency experiment* $\mathbf{SecExp}^{\Delta\mathsf{TRA},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$*, if it produces a valid certificate ψ, which is transparent on time $\psi.\rho[\Delta\mathsf{TRA}].clk$, yet there is an honest authority ι that is not aware of ψ after time $\psi.\rho[\Delta\mathsf{TRA}].clk+\Delta$. See Algorithm 4.*

The $\mathbf{SecExp}^{\Delta\mathsf{TRA},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa,\mathsf{N})$ game executes adversary $\mathcal{A}$ using **Exec**, verifies that $\mathcal{A}$ followed the model $\mathcal{M}$, and extracts the execution details as described in $\mathbf{SecExp}^{\mathsf{ACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$.

The adversary wins the game if three requirements are met. First, ψ must be a valid transparent certificate (line 5.1). Second, authority ι and the transparency issuer of ψ are honest authorities (line 5.2). Notice that the game checks that the transparency issuer is honest, but it does not require the transparency issuer to be identical to the issuer of the certificate itself (i.e., accountability issuer). This is since even if the issuer of the certificate is corrupt, this does not prevent the transparency issuer from broadcasting the certificate to the rest of the authorities. For example, in CT, the transparency issuers (loggers) are distinct from the certificate issuers. The third requirement is that there is an honest authority ι that is not aware of the transparency of ψ on time t (lines

5.3-5.5). This requirement is validated by verifying that on time t authority ι was instructed to perform $\mathcal{P}.\mathsf{Query}$ with $\psi.id$, but the output of this invocation did not contain a transparent certificate based on ψ, and was sometime after the time that ψ was supposed to be transparent. Since the $\mathcal{P}.\mathsf{Query}$ algorithm outputs all certificates associated with $\psi.id$, this is a proof that an honest authority is not aware of ψ.

Requirement 8. *Revocation transparency (ΔReTRA). Adversary $\mathcal{A}$ wins in the Δ-revocation transparency experiment* $\mathbf{SecExp}^{\Delta\mathsf{ReTRA},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$ *if it produces a valid certificate ψ_r which is revoked on time $\psi_r.\rho[\Delta\mathsf{ReTRA}].clk$, yet there is an honest authority ι that is not aware of ψ_r after time $\psi_r.\rho[\Delta\mathsf{ReTRA}].clk + \Delta$. See Algorithm 5.*

The $\mathbf{SecExp}^{\Delta\mathsf{ReTRA},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$ game executes adversary $\mathcal{A}$ using **Exec**, verifies that $\mathcal{A}$ followed the model $\mathcal{M}$, and extracts the execution details as described in $\mathbf{SecExp}^{\mathsf{ACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$.

The adversary wins the game if four requirements are met. First, ψ_r must be a valid revoked certificate with the $\Delta\mathsf{ReTRA}$ attribute attribute (line 5.1). Second, authority ι and the revocation transparency issuer of ψ_r must be honest authorities (line 5.2). The third requirement is that the honest authority ι is not aware of the revocation transparency of ψ_r on time t (lines 5.3-5.4). This requirement is validated by verifying that on time t authority ι was instructed to perform $\mathcal{P}.\mathsf{Query}$ with $\psi_r.id$, but the output of this invocation did not contain a revoked transparent certificate based on ψ_r. Since the $\mathcal{P}.\mathsf{Query}$ algorithm outputs all the certificate associated with $\psi_r.id$, this is a proof that an honest authority is not aware of ψ_r. The fourth and final requirement ensures that time t was sometime after the time that ψ_r was supposed to be transparently revoked $\psi_r.\rho[\Delta\mathsf{ReTRA}].clk + \Delta$ (line 5.5). This requirement ensures that the evidence the adversary obtained of winning the game is indeed relevant, since until time $\psi_r.\rho[\Delta\mathsf{ReTRA}].clk + \Delta$ not all honest authorities are required to know ψ_r.

Requirement 9. *Equivocation detection (ΔEQ-D). Adversary $\mathcal{A}$ wins in the Δ-equivocation detection experiment* $\mathbf{SecExp}^{\Delta\mathsf{EQ\text{-}D},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$ *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 ΔEQ-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 ΔEQ-D property has passed. See Algorithm 6.*

The $\mathbf{SecExp}^{\Delta\mathsf{EQ\text{-}D},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$ game executes adversary $\mathcal{A}$ using **Exec**, verifies that $\mathcal{A}$ followed the model $\mathcal{M}$, and extracts the execution details as described in $\mathbf{SecExp}^{\mathsf{ACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$.

The adversary wins the game if several requirements are met. First, ψ and ψ' must be valid certificates for the same identifier (ψ provided by the adversary and ψ' by some ι, line 4) with the equivocation detection attribute (lines 5.1-5.3). Second, both certificates have overlapping validity periods (line 5.4). Third, the run did not include revocation (line 5.5). Fourth, time t must be after the Δ

Algorithm 6 $\mathbf{SecExp}^{\Delta\mathsf{EQ\text{-}D},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$

1: $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$
2: **if** $\mathcal{M}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \perp$ **then** Return $\perp$
3: $\left\{Out_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
4: $(\psi, \iota) \leftarrow Out_{\mathcal{A}}, \psi' \leftarrow Out_\iota^t$
5: Return:
// ψ and ψ' are non-equivocal
5.1: $\mathcal{P}.\mathsf{WasValid}(\psi, \Delta\mathsf{EQ\text{-}D}) \wedge$
5.2: $\mathcal{P}.\mathsf{WasValid}(\psi', \Delta\mathsf{EQ\text{-}D}) \wedge$
// ψ and ψ' have the same identifier
5.3: $\psi.id = \psi'.id \wedge$
// ψ and ψ' have overlapping validity
// periods, yet different *PubInfo*
5.4: $\psi.sd < \psi'.sd < \psi.ed \wedge \psi.pub \neq \psi'.pub \wedge$
// The run did not include ψ's revocation
5.5: $(\nexists t' \leq t, j)\Big(\psi_r \leftarrow Out_j^{t'}; \mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \wedge Core(\psi_r) = Core(\psi)\Big)$
// The Δ time for detection has passed
5.6: $t > \Delta + max(\psi.\rho[\Delta\mathsf{EQ\text{-}D}].clk, \psi'.\rho[\Delta\mathsf{EQ\text{-}D}].clk)$
// No one detected the equivocation
5.7: $(\nexists t' \leq t, j)\left((\psi, \psi', \text{'Equivocation'}) \leftarrow Out_j^{t'}\right)$

Algorithm 7 $\mathbf{SecExp}^{\mathsf{EQ\text{-}P},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$

1: $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$
2: **if** $\mathcal{M}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \perp$ **then** Return $\perp$
3: $\left\{Out_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
4: $(\psi, \iota) \leftarrow Out_{\mathcal{A}}, \psi' \leftarrow Out_\iota^t$
5: Return:
// ψ and ψ' are non-equivocal
5.1: $\mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{EQ\text{-}P}) \wedge$
5.2: $\mathcal{P}.\mathsf{WasValid}(\psi', \mathsf{EQ\text{-}P}) \wedge$
// ψ and ψ' have the same identifier
5.3: $\psi.id = \psi'.id \wedge$
// ψ and ψ' have overlapping validity
// periods, yet different *PubInfo*
5.4: $\psi.sd < \psi'.sd < \psi.ed \wedge \psi.pub \neq \psi'.pub \wedge$
// The run did not include ψ's revocation
5.5: $(\nexists t' \leq t, j)\Big(\psi_r \leftarrow Out_j^{t'}; \mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \wedge Core(\psi_r) = Core(\psi)\Big)$

time period has passed, relative to the time that the certificates were upgraded with the ΔEQ-D property, the latest of the two certificates (line 5.6). Finally, none of the entities has detected the equivocation (line 5.7).

Requirement 10. *Equivocation prevention (*EQ-P*). Adversary $\mathcal{A}$ wins in the equivocation prevention experiment* $\mathbf{SecExp}^{\mathsf{EQ\text{-}P},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$ *if it produces two valid, non-revoked certificates ψ, ψ' for the same identifier ($\psi.id = \psi'.id$) and for overlapping validity periods, where each certificate has different public information ($\psi.pub \neq \psi'.pub$). See Algorithm 7.*

The $\mathbf{SecExp}^{\mathsf{EQ\text{-}P},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N})$ game executes adversary $\mathcal{A}$ using **Exec**, verifies that $\mathcal{A}$ followed the model $\mathcal{M}$, and extracts the execution details as described in $\mathbf{SecExp}^{\mathsf{ACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$.

The adversary wins the game if three requirements are met. First, ψ and ψ' must be valid certificates for the same identifier (ψ provided by the adversary and ψ' by some ι, line 4) with the equivocation prevention attribute (lines 5.1-5.3). Second, both certificates have overlapping validity periods (line 5.4). Third, the run did not include revocation (line 5.5).

4.5 Liveness Requirement

Liveness requires that operations complete, eventually or in bounded time, and with appropriate outputs. Since we defined the scheme as a set of algorithms, they all immediately return with some value; however, we allowed *pending* responses, which require later an *upgrade* to the final outcome. Pending responses are required to handle cases where some cooperation and communication between authorities is required to complete the operation successfully, e.g., to ensure transparency or non-equivocation. Therefore, for a PKI scheme to ensure liveness, it should ensure that such *pending* responses are, in due time, resolved. However, clearly, this can only be guaranteed in runs which satisfy some *liveness conditions*, such as, a sufficient number of benign authorities and reliable, bounded-delay communication. We denote the liveness condition LiveC, namely, LiveC is the *liveness-conditions predicate* over runs, s.t. if $\mathsf{LiveC}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \top$, then operations should complete successfully.

Requirement 11. *Liveness.* Adversary $\mathcal{A}$ wins in the $\Delta LiveC - Liveness$ experiment $\mathbf{SecExp}^{\Delta\mathsf{LiveC}-\mathsf{Liveness},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$, if the execution fails to upgrade a certificate (with some attribute *attr*), although a pending certificate existed whose pending period has completed, and although the liveness criteria was met. See Algorithm 8.

Algorithm 8 $\mathbf{SecExp}^{\Delta\mathsf{LiveC}-\mathsf{Liveness},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^{\kappa}, \mathsf{N})$

1: $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^{\kappa}, \mathsf{N})$
2: **if** $\mathcal{M}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \bot$ **then** Return $\bot$
3: $\left\{Alg_i^{\hat{t}}, Inp_i^{\hat{t}}, Out_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
4: $\iota \leftarrow Out_{\mathcal{A}}, [\psi, attr, \alpha] \leftarrow Inp_{\iota}^{t}$
5: Return:
// ι is an honest authority and $\mathcal{P}$.Upgrade was invoked on ι on round t
5.1: $\iota \in \mathsf{N} - \mathsf{N_F} \ \wedge \ Alg_{\iota}^{t} = \mathcal{P}.\mathsf{Upgrade} \ \wedge$
// ψ is a valid pending certificate that its pending period is over
5.2: $\mathcal{P}.\mathsf{WasValid}(\psi, attr) = Pending \ \wedge \ Clk_{\iota}^{t} \geq \psi.\rho[attr].clk + \Delta$
// The liveness criteria was met, however, no progress was made, i.e., ι did not upgrade ψ
5.3: $\mathsf{LiveC}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) \ \wedge \ \mathcal{P}.\mathsf{WasValid}(Out_{\iota}^{t}, attr) \neq \top$

5 Applying the PKI Framework

The goals of the PKI framework is to facilitate analysis of PKI schemes, including proofs of security by reductions, allowing comparison and evaluation of different PKI schemes. The challenge is that there are very different PKI schemes, with different approaches, goals and models; our goal is that the framework will be applicable to most or all of them, yet precise and usable.

To validate that our framework meets these objectives and fine-tune it, we applied it to two very different PKI schemes; this section presents the main results. The first scheme is *Certificate Transparency (CT)*, the only widely-deployed 'post-X.509' PKI scheme; the second scheme is a *proof-of-concept* PKI scheme called PoC-PKI.

5.1 Certificate Transparency

CT is widely known and has multiple extensions and influence, and is being standardized by the IETF [36]. That said, CT is quite complex, not always well and completely defined, and subject of significant criticism; all this makes precise analysis challenging and important. We applied the PKI framework to CT according to RFC6962 [36] and previous publications; unfortunately, we found several crucial under-specified aspects in the RFC. For example, a critical aspect of CT is the gossip protocol[4], but RFC6962 does not define this protocol. We report in this section on the analysis of 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. Obviously, the analysis can be easily adapted to support more improved, efficient and/or complex variants of CT. See [1] for the complete design of CT_{comp} as well as its analysis.

Authorities. CT considers, in addition to Certificate Authorities, also several other authorities: *loggers*, who keep public logs of certificates issued by different CAs; *monitors*, who validate that logs are published correctly; and *auditors*, who validate that certificates appear in the appropriate log.

Each monitor in N_M monitors at least one logger, but can monitor as many loggers as it chooses; we describe $Mon(\ell) \subseteq \mathsf{N_M}$ as the subset of the monitors, that monitors a specific logger $\ell \in \mathsf{N_L}$. Formally, we assume that the 'roles' of each authority are specified as the corresponding string, in the first input from the adversary, e.g., authority i is a monitor if 'Monitor' $\in Inp_i^1$.

Model Function $\mathcal{M}$ for CT_{comp}. CT_{comp} assumes bounded delay communication and bounded drift clocks[5], with the corresponding model functions $\mathcal{M}^{BD-COM}_{\Delta_{com}}$, $\mathcal{M}^{BD-SYNC}_{\Delta_{clk}}$ defined in Section 3.3; in addition, it uses a unique *faults model* $\mathcal{M}^{CT-FAULT}$, defined below. Together, we have:

$$\mathcal{M}^{CT}_{\Delta_{com},\Delta_{clk}}(\xi) = \{\mathcal{M}^{CT-FAULT}(\xi) \wedge \mathcal{M}^{BD-COM}_{\Delta_{com}}(\xi) \wedge \mathcal{M}^{BD-SYNC}_{\Delta_{clk}}(\xi)\}$$

[4] "All clients should gossip with each other, exchanging STHs at least"

[5] 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 clock-synchronization is required to allow some of CT's operations, although it is not explicitly stated.

The fault model function is unique to CT; the adversary can control any entity in N *as long as* every logger is monitored by *at least* one honest monitor. Namely:

$$\mathcal{M}^{CT-FAULT}(\xi) = \{(\forall i \in \xi.\mathsf{N} - \xi.\mathsf{N_F})i \in \xi.R.\mathsf{N_M} \Leftrightarrow \text{`Monitor'} \in \xi.Inp_i^1\}$$

Analysis. Our analysis proves that CT_{comp} achieves most of the properties of the PKI framework. The theorems refers to $CT_{comp}^{\mathcal{S},\mathcal{H}}$, which specifies the signature scheme $\mathcal{S}$ and CRHF $\mathcal{H}$ used by CT_{comp}; assume both are secure (using standard definitions).

Theorem 1. *$CT_{comp}^{\mathcal{S},\mathcal{H}}$ satisfies security requirements* accountability *and* revocation accountability*, under model $\mathcal{M}^{CT}_{\Delta_{com},\Delta_{clk}}$.*

Proof. See [1]. □

We next prove that CT_{comp} ensures Δ_{CT}-transparency, with $\Delta_{CT} = 4\Delta_{com} + 2\Delta_{clk}$. Δ_{CT} 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.

Theorem 2. *$CT_{comp}^{\mathcal{S},\mathcal{H}}$ satisfies security requirement Δ_{CT}-transparency under model $\mathcal{M}^{CT}_{\Delta_{com},\Delta_{clk}}$.*

Proof. See [1]. □

Notice that *CT does not prevent equivocation*: it is possible for a CA to issue a (potentially fake) certificate for an identifier (domain) which already has an existing (legitimate) certificate. This can be detected - if both certificates are 'transparent'. Equivocation does not require corrupt authorities, since CT does not require CAs or loggers to refuse issuing equivocal certificates.

Liveness. The only scenario in which pending certificates are used in CT_{comp} is for transparency. The liveness conditions for such scenario checks whether there was a legitimate reason for the logger to decline the request[6] to add some certificate to the log. We define the function $publish(\psi) \in \{\top, \bot\}$ that represent these requirements in CT_{comp} as they involve 'outside' consideration, e.g., some loggers might not trust specific root CAs. Hence, the appropriate liveness condition in CT_{comp} is:

$$\mathsf{LiveC}^{CT_{comp}}(\xi) = \{(\exists \ell \in \mathsf{N} - \mathsf{N_F}) \ \wedge \ publish(\xi.R.Inp_\iota^t)\}$$

[6] From RFC6962:"logs MUST refuse to publish certificates without a valid chain to a known root CA"

Theorem 3. *$CT^{\mathcal{S},\mathcal{H}}_{comp}$ satisfies security requirement* $\mathsf{Live}\mathsf{C}^{CT_{comp}}$*-liveness under model $\mathcal{M}^{CT}_{\Delta_{com},\Delta_{clk}}$.*

Proof. See [1]. □

5.2 PoC-PKI

We now briefly describe the design of our provably secure proof-of-concept PKI, PoC-PKI; see [4] for the full description and proofs. PoC-PKI takes a different approach than public log based PKI schemes. Instead of allowing a number of (potentially) untrusted logging entities to maintain the certificates and having monitoring entities enforce proper behavior of loggers, PoC-PKI uses threshold signature scheme to simulate a *collectively* trusted entity that manages the certificates. Similar ideas were proposed in CoSi [51] and ARPKI [8].

In PoC-PKI, the most basic form of a certificate is an *accountable* certificate, i.e., a certificate *issued* by an authorized authority. An accountable certificate can be *upgraded* with various properties. For example, an accountable certificate ψ can be upgraded to be Δ-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 Δ time. For non-equivocation, PoC-PKI use *pending* certificates. The request to upgrade ψ into a non-equivocal certificate is propagated 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 several scheme: 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$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$; for brevity, we use the shorter notation PoC-PKI instead of the fully-qualified notation where the identification of $\mathcal{S}$, $\mathcal{E}$ and $\mathcal{TS}$ is irrelevant and confusion seems unlikely.

Model Function $\mathcal{M}$ for PoC-PKI. Following the common models described in Section 3.3, the model function is defined as:

$$\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}(\xi) = \{\mathcal{M}^{NumF}_{\lfloor(n/3)\rfloor}(\xi) \wedge \mathcal{M}^{BD-COM}_{\Delta_{com}}(\xi) \wedge \mathcal{M}^{BD-SYNC}_{\Delta_{clk}}(\xi)\} \tag{2}$$

where the fault model $\mathcal{M}^{NumF}_{\lfloor(n/3)\rfloor}$ sets the number of entities that can arbitrarily misbehave to less than a third of all authorities, the communication model $\mathcal{M}^{BD-COM}_{\Delta_{com}}$ is a bounded communication delay (bounded by Δ_{com}), and the synchronization model $\mathcal{M}^{BD-SYNC}_{\Delta_{clk}}$ is a bounded-drift clock synchronization model (bounded by Δ_{clk}).

Analysis. We use the notation PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ for implementation of PoC-PKI that uses signature scheme $\mathcal{S}$, encryption scheme $\mathcal{E}$ and threshold signature scheme $\mathcal{TS}$.

Let $\Delta_{\text{PoC-PKI}} = \Delta_{com} + \Delta_{clk}$ denote the delay ensured by PoC-PKI for transparency and revocation transparency. 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, using the output messages $\{m^1_{i,j}\}_{j\in\mathsf{N}}$ (see line 8.2.1 in the **Exec** algorithm). Hence, it takes at most Δ_{com} time for the messages to arrive to other entities. Following line 8.2.2 in the **Exec** algorithm, entities process this message in the following round. However, their clocks might not be completely synchronized, which can add up to Δ_{clk} additional delay.

Theorem 4. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves accountability, revocation accountability,* $\Delta_{\text{PoC-PKI}}$*-transparency,* $\Delta_{\text{PoC-PKI}}$*-revocation transparency and equivocation prevention under model* $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$.

Proof. See [4]. □

Liveness. The only scenario in which pending certificates are used in PoC-PKI is for equivocation prevention. The liveness conditions for such scenario checks whether there was no legitimate reason to decline the upgrade of the certificate. Since the honest authority that refuses to complete the upgrade operation outputs an 'explanation', i.e., an already *existing* equivocating certificate, the liveness condition can verify if the explanation is legitimate. Hence, the appropriate liveness condition in PoC-PKI is:

$$\mathsf{LiveC}^{\text{PoC-PKI}}(\xi) = \left\{ \begin{array}{l} \left(\forall(\psi, \mathsf{EQ\text{-}P}) = \xi.R.Inp^{\xi.t}_{\xi.Out_{\mathcal{A}.\iota}}, \psi\text{'} = \xi.R.Out^{\xi.t}_{\xi.Out_{\mathcal{A}.\iota}}\right) \\ \mathcal{P}.\mathsf{WasValid}(\psi\text{'}, \mathsf{EQ\text{-}P}) = \bot \ \vee \ \psi.id \neq \psi\text{'}.id \ \vee \\ \psi.sd > \psi\text{'}.ed \end{array} \right\} \tag{3}$$

Theorem 5. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *achieves* $\mathsf{LiveC}^{\text{PoC-PKI}}$-liveness *under model* $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$.

Proof. See [4]. □

6 Conclusions and Future Work

We presented a *PKI framework*, which defines model and security requirements for PKI schemes. We showed the flexibility and applicability of our framework by applying it to two very different PKI schemes: PoC-PKI, a 'proof-of-concept' PKI that meets all requirememnts, and CT_{comp}, a completion of the Certificate Transparency (CT) specifications.

We hope that this work can provide a good foundation and starting point for collaborative effort by the cryptographic and security communities to 'debug',

improve and extend these definitions, as well as explore variants - as was the case for other basic cryptographic schemes, e.g., encryption. Some possible extensions were already discussed, informally, in previous works (and this work), most notably, several different notions of privacy. Other directions for further research include (1) analysis of existing (and new) PKI schemes with respect to our framework, (2) designing a practical PKI that will ensure all of our security requirements, including prevention of non-equivocation, possibly as an optimized variant of PoC-PKI, which we presented in a simplified, sub-optimal design, (3) adopting the framework and/or its execution model, to define and study security properties of other complex cryptographic systems, and (4) extending the framework to support secure compositions, e.g. following UC [13], and specifically [14] (which present a basic UC model for certification).

References

1. Analysis and specifications of the certificate transparency pki scheme, available from authors
2. Gossiping in CT draft-ietf-trans-gossip-04. https://tools.ietf.org/html/draft-ietf-trans-gossip-04
3. Namecoin, https://www.namecoin.org/
4. Provable security for pki schemes (full version), available from authors
5. 0002, J.L., 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)
6. 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)
7. Axon, L., Goldsmith, M.: Pb-pki: A privacy-aware blockchain-based pki. In: SECRYPT (2017)
8. 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)
9. 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)
10. Braun, J.: Maintaining security and trust in large scale public key infrastructures. Ph.D. thesis, Technische Universität (2015)
11. Braun, J., Kiefer, F., Hülsing, A.: Revocation and non-repudiation: when the first destroys the latter. In: European Public Key Infrastructure Workshop. pp. 31–46. Springer (2013)
12. Callegati, F., Cerroni, W., Ramilli, M.: Man-in-the-middle attack to the https protocol. IEEE Security & Privacy **7**(1), 78–81 (2009)
13. 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.

14. 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
15. 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)
16. CCITT, B.B.: Recommendations x. 509 and iso 9594-8. Information Processing Systems-OSI-The Directory Authentication Framework (Geneva: CCITT) (1988)
17. 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)
18. Comodo™: Incident Report. Published online, http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html (March 2011)
19. 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)
20. 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)
21. 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)
22. Dyer, J.: China accused of doling out counterfeit digital certificates in 'serious' web security breach. VICE News (April 2015)
23. 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)
24. (EFF), E.F.F.: The eff ssl observatory, https://www.eff.org/observatory
25. Fouque, P.A., Stern, J.: Fully distributed threshold rsa under standard assumptions. Advances in CryptologyASIACRYPT 2001 pp. 310–330 (2001)
26. Fromknecht, C., Velicanu, D., Yakoubov, S.: A decentralized public key infrastructure with identity retention. IACR Cryptology ePrint Archive **2014**, 803 (2014)
27. 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)
28. Goldreich, O.: Foundations of cryptography. Cambridge University Press (2009)
29. 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)
30. Hruska, J.: Apple, microsoft buck trend, refuse to block unauthorized chinese root certificates. ExtremeTech (April 2015)
31. Huang, J., Nicol, D.M.: An anatomy of trust in public key infrastructure. International Journal of Critical Infrastructures **13**(2-3), 238–258 (2017)
32. 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)
33. Kotcher, R., Pei, Y., Jumde, P., Jackson, C.: Cross-origin pixel stealing: timing attacks using css filters. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 1055–1062. ACM (2013)
34. 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)

35. Laurie, B., Kasper, E.: Revocation transparency. Google Research, September (2012)
36. 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
37. Lekkas, D.: Establishing and managing trust within the public key infrastructure. Computer Communications **26**(16), 1815–1825 (2003)
38. Marchesini, J., Smith, S.: Modeling public key infrastructures in the real world. In: European Public Key Infrastructure Workshop. pp. 118–134. Springer (2005)
39. 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)
40. Maurer, U.: Modelling a public-key infrastructure. In: European Symposium on Research in Computer Security. pp. 325–350. Springer (1996)
41. 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)
42. 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)
43. Prins, J.: Diginotar certificate authority breach 'operation black tulip (2011)
44. 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
45. 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)
46. Ryan, M.D.: Enhanced certificate transparency and end-to-end encrypted mail. In: NDSS (2014)
47. 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)
48. 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
49. Shoup, V.: Practical threshold signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 207–220. Springer (2000)
50. 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)
51. 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)
52. 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)
53. 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)

54. 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)
55. Yu, J., Cheval, V., Ryan, M.: DTKI: A new formalized PKI with verifiable trusted parties. The Computer Journal **59**(11), 1695–1713 (2016)
56. 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 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 B for rigorous proofs and analysis).

A.1 High-Level Overview

In PoC-PKI, besides clients, there is only one type of entities, which are called *authorities*. Authorities issue certificates to clients using the PoC-PKI.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.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 ψ and $attr$ signed by $\psi.\rho[attr].\iota \in \mathsf{N}$.

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 Δ-transparent certificate, the authority uses the PoC-PKI.$\mathsf{Upgrade}$ 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 Δ-revocation transparency upgrade; the authority uses the PoC-PKI.$\mathsf{Upgrade}$ 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 Δ time period passes.

When a client requests to upgrade a certificate into an unequivocal certificate, the authority also use the PoC-PKI.$\mathsf{Upgrade}$ 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, i.e., containing at least $|\mathsf{N}| - f$ properly-signed 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.

A.2 Preliminaries

PoC-PKI uses three underlying cryptographic schemes: public-key encryption, signatures and threshold-signatures. These are all standard cryptographic schemes with multiple known implementations, including provably-secure constructions from basic primitives. We briefly recall the definitions of these schemes along with their security games.

Definition 2. *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 3. *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*.*

Definition 4. *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 5. *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*.*

While definitions 2,3,4,5 are standard definitions of encryption and signature schemes, choosing a definition of a threshold signature scheme is slightly more complex. 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. For example, in [27], the threshold scheme is presented similarly to a classical signature scheme, with the necessary adjustments. Namely, the key generation algorithm is modified to output shares of the signature key and the signing algorithm is a distributed algorithm that outputs the *final* group signature. Another example is [49], 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 [5, 25], present and discuss more possible designs, definitions and implementations.

We chose to define the threshold scheme using the four algorithm notation $\mathcal{TS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Ver})$, where: $\mathcal{TS}.\mathsf{Gen}$ is the key generation algorithm, $\mathcal{TS}.\mathsf{Sign}$ is the *individual* singing 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 are a simplified generalization of the existing designs and definitions, and allow a clear presentation of PoC-PKI; however, further work (which is beyond the scope of this paper) is required to explore the relationships between these definitions.

Definition 6. *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 7. *A* (t, n)*-threshold-signature scheme* $\mathcal{TS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Ver})$ *is existentially unforgeable, if there is no PPT adversary* $\mathcal{A}$ *that controls up to* t *players and has an oracle access to* $\mathcal{TS}$ *that can produce a cryptographically valid group signature on any previously unsigned 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. $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{TS}.\mathsf{Sign}(tsk_i,\cdot)}_{tvk}$ *(*$\mathcal{A}$ *outputs message* m *and signature* σ*, given* tvk *and oracle access to* $\mathcal{TS}$ *with key* tsk_i*)*
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 *times, otherwise, the experiment outputs* 0*.*

Definition 8. *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:*

$$\begin{aligned}(\forall m \in M, W \subseteq \{\sigma_i = \mathcal{TS}.\mathsf{Sign}(tsk_i, m) \mid i \in \mathsf{N} - \mathsf{N_F}\},\\ W' \leftarrow \mathcal{A})|W| \geq t+1 \Rightarrow \\ \mathcal{TS}.\mathsf{Ver}(tvk, m, \mathcal{TS}.\mathsf{Combine}(W \cup W')) = \top\end{aligned} \tag{4}$$

A.3 System Model

PoC-PKI's system model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ uses common models described in Section 3.3, and is defined as:

$$\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}(\xi) = \{\mathcal{M}^{NumF}_{\lfloor(n/3)\rfloor}(\xi) \wedge \mathcal{M}^{BD-COM}_{\Delta_{com}}(\xi) \wedge \mathcal{M}^{BD-SYNC}_{\Delta_{clk}}(\xi)\}$$

Namely, the fault model $\mathcal{M}^{NumF}_{\lfloor(n/3)\rfloor}$ sets the number of entities that can arbitrarily misbehave to less than a third of all authorities, the communication model $\mathcal{M}^{BD-COM}_{\Delta_{com}}$ is a bounded communication delay (bounded by Δ_{com}), and the synchronization model $\mathcal{M}^{BD-SYNC}_{\Delta_{clk}}$ is a bounded-drift clock synchronization model (bounded by Δ_{clk}).

A.4 PoC-PKI Algorithms

PoC-PKI.$\mathsf{Gen}(1^\kappa)$ (Algorithm 9). This algorithm receives as input security parameter 1^κ, and uses it to generate cryptographic encryption and signing keys.

Algorithm 9 PoC-PKI.$\mathsf{Gen}(1^\kappa)$

// Generate decryption/encryption key pair using the secure encryption scheme $\mathcal{E}$
1: $(dk, ek) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$
// Generate signing/verification key pair using the secure signature scheme $\mathcal{S}$
2: $(sk, vk) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$
// Output key pairs
3: Return $(PrivInfo = (dk, sk), PubInfo = (ek, vk))$

PoC-PKI.$\mathsf{GroupGen}(1^\kappa, \mathsf{N}, \{PubInfo_i\}_{i\in\mathsf{N}})$ (Algorithm 10). This algorithm receives as input security parameter 1^κ, the set N, and public information $PubInfo_i =$

(ek_i, vk_i) for every authority $i \in \mathsf{N}$. It first uses $\mathcal{TS}.\mathsf{Gen}$ to generate group verification key tvk and $|\mathsf{N}|$ partial signature key secret shares tsk_i (line 1). Then, it computes a hint $h_i = \mathcal{E}.\mathsf{Enc}(PubInfo_i.ek, tsk_i)$ which is the encrypted partial secret share tsk_i for every authority $i \in \mathsf{N}$ (line 2). The algorithm concludes by outputting the the group verification key tvk, the public information $\{PubInfo_i\}_{i\in\mathsf{N}}$, and the encrypted set of hints H (lines 3-4).

Algorithm 10 PoC-PKI.$\mathsf{GroupGen}(1^\kappa, \mathsf{N}, \{PubInfo_i\}_{i\in\mathsf{N}})$

// Generate a group verification key tvk and partial signing keys tsk_i
1: $(tvk, \{tsk_i\}_{i=1}^{|\mathsf{N}|}) \leftarrow \mathcal{TS}.\mathsf{Gen}(1^\kappa, |\mathsf{N}|, \lfloor |\mathsf{N}|/3 \rfloor)$
// Encrypt partial signing key tsk_i such that only authority i can decrypt it
2: $H = \{h_i \leftarrow \mathcal{E}.\mathsf{Enc}(PubInfo_i.ek, tsk_i)\}_{i\in\mathsf{N}}$
// Output the public information and individual hints
3: $PubInfo \leftarrow (tvk, \{PubInfo_i\}_{i\in\mathsf{N}}, H)$
4: Return $PubInfo$

PoC-PKI.$\mathsf{Issue}(id, pub, sd, ed)$ (Algorithm 11). Inputs are 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 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 11 PoC-PKI.$\mathsf{Issue}(id, pub, sd, ed)$

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

PoC-PKI.$\mathsf{Query}(id)$ (Algorithm 12). Once a certificate appears in the local state it can be queried using the Query algorithm. Given an identifier id, the algorithm returns all certificates locally associated with id.

Algorithm 12 PoC-PKI.Query(id)

// Check if there are certificates associated with id
1: **if** id **in** $S.certs$ **then** return $S.certs[id]$
// Otherwise
2: Return $\perp$

PoC-PKI.Revoke(ψ) (Algorithm 13). 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).

Algorithm 13 PoC-PKI.Revoke(ψ)

// Verify that ψ was issued by the authority and ψ is a valid, not expired certificate
1: **if** $\psi.\rho[\mathsf{ACC}].\iota \neq S.\iota \ \vee \ \mathcal{P}.\mathsf{WasValid}(\psi) \neq \top \ \vee \ \psi.ed < clk$ **then** return $\perp$
// If ψ was already revoked, return it
2: **if** $\exists \psi_r \in S.certs[\psi.id] \ s.t. \ \mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \ \wedge \ Core(\psi) = Core(\psi_r)$ **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

PoC-PKI.IsRevoked(ψ) (Algorithm 14). 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 a signed proof that the certificate was not revoked until the current local time under the NR attribute and outputs the certificate.

Algorithm 14 PoC-PKI.IsRevoked(ψ)

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

Note: sending/broadcasting messages. Recall that our execution model (Algorithm 1) allows an algorithm running in one authority i, to specify messages $m_{i,j}$ to be sent to authority j. We next describe $\mathsf{Upgrade}(\psi, attr, [\alpha])$, which is the first - and one of few - algorithms in PoC-PKI that send messages. For simplicity, PoC-PKI sends most messages m to all entities, which we write as $Broadcast(m)$.

PoC-PKI.$\mathsf{Upgrade}(\psi, attr, [\alpha])$ (Algorithm 15). 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.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 ($\psi.\rho[\mathsf{EQ\text{-}P}].clk + \Delta$) 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.Incoming algorithm). Therefore, since line 2 did not found such certificate, this means that the upgrade failed.

Algorithm 15 PoC-PKI.Upgrade$(\psi, attr, [\alpha])$

```
    // Verify that ψ is a valid, not expired certificate
1:  if P.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') ∧ P.WasValid(ψ', attr) then return ψ'
3:  ψ' ← ψ
4:  switch attr
        // Transparency upgrade
4.1:    case ΔTRA ∨ ΔReTRA
            // Add the transparency proof to ψ'
4.1.1:      data ← (Core(ψ), attr, clk)
4.1.2:      σ = S.Sign(S.PrivInfo.sk, data)
4.1.3:      ψ'.ρ[attr] ← (σ, S.ι, clk)
4.1.4:      Broadcast(ψ')
        // Equivocation prevention upgrade for a non-pending certificate
4.2:    case (EQ-P ∧ P.WasValid(ψ, EQ-P) = ⊥)
            // Add the PendEQ-P attr to ψ'
4.2.1:      data ← (Core(ψ), PendEQ-P, clk)
4.2.2:      σ = S.Sign(S.PrivInfo.sk, data)
4.2.3:      ψ'.ρ[EQ-P] ← (σ, S.ι, clk)
4.2.4:      Broadcast(ψ')
        // Non-equivocation upgrade for pending certificate
4.3:    case EQ-P ∧ P.WasValid(ψ, EQ-P) = Pending
            // If upgrade time (Δ = 2) did not pass, return the pending certificate
4.3.1:      if clk < ψ.ρ[EQ-P].clk + Δ then return ψ
            // Upgrade time passed already yet still 'pending': failure
4.3.2:      Return (⊥, S.certs[ψ.id].ρ[attr].failure)
    // Add the certificates to the local set of certificates
5:  S.certs[ψ.id] += {ψ, ψ'}
6:  Return ψ'
```

PoC-PKI.WasValid$(\psi, [attr, tms])$ (see Algorithm 16). 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 (NR), 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).

Algorithm 16 PoC-PKI.WasValid($\psi, [attr, tms]$)

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

// 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

2: **if** $\mathsf{Authorized}(\chi.\iota, \psi.id) \neq \top \ \vee \ tms < \psi.sd \ \vee \ tms > \psi.ed \ \vee$
$\mathcal{S}.\mathsf{Ver}(S.PubInfo_{\chi.\iota}.vk, (Core(\psi), \mathsf{ACC}, \chi.clk), \chi.\sigma) \neq \top$

2.1: Return $\bot$

// If no attribute was supplied or the attribute is accountability, return true

3: **if** $attr = \bot \ \vee \ attr = \mathsf{ACC}$ **then** return $\top$

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

// For the attributes that are implemented solely using proofs

5: **if** $attr \in \{\Delta\mathsf{TRA}, \Delta\mathsf{ReTRA}, \mathsf{REV}\}$

// Check that the proof is cryptographically valid

5.1: Return $\mathcal{S}.\mathsf{Ver}(S.PubInfo_{\eta.\iota}.vk, (Core(\psi), attr, \eta.clk), \eta.\sigma)$

// For the NR attribute

6: **if** $attr = \mathsf{NR}$

// Check that the proof is cryptographically valid *and* that it is relevant to tms

6.1: Return $\mathcal{S}.\mathsf{Ver}(S.PubInfo_{\eta.\iota}.vk, (Core(\psi), attr, \eta.clk), \eta.\sigma) \ \wedge \ \psi.\rho[\mathsf{NR}].clk \geq tms$

// For the equivocation prevention attribute

7: **if** $attr = \mathsf{EQ\text{-}P}$

// Check if ψ is pending

7.1: **if** $\mathcal{S}.\mathsf{Ver}(S.PubInfo_{\eta.\iota}.vk, (Core(\psi), Pend\mathsf{EQ\text{-}P}, \eta.clk), \eta.\sigma)$ **then** return $Pending$

// Certificate is not pending, check if the group proof is cryptographically valid

7.2: Return $\mathcal{TS}.\mathsf{Ver}(S.PubInfo.tvk, (Core(\psi), \mathsf{EQ\text{-}P}), \eta.\sigma)$

8: Return $\bot$

PoC-PKI.Incoming (see Algorithm 17). 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 17 PoC-PKI.Incoming(M)

```
    // Process each of the messages
1:  for each m' ∈ M
        // If m' is a broadcasted certificate
1.1:    if ψ ← m' s.t. P.WasValid(ψ)
            // Add the certificate to the local state
1.1.1:      S.certs[ψ.id] += ψ
            // If ψ is pending equivocation prevention
1.1.2:      if P.WasValid(ψ, EQ-P) = Pending
                // If a conflicting certificate is known, return it to abort
1.1.3:          if ∃ψ' ∈ S.certs s.t. ψ.id = ψ'.id ∧
                   Core(ψ) ≠ Core(ψ') ∧ P.WasValid(ψ', EQ-P) ≠ ⊥
                    // Prepare a rejection response with the conflicting certificates
1.1.4:              res ← (ψ, ψ')
                // No conflicting certificate - approve the request for non-equivocation
1.1.5:          else
                    // Prepare a partial signature approving the upgrade
1.1.6:              res ← (ψ, σ = TS.Sign(S.PrivInfo.tsk, (Core(ψ), EQ-P)))
                    // Send back the relevant response
1.1.7:          Send res to authority ψ.ρ[EQ-P].ι
        // If m' is an upgrade rejection
1.2:    else if (ψ, ψ') ← m' s.t.P.WasValid(ψ, EQ-P) = Pending ∧ P.WasValid(ψ', EQ-P) ≠ ⊥
            // Store the conflicting certificate
1.2.1:      S.certs[ψ.id].ρ[EQ-P].failure = ψ'
        // If m' is a partial approval of a pending certificate which has not been rejected yet
1.3:    else if (ψ, σ) ← m' s.t. P.WasValid(ψ, EQ-P) = Pending ∧
                                    S.certs[ψ.id].ρ[EQ-P].failure = ⊥
            // Add new partial signature to the local state
1.3.1:      S.toUpgrade[ψ.id] += σ
            // Check to see if enough semi-signatures arrived
1.3.2:      if |S.toUpgrade[ψ.id]| < |N| − ⌊|N|/3⌋ then continue
            // Enough semi-proofs have arrived - try to combine them
1.3.3:      ψ' ← ψ
1.3.4:      ψ'.ρ[NEQ].σ ← TS.Combine(S.toUpgrade[ψ.id])
            // Check if the upgrade was successful
1.3.5:      if P.WasValid(ψ', EQ-P) then S.certs[ψ'.id] += ψ'
```

The PoC-PKI.Time algorithm is not required in PoC-PKI, since in PoC-PKI there are no time-based events.

B Analysis of PoC-PKI

In this section, we provide reduction-based proofs showing that PoC-PKI achieves its safety and liveness properties. We first show that PoC-PKI achieves accountability, Δ-transparency, revocation accountability and Δ-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 existential unforgeability of a secure threshold signature scheme. We conclude by showing that PoC-PKI achieves the liveness properties of the PKI framework.

B.1 Proofs of Accountability, Revocation Accountability, Δ-Transparency and Δ-Revocation Transparency

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

1. Given a security property $\xi \in \{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{ReACC}, \Delta\mathsf{ReTRA}\}$, we assume to the contrary that PoC-PKI *does not* achieves ξ.
2. Hence, there must exist a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ that wins, with a non-negligible probability, the security experiment defined for ξ.
3. We show how to use this PPT adversary $\mathcal{A}$, to build another PPT adversary, AdvEU, that breaks the existential unforgeability of the secure signing scheme $\mathcal{S}$, thus contradicting that such $\mathcal{A}$ exists. Specifically:
   (a) We first define a variation of PoC-PKI called $\overline{\text{PoC-PKI}}^{OSign(\cdot)}_{\iota,vk}$ (see Def. 9).
   (b) Then, we construct the AdvEU adversary (see Alg. 18) that executes the given adversary $\mathcal{A}$ in $\overline{\text{PoC-PKI}}^{OSign(\cdot)}_{\iota,vk}$ under the **Exec** execution model, and outputs a message m and signature σ over m.
   (c) Finally, we argue that if adversary $\mathcal{A}$ prevents PoC-PKI from achieving property ξ, then adversary AdvEU breaks the existential unforgeability of $\mathcal{S}$ (see Claim 1).

The $\overline{\text{PoC-PKI}}^{OSign(\cdot)}_{\iota,vk}$ scheme. We start by defining the $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(\cdot)}_{\iota,vk,dk,ek}$ scheme; for brevity, where the identities of $\mathcal{S}$, $\mathcal{E}$ and $\mathcal{TS}$ are clear or irrelevant, we may use the shorthand $\overline{\text{PoC-PKI}}^{OSign(\cdot)}_{\iota,vk}$.

Definition 9. *Let $\mathcal{S}$, $\mathcal{E}$ and $\mathcal{TS}$ be a signature, encryption and threshold-signature schemes, respectively, and let $(sk, vk) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and $(dk, ek) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$, for a given 1^κ. Given a PPT oracle OSign, let $\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(\cdot)}_{\iota,vk,dk,ek}$ (abbreviated as $\overline{\text{PoC-PKI}}^{OSign(\cdot)}_{\iota,vk}$) be a PKI scheme where one designated authority $\iota \in \mathsf{N}$ executes the* PoC-PKI *with the following changes, and other authorities in N execute* PoC-PKI *without any changes:*

1. $\overline{\text{PoC-PKI}}^{OSign(\cdot)}_{\iota,vk}.\mathsf{Gen}$ *is the same as* PoC-PKI.Gen *(Alg. 9), except for replacing lines* $2-3$ *of* PoC-PKI.Gen *with the following line:*

$$Return\ (PrivInfo = (dk, nil), PubInfo = (ek, vk))$$

2. *In the* Issue, Upgrade, Revoke *and* IsRevoked *algorithms,* replace *the following line of code:*

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

with the following line of code:

$$\sigma = OSign(data)$$

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

The AdvEU **adversary.** We now describe the $\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}$ algorithm (see Alg. 18), referred to as AdvEU for brevity. The algorithm takes as input a security parameter 1^κ, an attribute ξ, a public verification key vk, and a set of authorities N. In addition, the algorithm takes an adversary $\mathcal{A}$, a PoC-PKI implementation, a secure signature scheme $\mathcal{S}$, a secure encryption scheme $\mathcal{E}$, a robust threshold signature scheme $\mathcal{TS}$, an execution model $\mathcal{M}$, and also has an oracle access to the signing function $\mathcal{S}.\mathsf{Sign}$, with the (unknown) secret signing key sk (which is vk's respective secret key), represented by the $OSign(\cdot)$ notation. The algorithm randomly selects an authority $\iota_r \in \mathsf{N}$ (line 1), and then executes $\overline{\text{PoC-PKI}}^{OSign(\cdot)}_{\iota_r,vk}$ with adversary $\mathcal{A}$ using the **Exec** execution model (line 2). In line 3, the algorithm extracts a certificate ψ and identifier $\iota \in \mathsf{N}$ that was chosen by $\mathcal{A}$. A valid execution according to the model $\mathcal{M}$ is verified in line 4, along with the verification that the chosen authority chosen by $\mathcal{A}$ during the execution is identical to the authority chosen by AdvEU, and that this authority is an honest authority. In line 5, the algorithm outputs a message m and a signature σ based on ψ.

Algorithm 18 $\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}(1^\kappa, \xi, vk, \mathsf{N})$

// Randomly choose authority $\iota_r \in \mathsf{N}$
1: $\iota_r \xleftarrow{R} \mathsf{N}$
// Execute $\overline{\text{PoC-PKI}}^{OSign(\cdot)}_{\iota,vk}$
2: $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\overline{\text{PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OSign(\cdot)}_{\iota_r,vk,dk,ek}}(1^\kappa, \mathsf{N})$
// Extract output from the execution
3: $(\psi, \iota) \leftarrow Out_{\mathcal{A}}$
// Ensure the adversary followed the model and chose to forge the honest authority ι_r
4: **if** $\mathcal{M}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \perp \ \vee \ \iota_r \neq \iota \ \vee \ \iota \neq \psi.\rho[\xi].\iota \ \vee \ \iota \in \mathsf{N_F}$ **then** Return $\perp$
// Output message m and forged signature σ over m based on ψ
5: Output (m, σ) *s.t.* $m = (Core(\psi), \xi, \psi.\rho[\xi].clk)$ and $\sigma = \psi.\rho[\xi].\sigma$

We now argue that if there exists an adversary $\mathcal{A}$ that can break the security of PoC-PKI, we can use algorithm AdvEU described in Alg. 18 with adversary $\mathcal{A}$ to break the existential unforgeability of $\mathcal{S}$.

Claim 1. Let $\xi \in \{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{ReACC}, \Delta\mathsf{ReTRA}\}$. If PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *does not* achieves ξ, then the signature scheme $\mathcal{S}$ is not existential unforgeable.

Proof. Assume to the contrary that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ does not achieves the security property $\xi \in \{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{ReACC}, \Delta\mathsf{ReTRA}\}$, yet $\mathcal{S}$ is a secure signature scheme. If PoC-PKI does not achieves property ξ, then there exists an adversary $\mathcal{A}$ that satisfies

$$Pr\left[\mathbf{SecExp}^{\xi,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N}) = 1\right] \notin Negl(1^\kappa) \tag{5}$$

However, for each property $\xi \in \{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{ReACC}, \Delta\mathsf{ReTRA}\}$, the matching security experiment $\mathbf{SecExp}^{\xi,\mathcal{M}}_{\mathcal{A},\text{PoC-PKI}^{\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N})$ contains the algorithm call PoC-PKI.$\mathsf{WasValid}(\psi, \xi)$. Following the implementation of PoC-PKI.$\mathsf{WasValid}(\psi, \xi)$ (described in Alg. 16), the algorithm executes

$$\mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota}, (Core(\psi), \xi, \eta.clk), \eta.\sigma) \tag{6}$$

for $\eta = \psi.\rho[\xi]$. Thus, following Eq. 5, $\mathcal{A}$ is a PPT adversary that achieves

$$\mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota}, (Core(\psi), \xi, \eta.clk), \eta.\sigma) = \top \tag{7}$$

However, this means that we can use $\mathcal{A}$ to construct a PPT adversary $\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}$ that breaks the existential unforgeability of $\mathcal{S}$.

First, since $\mathcal{A}$ is polynomial then $\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}$ is also polynomial. Second, since $\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}(1^\kappa, \xi, vk, \mathsf{N})$ simulates $\mathcal{A}$ with the public verification key vk for $\iota \in \mathsf{N}$, then

$$S.PubInfo.pk_{\eta.\iota} = vk \tag{8}$$

and the output of AdvEU is

$$(m = (Core(\psi), \xi, \psi.\rho[\xi].clk), \sigma = \psi.\rho[\xi].\sigma) \tag{9}$$

Therefore, combining Eq. 7,8 and 9, we get

$$\begin{aligned} \mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota}, (Core(\psi), \xi, \eta.clk), \eta.\sigma) &= \\ \mathcal{S}.\mathsf{Ver}(vk, m, \sigma) &= \top \end{aligned} \tag{10}$$

Finally, the AdvEU adversary does not *always* successfully output a pair (m, σ), because the authority chosen in line 1 of the algorithm might not be the honest authority chosen by $\mathcal{A}$ during the execution of $\overline{\text{PoC-PKI}}^{OSign(\cdot)}_{\iota,vk}$ (captured

by line 4 of the AdvEU algorithm). But, since the authorities chosen by AdvEU and $\mathcal{A}$ are chosen independently, the probability this happens is only $1/|\mathsf{N}|$, i.e., we still have a non-negligible probability to succeed. Thus, combined with Eq. 5 and 10, we constructed an adversary AdvEU that satisfies

$$Pr\left[\mathbf{Exp}^{EU}_{\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}},\mathcal{S}}(1^\kappa)=1\right] \notin Negl(1^\kappa) \tag{11}$$

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

Proving PoC-PKI's properties through reduction to a secure signature scheme. We now apply Claim 1 to argue that PoC-PKI achieves accountability, Δ-transparency, revocation accountability and Δ-revocation transparency. Since PoC-PKI achieves each of these attributes using the signature scheme $\mathcal{S}$ slightly differently, we provide a separate claim for each attribtue. In each claim we first explain how PoC-PKI uses the signature scheme $\mathcal{S}$ to achieve the specific property and then employ the reduction described in the claim to argue that PoC-PKI indeed achieves the specific attribute.

Claim 2. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves accountability 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 secure, robust threshold signature scheme.

Proof. In PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$, the only way to generate a valid accountable certificate ψ, is by invoking the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Issue algorithm on authority ι which is authorized to issue ψ. According to the implementation described in Alg. 11, the algorithm PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Issue uses the secure $\mathcal{S}$.Sign algorithm to generate the proof that ψ is an accountable certificate issued by ι.

Assume to the contrary that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *does not* achieves accountability. However, following Claim 1, if such adversary $\mathcal{A}$ exists, we can use $\mathcal{A}$ to build $\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}$ that breaks the existential unforgeability of the secure signature scheme $\mathcal{S}$.

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

Claim 3. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves revocation accountability 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 secure, robust threshold signature scheme.

Proof. In PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$, the only way to revoke a certificate ψ, is by invoking the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Revoke algorithm on $\psi.\rho[\mathsf{ACC}].\iota$ (the issuer of ψ). According to the implementation described in Alg. 13, the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$.Revoke algorithm uses the secure $\mathcal{S}$.Sign algorithm to generate the proof that ψ was revoked by ι.

Assume to the contrary that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *does not* achieves revocation accountability. However, following Claim 1, if such adversary $\mathcal{A}$ exists, we can use $\mathcal{A}$ to build $\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}$ that breaks the existential unforgeability of the secure signature scheme $\mathcal{S}$.

Therefore, PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves revocation accountability. □

Claim 4. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves $\Delta_{\text{PoC-PKI}}$-transparency 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 secure, robust threshold signature scheme.

Proof. In PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$, the only way to generate a valid transparent certificate ψ, is by invoking the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .Upgrade algorithm on authority ι. According to the implementation described in Alg. 15, the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .Upgrade algorithm uses the secure $\mathcal{S}$.Sign algorithm to generate the proof that ψ is a transparent certificate.

Assume to the contrary that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *does not* achieves $\Delta_{\text{PoC-PKI}}$-transparency. However, following Claim 1, if such adversary $\mathcal{A}$ exists, we can use $\mathcal{A}$ to build $\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}$ that breaks the existential unforgeability of the secure signature scheme $\mathcal{S}$.

Therefore, PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves $\Delta_{\text{PoC-PKI}}$-transparency. □

Claim 5. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves $\Delta_{\text{PoC-PKI}}$-revocation transparency 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 secure, robust threshold signature scheme.

Proof. In PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$, the only way to achieve $\Delta_{\text{PoC-PKI}}$-revocation transparency is by invoking the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .Upgrade algorithm on authority ι. According to the implementation described in Alg. 15, the PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ .Upgrade algorithm uses the secure $\mathcal{S}$.Sign algorithm to generate the proof that ψ is $\Delta_{\text{PoC-PKI}}$-transparently revoked by ι.

Assume to the contrary that PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ *does not* achieves $\Delta_{\text{PoC-PKI}}$-revocation transparency. Namely: However, following Claim 1, if such adversary $\mathcal{A}$ exists, we can use $\mathcal{A}$ to build $\mathsf{AdvEU}^{OSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}$ that breaks the existential unforgeability of the secure signature scheme $\mathcal{S}$.

Therefore, PoC-PKI achieves $\Delta_{\text{PoC-PKI}}$-revocation transparency. □

B.2 Proof of Non-Equivocation

Proving that PoC-PKI achieves equivocation prevention is different from proving the other properties, because PoC-PKI prevents equivocation using both a secure encryption scheme $\mathcal{E}$ and a robust (t,n)-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 achieves equivocation prevention, we use the following methodology:

1. We define a variation of PoC-PKI called $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ where equivocation prevention relies solely on the threshold scheme $\mathcal{TS}$ and not on a secure encryption scheme (Def. 10).
2. Then, we show that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ achieves equivocation prevention, see Claims 6,7.
3. Finally, we show that the security argument also holds for the original PoC-PKI scheme, see Claim 8.

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 $\overline{\mathcal{TS}\text{-PoC-PKI}}^{OTSign(\cdot)}$ scheme, where encryption is not used for equivocation prevention. The fact that $\overline{\mathcal{TS}\text{-PoC-PKI}}^{OTSign(\cdot)}$ only slightly varies from PoC-PKI, allows us to prove that equivocation prevention can be achieved in PoC-PKI using a secure threshold signature scheme, without (at first) the need to handle the security of the encryption scheme so that our proof methodology resembles one for a standard signature scheme. Lastly, we show that if we add the encryption back to the $\overline{\mathcal{TS}\text{-PoC-PKI}}^{OTSign(\cdot)}$ scheme, thus ending up the with the original PoC-PKI scheme, the security argument that PoC-PKI achieves equivocation prevention still holds, as long as the encryption scheme is secure.

The $\overline{\mathcal{TS}\text{-PoC-PKI}}^{OTSign(\cdot)}$ scheme. We start by defining the variation of the PoC-PKI, called $\overline{\mathcal{TS}\text{-PoC-PKI}}^{OTSign(\cdot)}$.

Definition 10. *Let $\mathcal{S}$,$\mathcal{E}$ and $\mathcal{TS}$ be a signature, encryption and threshold-signature schemes, respectively. Given a PPT $OTSign(\cdot)$ oracle, let $\overline{\mathcal{TS}\text{-PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign(\cdot)}$ (abbreviated as $\overline{\mathcal{TS}\text{-PoC-PKI}}^{OTSign(\cdot)}$) be a PKI scheme identical to* PoC-PKI*, except for the following changes:*

1. *In the $\overline{\mathcal{TS}\text{-PoC-PKI}}^{OTSign(\cdot)}.\mathsf{GroupGen}$ algorithm, replace the following line (line 2 in Alg. 10):*

$$H = \{h_i \leftarrow \mathcal{E}.\mathsf{Enc}(PubInfo_i.ek, tsk_i)\}_{i\in\mathsf{N}} \tag{12}$$

with the following code:

$$H = \{h_i \leftarrow \mathcal{E}.\mathsf{Enc}(PubInfo_i.ek, \text{`0'})\}_{i\in\mathsf{N}} \tag{13}$$

2. *In the $\overline{\mathcal{TS}\text{-PoC-PKI}}^{OTSign(\cdot)}.\mathsf{Incoming}$ algorithm, replace the following line of code:*

$$res \leftarrow (\psi, \sigma = \mathcal{TS}.\mathsf{Sign}(S.PrivInfo.tsk, (Core(\psi), \mathsf{EQ\text{-}P}))) \tag{14}$$

with the following line of code:

$$res \leftarrow (\psi, \sigma = OTSign((Core(\psi), \mathsf{EQ\text{-}P}))) \tag{15}$$

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

Note the two modifications that happen in $\overline{\mathcal{TS}\text{-PoC-PKI}}^{OTSign(\cdot)}$ as opposed to PoC-PKI. 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 $\mathcal{TS}.\mathsf{Sign}$ in the Incoming algorithm, the scheme uses the oracle $OTSign$. These two modifications essentially eliminate the part that the encryption scheme $\mathcal{E}$ plays in equivocation prevention, hence, in $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$, equivocation prevention is implemented solely using the threshold signature scheme $\mathcal{TS}$.

Proving that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ achieves equivocation prevention. We now start the second phase of our proof process, and begin by showing that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ 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 $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign(\cdot)}$ scheme uses $\mathcal{TS}$, then no PPT adversary can *abuse* $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ to generate two conflicting certificates ψ,ψ' with the non-equivocation attribute.

Proof. The only place in $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ where authorities generate partial signatures is in line 1.1.6 of the Incoming algorithm (Alg. 17), where an authority generates a share for the signature proof of a non-equivocal certificate ψ. However, this line is executed only if the check in line 1.1.3 is satisfied, i.e., there is no conflicting certificate ψ' in the $S.certs$ repository. In other words, if line 1.1.3 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 1.1.6, 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:

$$n_\psi + n_{\psi'} \leq |\mathsf{N}| - f \tag{16}$$

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. 16:

$$n_{\psi'} \leq |\mathsf{N}| - f - n_\psi = 0 \tag{17}$$

hence, the total number of shares of signatures for ψ' is at most f (i.e., only from the malicious authorities). Since following Def. 7, 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'. □

The AdvEU-TS adversary. We now describe the $\mathsf{AdvEU\text{-}TS}^{OTSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}$ algorithm (see Alg. 19), referred to as AdvEU-TS for brevity. The algorithm takes as input a security parameter 1^κ, a public group verification key tvk, and a set of authorities N. In addition, the algorithm takes an adversary $\mathcal{A}$, a PoC-PKI implementation, a secure signature scheme $\mathcal{S}$, a secure encryption scheme $\mathcal{E}$, a robust threshold signature scheme $\mathcal{TS}$, an execution model $\mathcal{M}$, and also has an oracle access to the signing function $\mathcal{TS}.\mathsf{Sign}$, with the (unknown) secret partial-signing key tsk_ι where $\iota \in \mathsf{N}$, represented by the $OTSign(\cdot)$ notation. The algorithm executes $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign(\cdot)}$ with adversary $\mathcal{A}$ using the **Exec** execution model (line 1), and in line 2 the algorithm extracts two certificates ψ,ψ' from the output of $\mathcal{A}$. A valid execution according to the model $\mathcal{M}$ is verified in line 3, along with the verification that the ψ,ψ' are conflicting certificates. In line 4, the algorithm outputs a message m and a signature σ based on ψ'.

Algorithm 19 $\mathsf{AdvEU\text{-}TS}^{OTSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}(1^\kappa, tvk, \mathsf{N})$

// Execute $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign(\cdot)}$

1: $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign(\cdot)}}(1^\kappa, \mathsf{N})$

// Extract output from the execution

2: $(\psi, \psi') \leftarrow Out_{\mathcal{A}}$

// Ensure the adversary followed the model

3: **if** $\mathcal{M}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \perp \ \vee \ \psi.id \neq \psi'.id \ \vee \ Core(\psi) = Core(\psi') \ \vee$
$\mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{EQ\text{-}P}) \neq \top \ \vee \ \mathcal{P}.\mathsf{WasValid}(\psi', \mathsf{EQ\text{-}P}) \neq \top$

3.1: Return $\perp$

// Output message m and forged signature σ over m based on ψ'

4: Output (m, σ) *s.t.* $m = (Core(\psi'), \mathsf{EQ\text{-}P}, \psi'.\rho[\mathsf{EQ\text{-}P}].clk)$ and $\sigma = \psi'.\rho[\mathsf{EQ\text{-}P}].\sigma$

We now complete the second phase of our proof by arguing that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ achieves equivocation prevention.

Claim 7. $\overline{\mathcal{TS}\text{- PoC-PKI}}^{\mathcal{S},\mathcal{E},\mathcal{TS},OTSign(\cdot)}$ achieves 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. To prove this claim, we demonstrate that if $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ does not achieves equivocation prevention, we can build an adversary that breaks the security of the threshold signature used in $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$.

Assume to the contrary that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ does not achieves equivocation prevention, thus, there exists an adversary $\mathcal{A}$ that satisfies:

$$Pr\left[\mathbf{SecExp}^{\mathsf{EQ\text{-}P},\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}}(1^\kappa, \mathsf{N}) = 1\right] \notin Negl(1^\kappa) \qquad (18)$$

Since $\mathcal{A}$ 'win' in the equivocation prevention security experiment (Eq. 18), then the conditions in lines $5.1 - 5.5$ of the security experiment (Alg. 7) must hold. In particular, $\mathcal{A}$ managed to produce two valid conflicting certificates ψ,ψ' with the equivocation prevention attribute EQ-P, namely:

$$\begin{aligned} &\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}.\mathsf{WasValid}(\psi, \mathsf{EQ\text{-}P}) = \\ &\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}.\mathsf{WasValid}(\psi', \mathsf{EQ\text{-}P}) = \top \end{aligned} \tag{19}$$

Hence, following Eq. 19 and the implementation of $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}.\mathsf{WasValid}$ described in Alg. 16, we get that:

$$\begin{aligned} \mathcal{TS}.\mathsf{Ver}(S.PubInfo.tvk, (Core(\psi), \mathsf{EQ\text{-}P}), \psi.\rho[\mathsf{EQ\text{-}P}].\sigma) \ &\equiv \\ \mathcal{TS}.\mathsf{Ver}(S.PubInfo.tvk, (Core(\psi'), \mathsf{EQ\text{-}P}), \psi'.\rho[\mathsf{EQ\text{-}P}].\sigma) \ &\equiv \ \top \end{aligned} \tag{20}$$

However, this means we can use $\mathcal{A}$ to construct a PPT adversary AdvEU-TS that breaks the security of $\mathcal{TS}$. Namely, since $\mathcal{A}$ can generate two conflicting certificates ψ,ψ' as described in Eq. 20 with non-negligible probability, and following Claim 6 that ψ' (without loss of generality) was not 'honestly' generated by $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ honest authorities, it shows that AdvEU-TS 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 $v = S.PubInfo.tvk$ and up to t oracle accesses on m, and therefore, following Def. 7:

$$Pr\left[\mathbf{Exp}^{EU}_{\mathcal{TS},\mathsf{AdvEU\text{-}TS}^{OTSign(\cdot),\mathcal{M}}_{\mathcal{A},\text{PoC-PKI},\mathcal{S},\mathcal{E},\mathcal{TS}}}(1^\kappa, \mathsf{N}, t) = 1\right] \in Negl(1^\kappa)$$

thus AdvEU-TS contradicts the unforgeability of $\mathcal{TS}$.

Therefore, $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ achieves equivocation prevention. □

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 $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ achieves equivocation prevention. To prove that PoC-PKI also achieves equivocation prevention, we need to show that the fact that PoC-PKI uses encryption to achieve equivocation prevention does not provide any advantage to the adversary in comparison to $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$.

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 $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ scheme, and if $b = 1$, we execute $\mathcal{A}$ with the PoC-PKI scheme.

3. $\mathcal{A}$ outputs b' $\in \{0, 1\}$.
4. The game outputs 1 if $b = b$', otherwise 0.

We now argue that PoC-PKI achieves equivocation prevention.

Claim 8. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves 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. The only difference between the PoC-PKI and the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ schemes is that the adversary has an advantage in PoC-PKI, because it also receives the encrypted secret information generated by PoC-PKI.GroupGen using the secure encryption scheme $\mathcal{E}$. Claim 7 shows that $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ achieves equivocation prevention, since $\mathcal{TS}$ is secure; we now show that security in the PoC-PKI 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 achieves equivocation prevention under the $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ model, namely:

$$Pr\left[\mathbf{SecExp}^{\mathsf{EQ\text{-}P},\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}}(1^\kappa, \mathsf{N}) = 1\right] \notin Negl(1^\kappa) \qquad (21)$$

Since the only difference between PoC-PKI and $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ 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 $\mathbf{SecExp}^{\mathsf{EQ\text{-}P},\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}}(1^\kappa, \mathsf{N})$ experiment (since we conclude it is an execution with the PoC-PKI scheme, where $\mathcal{A}$ has an advantage), and outputs $b' = 0$ otherwise (since it is probably an execution with the $\overline{\mathcal{TS}\text{- PoC-PKI}}^{OTSign(\cdot)}$ 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 also achieves equivocation prevention. □

B.3 Proof of Liveness

Recall that PoC-PKI employs an *immediate response* approach, where every algorithm's execution produces an immediate non-pending response, except for upgrading a certificate with the equivocation prevention attribute; in such a case, a pending certificates is generated first. Hence, the only property that we need to define the livness conditions for is equivocation prevention.

Proof methodology. We show that PoC-PKI achieves liveness of equivocation prevention in a two steps process. First, we argue that in any valid execution of PoC-PKI, where the upgrade liveness criteria are satisfied (i.e., there is no

valid reason not to upgrade a certificate ψ), ψ will be upgraded (see Claim 9). Then, we show that an adversary under PoC-PKI's threat model cannot prevent a valid equivocation prevention upgrade, thus resulting in PoC-PKI satisfying the liveness requirements of equivocation prevention (see Claim 10).

Claim 9. Let $[t, \mathsf{N_F}, Out_{\mathcal{A}}, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\text{PoC-PKI}}(1^{\kappa}, \mathsf{N})$ be an execution of PoC-PKI with adversary $\mathcal{A}$ for a set of authorities N and security parameter 1^{κ} under execution model **Exec**. Also, let authority $\iota \leftarrow Out_{\mathcal{A}}$ be an honest authority ($\iota \in \mathsf{N} - \mathsf{N_F}$). If the execution is a valid execution under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ and the liveness criteria of equivocation prevention is met, then on any time after $\psi.\rho[\mathsf{EQ\text{-}P}].clk + \Delta$, ι outputs a valid upgraded non-equivocal certificate. Namely:

$$\begin{pmatrix} \forall [t, \mathsf{N_F}, Out_{\mathcal{A}}, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\text{PoC-PKI}}(1^{\kappa}, \mathsf{N}), \iota \leftarrow Out_{\mathcal{A}}, \\ Clk^t_{\iota} > \psi.\rho[\mathsf{EQ\text{-}P}].clk + \Delta, \psi' \leftarrow Out^t_{\iota} \end{pmatrix}$$
$$\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \top \ \wedge \ \mathsf{LiveC}^{\text{PoC-PKI}}(\mathsf{N}, \mathsf{N_F}, Out_{\mathcal{A}}, R) = \top$$
$$\wedge \ \iota \in \mathsf{N} - \mathsf{N_F} \Rightarrow Core(\psi) = Core(\psi') \ \wedge \ \mathcal{P}.\mathsf{WasValid}(\psi', \mathsf{EQ\text{-}P}) = \top \quad (22)$$

Proof. Assume to the contrary that there exists an execution of PoC-PKI that negates the claim, i.e., certificate ψ is not upgraded with the $\mathsf{EQ\text{-}P}$ attribute, even though the liveness criteria was met. Following the liveness criteria $\mathsf{LiveC}^{\text{PoC-PKI}}$ described in Section 5.2, if the liveness criteria is met, then ι did not output any valid 'explanation' why the upgrade should fail, i.e., no existing valid conflicting certificate known to other authorities that prevents the requested upgrade.

However, this means that at least $|\mathsf{N}| - f - 1$ of the authorities sent approvals to ι (all the honest authorities), and they did so before the upgrade time has expired. Since there are only f malicious authorities and PoC-PKI does not accept more than one partial proof per authority, ι had enough valid partial signatures from honest authorities to combine, and no more than f invalid partial signatures; hence, ι would have generated a valid combined signature of non-equivocation. Hence, since ι is an honest authority, if the liveness criteria was met, ι would have outputted an upgraded certificate on any time $Clk^t_{\iota} \geq \psi.\rho[attr].clk + \Delta$, thus contradicting the assumption that an execution that negates the claim exists. □

Claim 10. PoC-PKI$^{\mathcal{S},\mathcal{E},\mathcal{TS}}$ achieves $\mathsf{LiveC}^{\text{PoC-PKI}}$-liveness under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$.

Proof. Since in PoC-PKI it holds that $|\mathsf{N}| > 3f$, any call to $\mathcal{TS}.\mathsf{Combine}$ is with at least $2f+1$ valid partial signatures, and therefore outputs a valid group signature, as long as $\mathcal{TS}$ is a robust (t, n)-threshold-signature.

Assume to the contrary that PoC-PKI does not achieves liveness of pending certificate upgrade for equivocation-prevention attribute. Then, there exists an adversary $\mathcal{A}$ such that

$$Pr\left[\mathbf{SecExp}^{\Delta\mathsf{LiveC}-liveness,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}}(1^{\kappa}, \mathsf{N}) = 1\right] \notin Negl(1^{\kappa}) \quad (23)$$

Thus, following experiment $\mathbf{SecExp}^{\Delta\mathsf{LiveC}-liveness,\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}}_{\mathcal{A},\text{PoC-PKI}}$ (Alg. 8), it means that although the liveness criteria $\mathsf{LiveC}(\mathsf{N},\mathsf{N_F},Out_{\mathcal{A}},R)$ was met, an upgraded certificate was not issued by an honest authority ι. However, following Claim 9 and PoC-PKI's model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$, we know that if $\mathsf{LiveC}(\mathsf{N},\mathsf{N_F},Out_{\mathcal{A}},R)=\top$, then ι has enough partial signature to successfully produce an upgraded certificate. Therefore, if such adversary $\mathcal{A}$ does exists, this means that $\mathcal{A}$ was able to *prevent* ι from combining using $\mathcal{TS}.\mathsf{Combine}$ enough valid partial signatures (at least $2f+1$) into a valid group signature with only f invalid partial signatures under model $\mathcal{M}^{\text{PoC-PKI}}_{\Delta_{com},\Delta_{clk}}$ that ensures reliable communication, thus contradicting the robustness of $\mathcal{TS}$.

Therefore, when using a robust threshold signature scheme, PoC-PKI achieves liveness of pending certificate upgrade for the equivocation-prevention attribute. □