Provable Security for PKI Schemes

It is possible to build a cabin with no foundations, but not a lasting building. - Eng. Isidor Goldreich [1].

Hemi Leibowitz
Dept. of Computer Science
Bar-Ilan University
Ramat Gan, Israel

Amir Herzberg
Dept. of Computer Science and Engineering
University of Connecticut
Storrs, CT, USA

Ewa Syta
Dept. of Computer Science
Trinity College
Hartford, CT, USA

***Abstract*—In this work we apply the systematic approach of game-based security specifications and proofs by reductions, to the design and evaluation of *public key infrastructure (PKI)* schemes. The importance of rigorous definitions and reduction-based proofs for cryptographic primitives is well-recognized, but this approach has not yet been applied to PKI schemes, despite their importance and pervasive use. This is most problematic in case of the advanced PKI properties such as *transparency, revocation transparency* and *non-equivocation*, which are non-trivial to define, analyze and prove.**

In response, we propose the first Public Identity Infrastructure (PII) framework that offers rigorous yet flexible game-based security for PKI schemes. We show the feasibility of the PII framework by presenting *United-*π, a simple, efficient and provably secure 'proof of concept' PKI scheme, that provably achieves all security properties we define.

***Index Terms*—**

I. Introduction

The security of our online infrastructure critically depends on the ability to securely distribute public keys. For example, the SSL/TLS protocol [2] underlines all secure (HTTPS) connections on the Internet providing authentication and encryption for domain validation and end-to-end security. For the client's browser to establish a secure connection to a server, the server must provide a digital certificate binding the server's identity to the provided public key. Such a certificate must be issued by a certificate authority (CA) trusted by the browser, either directly (a root CA) or indirectly (an intermediate CA). Unfortunately, current browsers trust hundreds of such CAs, any of which can issue a fake certificate that an attacker can subsequently use for 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. Over the years, we have seen many failures of the currently deployed CA-based system. For example, hackers stole the master keys of CAs [3], [4] and issued fake certificates for major websites and the CAs themselves abused their powers by improperly delegating their certificate-issuing authority [5].

A number of browser-based stopgap solutions, such as pinning certificates for specific websites [6] or following the "trust-on-first-use" (TOFU) model and trusting the first encountered certificate [7], were proposed, alongside more systematic attempts to improve the PKI such as Certificate Transparency [8], [9], Enhanced-CT [10], Sovereign Key [11], CONIKS [12], AKI [13], PoliCert [14], ARPKI [15], DTKI [16], CoSi [17], IKP [18], CertCoin [19], PB-PKI [20], Catena [21], CertLedger [22], and more. These proposals are designed to achieve additional, 'stronger' security properties as compared to X.509, such as non-equivocation and transparency.

Considering the wide use and importance of PKIs, and the known failures, it is remarkable that the security properties of PKIs have not been formally defined - and hence, also not proven - although the importance of provable-security is widely recognized. A possible explanation to this situation is the fact that most currently-deployed PKIs are based on the X.509 standard [23], which focuses on the basic security goals of *accountability* and *revocation accountability*. Both goals are simple and intuitive, and for X.509, follow immediately from the security of the underlying signature scheme. However, recent PKI schemes have more advanced goals, such as transparency and non-equivocation. Defining and proving security for these properties is more challenging; yet, similarly to X.509, no formal definitions or proofs are typically published. This makes it challenging to securely build systems, which depend on PKI schemes and their advanced features, and to compare and select a scheme that best fits a specific application or scenario.

Indeed, when we explored the 'PKI landscape' (see Section II), we found that while many PKI systems have similar ideas and goals, they are surprisingly non-trivial to compare against one another. In the absence of an agreed-upon list of formally-defined goals and requirements for a PKI scheme, existing systems establish their own security goals, which are often not well-defined and tied to a specific implementation. Inconsistent terminologies, different security and communication models, additional entities, and explicit and implicit assumptions, all further complicate such comparisons. Interestingly, certain PKI schemes do not provide properties that other systems consider necessary, even when they are straightforward to support, while other schemes achieve stronger properties without explicitly claiming them. Lastly, the lack of a formal framework for security of PKI schemes makes it hard 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 scheme.

Our goal is to lay a solid foundation for PKI by presenting the *Public Information Infrastructure (PII) framework*, with well-defined correctness, safety and liveness requirements, allowing for reduction-based proofs of security. The PII framework reflects the goals of recent, advanced PKI proposals, and yet it is general enough to allow future work to define additional requirements and properties.

We use the term PII instead of the traditional term *PKI* for two reasons. First, the security properties we focus on are not specific to the certification of public keys and apply to the certification of any type of public information; this was also done in X.509, with the introduction of attribute certificates. Second, similarly to other PKI proposals, PII focuses on *security against corrupt CAs*, but not on *limiting the certificates a particular CA is authorized for*. This is in contrast to PKIX [24] and other 'classical' PKI schemes, which focus on naming and other constraints for the certificates that a CA is authorized to issue. Such constraints are important, as they limit the scope of damages by a corrupt CA, and may be applied to any PII scheme as a complementary security measure.

To define the PII framework, we reviewed and analyzed a number of existing PKI schemes, as well as considered applications built upon or along-side existing PKIs, to ensure that the PII framework reflects current applications while remaining flexible to accommodate future uses and extensions. This allowed us to develop game-based definitions for what we identified as the main security requirements for PKI/PII schemes, namely: *accountability*, Δ*-transparency*, *non-equivocation*, *revocation accountability* and Δ*-revocation transparency*. We briefly explain these requirements, and map them to existing PKIs in section II and Table I.

Our PII framework does not attempt to address *all* PKI issues, but to provide a foundation to eventually do so. The set of requirements we propose should not be viewed as final and complete. Indeed, we intentionally designed this framework to be able to accommodate additional requirements and alternative definitions. Considering that this area has been a research focus for many years resulting in numerous designs, often taking different approaches and focusing on different issues, it would be overly optimistic to hope for one-size-fits-all solution. Instead, our goal is to open up, facilitate and suitably guide a debate and a collaborative effort to arrive at a framework with best-fitting requirements (or sets of requirements) and models. This may be compared to the extensive and ongoing efforts in defining different definitions and variants for various cryptographic schemes, e.g., signature and encryption schemes. There are important 'advanced' aspects of PKIs, which we leave, for this reason, to future work, e.g., privacy and cross-certification.

We aimed to provide a solid, precise but also flexible foundation. In particular, our framework supports different communication and adversary models. Furthermore, the PII framework includes not just correctness and safety properties, but also liveness properties, which are harder to capture formally, but important in practice. Our framework may be adopted to define security for other advanced cryptographic schemes or systems.

We show the practicality of our PII framework by presenting *United-*π, an efficient, provably-secure secure implementation of an PII framework. Our analysis and reduction-based proofs of security of *United-*π, prove the feasibility and applicability of our framework. While *United-*π is a 'proof of concept' PKI, it is efficient, and offers a simple yet flexible design.

In summary, this paper makes the following **contributions**: *(i) Public Identity Infrastructure (PII) framework:* a rigorous, flexible game-based security requirements that encompasses correctness, safety and liveness properties of PKI (and more generally, PII) schemes; *(ii) United-*π*, a provably-secure PII construction:* an efficient, usable 'proof-of-concept' PKI scheme with reduction-based proofs of the PII properties; and *(iii) Systematization of Knowledge:* a review of the *PKI landscape* and a comparison of the security properties of current schemes.

The paper is **organized** as follows. Section II reviews the PKI landscape and compares proposed schemes. Section III presents the execution model for PII. Section IV presents the Public Identity Infrastructure (PII) scheme and its correctness, safety and liveness requirements. In Section V, we present *United-*π, a provably-secure PII system; its proof sketches are included in Appendix A while rigorous proofs in the full version of this work [25]. We conclude and discuss future work in Section VII.

The paper is **organized** as follows. Section II reviews the PKI landscape and compares proposed schemes. Section III presents the adversary and system framework, allowing flexibility, e.g., synchronous and asynchronous models. Section IV presents the Public Identity Infrastructure (PII) scheme and its correctness, safety and liveness properties. In Section V, we present *United-*π, a provably-secure PII system; its analysis is in Section A. We conclude and discuss future work in Section VII.

For a quick first reading, the reader may want to first skip Section III and all but the first two subsections of section IV.

II. The PKI Landscape

In this section, we discuss the main PKI schemes proposed and deployed so far, and their security properties. Then, we map the security properties offered by these schemes to the properties identified in our framework (see Table I).

A. X.509, PKIX and 'Basic' PKI Safety Properties

The basic goal of a PKI scheme is to ensure *authenticity* of *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. The ITU X.509 specification [23] defines the main entities, properties and functionalities of a PKI as well as the specific format for certificates. A typical certificate contains an *identifier* and some *public information* (normally a public key), and a signature generated by a CA over the certificate's information, where the signature serves as the CA's endorsement of the mapping defined in the certificate. From version 3, X.509 supports certificate *extensions* which most current deployments use. The most widely used set of extensions is the IETF's PKIX [24], defined for use in Internet protocols, and its refinements for specific protocols, such as TLS [26] and S/MIME [27]. An important part of every PKI is the *validation process*, allowing *relying parties*, individuals or entities, to determine if a given certificate is valid in order to decide if to rely on its contents. Typically, a certificate is *valid* if it is unexpired, the certificate's signature is valid, and most importantly, the signature was generated by a trusted, 'authorized' CA, as defined for that PKI.

*Accountability (*ACC*).* The most fundamental security property of a PKI (and PII) scheme is *accountability* (ACC for short), i.e., the ability to identify the CA that issued a given certificate. Accountability provides a retroactive defense against a corrupt CA. In most PKI schemes, including X.509 and PKIX, 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 issuing a certificate to an impersonator. Note that we use the term accountability as a technical, well-defined property, which does not necessarily have any specific legal or financial implications. The widely-used notion of accountability is rather straightforward and we explicitly define it, we believe for the first time, in Section IV-D.

*Revocation accountability (*ReACC*).* A certificate can be considered valid only after its *issue date* and until its *expiration date*, both of which are 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 X.509 and PKIX are the *certification revocation lists (CRLs)* [28] and *online certificate status protocol (OCSP)* [29]. While the premise of revocation seems straightforward, it is surprisingly non-trivial to formulate its properties. Again, we are not aware of any existing such definition. We define two security requirements related to revocation, revocation accountability (ReACC, provided by X.509 and discussed next) and revocation transparency (ΔReTRA, not provided by X.509, and discussed in Section II-B).

Revocation accountability ensures accountability of the revoked certificates and as such, it is similar to the accountability of issuing certificates. Namely, this property 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.

B. Beyond X.509: Advanced PKI Safety Requirements

In Section II-A, we discussed accountability and revocation accountability, the two basic PKI properties provided by X.509 and PKIX. We now discuss more recent PKI schemes and their additional security requirements in terms of the properties we propose in PII: Δ-*transparency* (ΔTRA), Δ-*revocation transparency* (ΔReTRA), and *non-equivocation* (NEQ). See definitions of these requirements in Section IV-D.

Transparency (ΔTRA*).* Accountability, as described above, mainly serves as a deterrent against misbehavior and 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 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 [30]–[33]. Furthermore, 'punishment' could only be applied *after* the damage was already done and discovered - if it was discovered at all. An attacker or corrupt CA could reduce the risk of being caught by minimizing the exposure of the fraudulent certificate. Except for efforts such as the Perspectives Project [34], or the EFF SSL Observatory [35] that aim to gather and inspect *all* SSL certificates used in practice, the burden of detecting any fraudulent activity is mostly on the victim that receives a fraudulent certificate since browsers are not equipped with mechanisms to detect many types of fraudulent certificates, much less to report them to any 'enforcement agency', if one existed.

This significant issue has motivated more recent PKI designs, where certificates are *transparently published* to allow third parties (e.g., trusted 'monitors') to inspect and detect any fraudulent certificates. This design ensures that once a fraudulent certificate is issued by some corrupt authority, their misbehavior would eventually be detected and that fraudulent certificates could be found *before* they are misused.

We refer to this property as Δ-*transparency*. 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 include, or come with, a *commitment of* Δ-*transparency*, where some party is accountable to publish the certificate, immediately or within a specified time frame Δ. By demanding a proof of transparency, a PKI system can ensure detection of conflicting or fraudulent certificates issued by a corrupt or compromised CA, and empowers clients to detect and resolve any discrepancies regarding their certificates.

All schemes we analyzed but X.509 offer transparency, which is typically achieved through *append only public logs*, where a certificate is valid only if it appears in the public logs. Certificate Transparency (CT) [8] and Enhanced-CT [10] use Merkle trees for the logs, while CertCoin [19], PB-PKI [20], Catena [21] and CertLedger [22] use blockchain-based public logs (public ledgers).

*Non-equivocation (*NEQ*).* Transparency is still a *reactive defense*: it does not *prevent* a corrupt CA from issuing a fraudulent certificate, but only ensures that such behavior will eventually be detected. In other words, while transparency ensures detection, it does not ensure *prevention*. In particular, consider the goal of *non-equivocation*, i.e., preventing the issuance of multiple, concurrently valid certificates that map the same identifier to different public information (e.g., public keys). Non-equivocation can *prevent* a corrupt CA from issuing a fake certificate for an already-certified identifier, e.g., domain name; hence, it could *prevent*, rather than merely *detect*, man-in-the-middle attacks impersonating existing secure domains [36]. A system that achieves transparency, but not non-equivocation, would detect such attack, but may not be able to prevent it. Non-equivocation is typically achieved through the use of logging and public ledgers (e.g., CoSi [17], Catena [21], CertCoin [19]).

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 (ΔReTRA) ensures that if a CA revoked a certificate, then *all* authorities should be aware of the revocation (within bounded time), preventing such undesirable scenarios. Many systems [10]–[16], [19] provide ΔReTRA and it is an explicit goal of Revocation Transparency (RT) [9].

Certificate-status Freshness. Interestingly, X.509 provides a different but related property to revocation transparency, which we refer to as *certificate-status freshness*. X.509's two revocation mechanisms, CRLs and OCSP, include a *timestamp* in their responses, using which, relying parties may decide whether to trust the certificate, i.e., if the response is 'fresh' enough. More concretely, the relying party considers an X.509 certificate as valid *only* if it received a valid and sufficiently recently-issued ('fresh') CRL or OCSP response, indicating that the certificate was not revoked. Note that some relying parties may neglect to validate (the freshness of) the corresponding CRLs or OCSP responses, allowing for attacks using exposed but revoked keys; e.g., such failures are known for browsers. We find certificate-status freshness a rather straightforward and intuitive notion, which is easy to achieve; therefore, we *do not* formalize it as a separate property.

C. Current PKI Schemes and the PII Framework

Indisputably, achieving provable security is a complex task. Therefore, in this work, we strive to alleviate this burden by formalizing the meaning of a provably-secure PKI/PII system and providing a framework that both existing and future systems can use to achieve provable security in a simplified manner. Hence, instead of 'reinventing the wheel', we design PII in a way that *embraces*, *complements* and *reflects* current PKI designs, rather than trying to redefine or fundamentally alter what a PKI system is or should be.

We have methodically examined the existing 'PKI landscape' by identifying and analyzing current PKI systems, both those deployed and proposed in the literature. As a result, in addition to correctness and liveness requirements, we have identified five fundamental safety requirements (discussed in Sections II-A and II-B, and formally presented in Section IV) that a PKI system should provide.

We note that we aimed to be as careful and comprehensive as possible in deciding which PKI systems to include in our analysis (and Table I) but due to space constraints, we focused on deployed systems and other representative schemes.

First, following our discussion of the 'basic' PKI security properties in Section II-A, we observe that most systems, with the exception of Certcoin, PB-PKI, and Catena, aim to achieve accountability. Both Certcoin and PB-PKI build on top of Namecoin [], 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 and consequently, there is no need to hold anyone accountable for the process of assigning identifiers. Catena, on the other hand, 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.

Interestingly, many systems directly focus on more advanced properties, such as transparency and non-equivocation, and treat certain properties (e.g., accountability and revocation) as intrinsic to PKI and do not always explicitly state them. This phenomenon is especially apparent in case of revocation. Many systems [17]–[21] do not directly address revocation at all, or do not state the details of how revocation should be handled, by whom and under which conditions and simply treat it as an extension of issuing certificates. PKI systems, that use the X.509 notion of a certificate, implicitly rely on the X.509 revocation mechanisms (CRLs and OCSP). This approach is somewhat understandable due to the pervasiveness of X.509 but it also establishes the X.509 revocation mechanisms are the status quo of revocation despite its known weaknesses. In Table I, we label accountability and revocation accountability as intuitively true for all systems, except for Certcoin, PB-PKI and Catena, since these properties and the corresponding proofs are rather straightforward to reconstruct. Note that CONIKS, Certcoin and PB-PKI do allow clients that own certificates to revoke them, but revocation can also be done by an adversary that compromised the client's secret keys, or alternatively, the client may unable to perform revocation if the secret keys are lost.

Transparency, on the other hand, is a property that all systems, except for X.509, support. This is likely in response to one of the main weaknesses of X.509 widely abused in practice, i.e., a lack of a mechanism to effectively propagate all issued certificates among CAs and clients. As Table I indicates, all analyzed systems have informal security arguments for

	System [reference]	Safety requirements					Liveness requirements	Comments and additional PKI properties
		ACC	ΔTRA	NEQ	ReACC	ΔReTRA		
1	X.509 and PKIX, with CRL or OCSP	◐	n/s	n/s	◐	n/s	◐	Certificate-status freshness
2	CT [8] with RT [9]	◐	⊙	n/s	◐	⊙	◐	Proofs for logging properties [37]
3	Enhanced-CT [10]	◐	⊙	⋆ ⊙	◐	⊙	◐	
4	Sovereign Key [11]	◐	⊙	⋆ ⊙	◐	⊙	◐	
5	CONIKS [12]	◐	⊙	⋆ ⊙	◐	⊙	◐	Privacy
6	AKI [13]	◐	⊙	⋆ ⊙	◐	⊙	◐	
7	PoliCert [14]	◐	⊙	n/s	◐	⊙	◐	
8	ARPKI [15]	◐	⊙	n/s	◐	⊙	◐	Symbolic proofs
9	DTKI [16]	◐	⊙	⋆ ⊙	◐	⊙	◐	Symbolic proofs, anti-oligpoly
10	CoSi [17]	◐	⊙	⊙	n/s	n/s	◐	
11	IKP [18]	◐	⊙	⊙	n/s	n/s	◐	
12	CertCoin [19] and PB-PKI [20]	n/s	⊙	⊙	n/s	n/s	◐	
13	CertLedger [22]	◐	⊙	⊙	⊙	⊙	◐	
14	Catena [21]	n/s	⊙	⊙	n/s	n/s	◐	
15	*United-π* (this work)	●	●	●	●	●	●	

TABLE I: PKI security requirements. ● - reduction-based proof, ◐ - intuitively true, ⊙ - security arguments (a proof may require assumptions), n/s - not supported, ⋆ - detection but not prevention

transparency, except for CT, ARPKI and DTKI, all of which provide more substantial security analyses. We note, however, that the properties and their proofs are not specific to PKIs per se. Rather, they focus on a specific way of *achieving* a secure PKI. Specifically, Dowling et al. [37] formalized security properties and provided reduction-based proofs for logging schemes such as CT, that cover two classes of security goals involving malicious loggers and malicious monitors. ARPKI and DTKI, on the other hand, verify their core security properties using automated symbolic proofs via the Tamarin prover [38]. We view this approach as an important effort but not as a replacement for game-based security, however.

We define non-equivocation as an active measure that *prevents* the existence of two conflicting certificates, e.g., two certificates with the same identifier mapping to two different public keys, and a number of systems [17]–[22] provide this property as such. Other systems [10]–[13], [16], however, provide only *detecting* such conflicting certificates while referring to this property as non-equivocation. This example further illustrates the need for a well-defined set of requirements and their definitions.

Unlike revocation accountability, revocation transparency is more complicated, both to formalize and achieve. Typically, revocation mechanisms do not intuitively indicate that they provide revocation transparency, even if some form of propagation of revocation information is implemented. Given that these mechanisms are more complex and subtle, they need explicit definitions and analyses. Therefore, for all systems that support revocation transparency [9]–[16], [19], we decided against using the 'intuitively true' symbol. Several systems do not discuss revocation transparency at all, such as X.509 and [17]–[21], even though is certain cases (especially [17] it would be realatively easy to achieve). CT originally did not have a built-in support for revocation transparency, and it was only later formalized in [9].

Lastly, security requirements for cryptographic schemes often consider only *safety requirements*; however, in practice, *liveness* is essential but 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/PII 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 I, 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 IV-E).

D. Out of Scope: Constraints on Certificates

The X.509 standard and the PKI schemes based on it, most notably PKIX [24], define additional *basic* and *naming* certificates constraints which are used, in addition to the previously discussed checks, to determine if a particular certificate is valid. These constraints do not apply to the *root* CAs, i.e., the CAs which are trusted directly by the relying party. Rather, they apply to the *intermediate* CAs endorsed by the root CAs. These constraints are specified in a certificate and play an important role in defining CAs and constraining their authority. The basic constraints indicate if the specific entity is a CA and whether it can designate other entities as CAs. The naming constraints, on the other hand, restrict the identifiers (name spaces) that a specific CA can issue certificates for; without such constraint, a CA can issue a certificate for *any* identifier. While these constraints limit the impact of a corrupt CA, they do not detect or prevent misbehavior, which is the focus of this work and of recently proposed PKI schemes. Specifically, these constraints may prevent a CA from issuing some certificates, e.g., to a domain not assigned to it, but they do not provide defenses against a corrupt CA issuing certificates with incorrect information within the domains assigned to it or not revoking a certificate despite a user's request, for example. Consequently, we focus on *defenses against a corrupt CA* which is complementary to the efforts to impose constraints on the certificates issued by a CA.

III. Entities and Execution Model

In this section, we define the execution model for the PII framework. The PII execution model is unique in its flexibility, supporting a variety of network, synchronization and adversary models. In particular, the PII execution model allows synchronous, partially synchronous or asynchronous communication models, and adversarial models ranging from passive eavesdropper to active MitM adversaries. The methodology of such a flexible execution model may be applicable to other systems beyond PKI/PII schemes.

A. Entities and Their State.

In PII, 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.). We denote the set of all authorities in the system as $\mathsf{N} = \{1, \ldots, \mathsf{n}\}$, where n is the number of all authorities.

Authorities may interact with one another to perform certain actions. Each authority has local clock, as well as a local state S, e.g. containing issued certificates, and the following information:

- $S.\iota$: the unique-identifier of the entity.
- $S.PrivInfo$: private (secret) information.
- $S.PubInfo$: public information.

Each authority supports several operations. Most of these operations are specific to PII/PKI schemes, and defined later. Below, we define three not PII-specific operations, performed locally by each authority: Gen, Time and 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 entities and output out.
- $\mathsf{Incoming}(S, clk, \{\widetilde{m}_i\}_{i\in\mathsf{N}}) \rightarrow (S', \{m_i\}_{i\in\mathsf{N}}, out)$: The 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, GroupGen, which generates common public information, such as a system-wide public key or a set of 'root' public keys. The GroupGen operation is invoked as a part of our execution model; to implement this in a real system, GroupGen could be run by a trusted third party, or using an appropriate multi-party computation protocol. The GroupGen algorithm takes as input security parameter 1^κ, set of authorities N, the number of malicious authorities $f < n$, and a set of public information $\{PubInfo_i\}_{i\in\mathsf{N}}$. GroupGen has only one output, the public group information $PubInfo$. Namely, $\mathsf{GroupGen}(1^\kappa, \mathsf{N}, f, \{PubInfo_i\}_{i\in\mathsf{N}}) \rightarrow PubInfo$.

B. 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 implementation of PII, respectively. This algorithm takes as input a security parameter 1^κ, the set of authorities N and the number of malicious authorities f.

The **Exec** algorithm begins with an *initialization phase* (lines 1-7). During the initialization phase, the Gen algorithm is first used to initialize the local state of all authorities, followed by the GroupGen algorithm, which generate the public information $PubInfo$, shared by all authorities. Finally (line 7), the adversary is invoked, providing it with all public information, and allowing it to set the inputs for the first round of the execution phase.

The *execution phase* (lines 8-9) is a loop of *rounds*, with incrementing round number t. In each round, each authority $i \in \mathsf{N}$ handles three events. First (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 8.3 (execution). This is intended for invoking the algorithms in PII which are not executed by the execution process directly (i.e., excluding Gen, GroupGen, Time, and Incoming). Second (8.2.2), we invoke the $\mathcal{P}$.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 (8.2.3), we invoke the $\mathcal{P}$.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 a MitM 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: $[t, Out_\mathcal{A}, \iota, R]$, where t is the number of rounds in the execution, $Out_\mathcal{A}$ is the adversary's output, ι is a specific (honest) authority, and R is the *run*, namely, the 'transcript' of the execution (line 10). Authority ι was chosen by the adversary in the beginning of the execution (line 1), before any other operation was invoked. Further, we mark the local state of the chosen authority ι (line 3). While asking the adversary to pick ι does not impact the execution of $\mathbf{Exec}_{\mathcal{A},\mathcal{P}}$, it enables a simple and coherent presentation of definitions and arguments, like the safety requirements of PII in Section IV-D and the liveness requirements in Section IV-E.

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

```
      // Ask the adversary to pick an honest authority from N
 1:   $\iota \leftarrow \mathcal{A}(\mathsf{N})$
      // Initialize local variables for all authorities
 2:   $\forall i \in \mathsf{N} : S_i^0 \leftarrow \bot, Clk_i^0 \leftarrow 0, \left(S_i^0.PrivInfo, S_i^0.PubInfo\right) \leftarrow \mathcal{P}.\mathsf{Gen}_i\left(1^\kappa\right)$
      // Mark the authority chosen by the adversary
 3:   $S_\iota^0.chosen = \top$
      // Generate public group information
 4:   $PubInfo \leftarrow \left(\mathcal{P}.\mathsf{GroupGen}(1^\kappa, \mathsf{N}, f, \{S_i^0.PubInfo\}_{i \in \mathsf{N}}), \{S_i^0.PubInfo\}_{i \in \mathsf{N}}\right)$
 5:   $\forall i \in \mathsf{N} : S_i^0.PubInfo \leftarrow PubInfo$
      // Initialize the round indicator $t$ and adversarial state $S_\mathcal{A}^0$
 6:   $t \leftarrow 0, S_\mathcal{A}^0 \leftarrow \bot$
      // Invoking the adversary before the first round
 7:   $[\{Alg_i^1, Inp_i^1, Clk_i^1\}_{i \in \mathsf{N}}, \{\widetilde{m}_{i,j}^0\}_{i,j \in \mathsf{N}}, S_\mathcal{A}^1] \leftarrow \mathcal{A}(1^\kappa, S_\mathcal{A}^0, \iota, PubInfo)$
      // 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:        $\left(S_1, \{m_{i,j}^1\}_{j \in \mathsf{N}}, Out_1\right) \leftarrow \mathcal{P}.Alg_i^t\left(S_i^{t-1}, Clk_i^t, Inp_i^t\right)$
8.2.2:        $\left(S_2, \{m_{i,j}^2\}_{j \in \mathsf{N}}, Out_2\right) \leftarrow \mathcal{P}.Incoming\left(S_1, Clk_i^t, \{\widetilde{m}_{i,j}^{t-1}\}_{j \in \mathsf{N}}\right)$
8.2.3:        $\left(S_i^t, \{m_{i,j}^3\}_{j \in \mathsf{N}}, Out_3\right) \leftarrow \mathcal{P}.Time\left(S_2, Clk_i^t\right)$
              // Aggregate all outputted values during round $t$ into a single set
8.2.4:        $Out_i^t \leftarrow Out_1 \cup Out_2 \cup Out_3, (\forall j) m_{i,j} = \bigcup_{\ell=1}^3 m_{i,j}^\ell$
          // Adversary sees and controls messages, and selects operations for next round
8.3:      $[\{Alg_i^{t+1}, Inp_i^{t+1}, Clk_i^{t+1}\}_{i \in \mathsf{N}}, \{\widetilde{m}_{i,j}^t\}_{i,j \in \mathsf{N}}, S_\mathcal{A}^{t+1},$
              $Abort, Out_\mathcal{A}] \leftarrow \mathcal{A}(1^\kappa, \{m_{i,j}\}_{i,j \in \mathsf{N}}, S_\mathcal{A}^t)$
      // Repeat, until the adversary indicates termination via the $Abort$ indicator
 9:   until $Abort \neq \bot$
      // Output:
10:   $R \leftarrow \left\{\left\{Alg_i^{\hat{t}}, Inp_i^{\hat{t}}, Out_i^{\hat{t}}, Clk_i^{\hat{t}}\right\}_{i \in \mathsf{N}}, \left\{m_{i,j}^{\hat{t}}, \widetilde{m}_{i,j}^{\hat{t}}\right\}_{i,j \in \mathsf{N}}\right\}_{\hat{t} \in \{1 \ldots t\}}$
      // The algorithm outputs the number of rounds $t$, output values chosen by the
      adversary $Out_\mathcal{A}$, the target honest authority $\iota$ chosen by the adversary and
      aggregation of all execution details $R$.
11:   Return $[t, Out_\mathcal{A}, \iota, R]$
```

C. Adversary, Communication and Synchronization Models

The execution model provides a significant flexibility, allowing its use for schemes designed for different communication, synchronization and adversary models, e.g., MitM vs. eavesdropping adversary. This is achieved by defining a *model predicate* $\mathcal{M}$ over runs, which returns *true* if, and only if, the run conforms with the model. Here are some examples:

Synchronous model: a run is valid only if the Clk_i^t are synchronized, i.e., for every i, j, t holds $Clk_i^t = Clk_j^t$.

Bounded-delay reliable communication: a run is valid only if every message sent is received within (say) one time unit, i.e., if $m \in m_{i,j}^t$ then for some $t' > t$ holds $m \in \widetilde{m}_{i,j}^{t'}$ and $Clk_j^{t'} \leq Clk_j^t + 1$.

Eavesdropper adversary: a run is valid only if messages are transmitted correctly, i.e., there is a one-to-one mapping Π between the set of messages sent by i to j ($\bigcup_t m_{i,j}^t$) and the set of messages received by j from i ($\bigcup_t \widetilde{m}_{j,i}^t$).

IV. PII Framework

In this section, we describe the Public Identity Infrastructure (PII) framework. We first provide an overview of the framework and then define the correctness, safety and liveness requirements.

A. Certificates and Functionalities

A PII 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 PII defines (see Section IV-B). 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 PII. We assume that each implementation of PII 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 PII, and each implementation can define its specific $\mathcal{P}.\mathsf{WasValid}$ to reflect its notion of a certificate validity. In PII, 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 [39]. 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 $\mathsf{AttrSet} = \{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{NEQ}, \mathsf{REV}, \Delta\mathsf{ReTRA}\}$ to support the safety requirements of accountability, Δ-transparency, non-equivocation, revocation accountability, and Δ-revocation accountability, 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* (NEQ attribute) certificate carries a guarantee that another valid 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 also returns a proof of non-revocation until the current time; otherwise, the algorithms returns $\perp$.

Pending Certificates. In our execution model, all PII 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.

B. PII Algorithms

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

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

PII follows the execution model described in Section III, 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.

$\mathsf{WasValid}(\psi, [attr, tms]) \to \top/Pending/\perp$: 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 $\perp$.

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' and output messages $\{m_i\}_{i\in\mathsf{N}}$. 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.

$\mathsf{Issue}(id, pub, sd, ed) \to \psi/\psi_p/\perp$: The algorithm takes as input an identity id, public information pub, start date sd and end date ed, and outputs a certificate ψ for (id, pub) that is valid 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 $\perp$.

$\mathsf{Upgrade}(\psi, attr, [\alpha]) \to \psi\text{'}/\psi_p/\perp$: 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 $\perp$.

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

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

$\mathsf{Query}(id) \rightarrow \{\psi\}/\bot$: 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 $\bot$.

C. Correctness Requirements

We next describe the PII correctness requirements for issuing, revoking, upgrading, and the revocation status check operation. The requirements specify that if an operation does not fail (return $\bot$), 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. It is also convenient to denote $Core(\bot) = \bot$.

Requirement 1. *Correctness of certificate issuance.*

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

Requirement 2. *Correctness of certificate revocation.*

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

Requirement 3. *Correctness of certificate upgrade.*

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

Correctness of certificate revocation status should return either a revoked or a non-revoked certificate, with same core.

Requirement 4. *Correctness of certificate revocation status.*

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

D. Safety Requirements

We now define the PII safety requirements. For each requirement ξ, we define a corresponding experiment $\mathbf{SafetyExp}^{\xi,\mathcal{M}}_{\mathcal{A},\mathcal{P}}$, where model $\mathcal{M}$ is as described in subsection III-C, and use it to define the requirement using the following 'generic' definition.

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

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

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

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

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

The $\mathbf{SafetyExp}^{\mathsf{ACC},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f)$ game initializes the experiment by calling the $\mathbf{Exec}$ algorithm, which simulates the adversary. In return, the $\mathbf{Exec}$ algorithm outputs the number of rounds t simulated in $\mathbf{Exec}$, adversary output $Out_{\mathcal{A}}$, an authority identifier ι, and R, which is the 'transcript' of the simulation (lines 1,3). The experiment verifies that the run followed the model $\mathcal{M}$ (line 2); if not, abort with output $\bot$.

For a run which follows the model, we 'parse' R to find the operations and inputs $\left\{Alg_i^{\hat{t}}, Inp_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1...t\}}$ (line 3), and use ψ to denote the certificate returned by the adversary in $Out_{\mathcal{A}}$ (line 4).

The adversary *wins* if ψ is a valid accountable certificate issued by the honest authority ι (line 5.1), 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 5.2). Note that we use $\mathbf{Corrupted}(R)$ to denote the set of authorities corrupted during R.

Requirement 6. Δ-*Transparency (*ΔTRA*).* Adversary $\mathcal{A}$ wins in the Δ-*Transparency experiment* $\mathbf{SafetyExp}^{\Delta\mathsf{TRA},\mathcal{M}}_{\mathcal{A},\mathcal{P}}$, 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 3.

Algorithm 3 $\mathbf{SafetyExp}^{\Delta\mathsf{TRA},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f)$

```
1: $[t, Out_{\mathcal{A}}, \iota, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f)$
2: if $\mathcal{M}(R) = \bot$ then Return $\bot$
3: $\left\{Alg_i^{\hat{t}}, Inp_i^{\hat{t}}, Out_i^{\hat{t}}, Clk_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1...t\}} \leftarrow R$
4: $\psi \leftarrow Out_{\mathcal{A}}$
5: Return:
      // $\psi$ is a valid transparent certificate and both $\iota$ and $\psi.\rho[\Delta\mathsf{TRA}].\iota$ are honest
5.1:  $\mathcal{P}.\mathsf{WasValid}(\psi, \Delta\mathsf{TRA})\ \wedge\ \iota, \psi.\rho[\Delta\mathsf{TRA}].\iota \in \mathsf{N} - \mathbf{Corrupted}(R)\ \wedge$
      // Honest authority $\iota$ is not aware of $\psi$ although it should
5.2:  $Clk_\iota^t \geq \psi.\rho[\Delta\mathsf{TRA}].clk + \Delta\ \wedge$
5.3:  $[Alg_\iota^t, Inp_\iota^t] = [\mathcal{P}.\mathsf{Query}, \psi.id]\ \wedge$
5.4:  $\nexists\psi\text{'} \in Out_\iota^t\ s.t.\ Core(\psi) = Core(\psi\text{'})$
```

Transparency: Requirement 6 presented in Algorithm 3. The $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\Delta\mathsf{TRA},\mathcal{M}}(1^\kappa, \mathsf{N})$ game executes adversary $\mathcal{A}$ using $\textbf{Exec}$, verifies that $\mathcal{A}$ followed the model $\mathcal{M}$, and extracts the execution details as described in $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{ACC},\mathcal{M}}$.

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 5.1). Notice that the game checks that the transparency issuer is honest and not the issuer of the certificate itself (i.e., accountability issuer), 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. The third requirement is that there is an honest authority ι that is not aware of the transparency of ψ on time t (lines 5.2-5.3). This requirement is validated by verifying that on time t authority ι was instructed to perform $\mathcal{P}$.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 $\mathcal{P}$.Query algorithm outputs all certificates associated with $\psi.id$, this is a proof that an honest authority is not aware of ψ.

Requirement 7. *Non-equivocation (*NEQ*).* Adversary $\mathcal{A}$ wins in the NEQ *experiment* $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{NEQ},\mathcal{M}}$ 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 4.

Algorithm 4 $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{NEQ},\mathcal{M}}(1^\kappa, \mathsf{N}, f)$

```
1: $[t, Out_{\mathcal{A}}, \iota, R] \leftarrow \textbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f)$
2: if $\mathcal{M}(R) = \perp$ then Return $\perp$
3: $\left\{Out_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
4: $\psi \leftarrow Out_{\mathcal{A}}$
5: $\psi' \leftarrow Out_\iota^t$
6: Return:
        // $\psi$ and $\psi$' are both non-equivocal certificates with same identifier
6.1:    $\mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{NEQ}) \wedge \mathcal{P}.\mathsf{WasValid}(\psi', \mathsf{NEQ}) \wedge \psi.id = \psi'.id \wedge$
        // $\psi$ and $\psi$' have overlapping validity periods, different PubInfo
6.2:    $\psi.sd < \psi'.sd < \psi.ed \wedge \psi.pub \neq \psi'.pub \wedge$
        // The run did not include a revocation of $\psi$
6.3:    $(\nexists t' \leq t, j)\left(\psi_r \leftarrow Out_j^{t'} ; \mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \wedge Core(\psi_r) = Core(\psi)\right)$
```

Non-equivocation: Requirement 7 presented in Algorithm 4. The $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{NEQ},\mathcal{M}}(1^\kappa, \mathsf{N})$ game executes adversary $\mathcal{A}$ using $\textbf{Exec}$, verifies that $\mathcal{A}$ followed the model $\mathcal{M}$, and extracts the execution details as described in $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{ACC},\mathcal{M}}$.

The adversary wins the game if three requirements are met. First, ψ and ψ' must be valid certificates (ψ provided by the adversary and ψ' by some ι, lines 4-5) with the non-equivocation attribute (line 6.1). Second, both certificates have overlapping validity periods (line 6.2). Third, ψ must also be a non-revoked certificate (line 6.3).

Requirement 8. *Revocation accountability (*ReACC*).* Adversary $\mathcal{A}$ wins in the ReACC *experiment* $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{ReACC},\mathcal{M}}$ 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 5.

Algorithm 5 $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{ReACC},\mathcal{M}}(1^\kappa, \mathsf{N}, f)$

```
1: $[t, Out_{\mathcal{A}}, \iota, R] \leftarrow \textbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f)$
2: if $\mathcal{M}(R) = \perp$ then Return $\perp$
3: $\left\{Alg_i^{\hat{t}}, Inp_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
4: $\psi_r \leftarrow Out_{\mathcal{A}}$
5: Return:
        // $\psi_r$ was issued by an honest authority and $\psi_r$ is a valid revoked certificate
5.1:    $\mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \wedge \iota = \psi_r.\rho[\mathsf{ACC}].\iota \wedge \iota \in \mathsf{N} - \textbf{Corrupted}(R) \wedge$
        // However, a revoke instruction was not invoked on $\iota$
5.2:    $\nexists\ \hat{t}\ s.t.\ Alg_\iota^{\hat{t}} = \mathcal{P}.\mathsf{Revoke} \wedge Inp_\iota^{\hat{t}} = \psi \wedge Core(\psi) = Core(\psi_r)$
```

Revocation Accountability: Requirement 8 presented Algorithm 5. The $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{ReACC},\mathcal{M}}(1^\kappa, \mathsf{N})$ game executes adversary $\mathcal{A}$ using $\textbf{Exec}$, verifies that $\mathcal{A}$ followed the model $\mathcal{M}$, and extracts the execution details as described in $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{ACC},\mathcal{M}}$.

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

Requirement 9. *Revocation transparency (*$\Delta\mathsf{ReTRA}$*).* Adversary $\mathcal{A}$ wins in the $\Delta\mathsf{ReTRA}$ *experiment* $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\Delta\mathsf{ReTRA},\mathcal{M}}$ 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 6.

Algorithm 6 $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\Delta\mathsf{ReTRA},\mathcal{M}}(1^\kappa, \mathsf{N}, f)$

```
1: $[t, Out_{\mathcal{A}}, \iota, R] \leftarrow \textbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f)$
2: if $\mathcal{M}(R) = \perp$ then return $\perp$
3: $\left\{Alg_i^{\hat{t}}, Inp_i^{\hat{t}}, Out_i^{\hat{t}}, Clk_i^{\hat{t}}\right\}_{i\in\mathsf{N},\hat{t}\in\{1\ldots t\}} \leftarrow R$
4: $[\psi_r, x, \hat{t}] \leftarrow Out_{\mathcal{A}}$
5: Return:
        // $\psi_r$ is a valid revoked certificate with the revocation transparency attribute
5.1:    $\mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \wedge \mathcal{P}.\mathsf{WasValid}(\psi_r, \Delta\mathsf{ReTRA}) \wedge$
5.2:    $\iota, \psi_r.[\Delta\mathsf{ReTRA}].\iota \in \mathsf{N} - \textbf{Corrupted}(R) \wedge$
        // Honest authority $\iota$ is not aware of $\psi$ although it should
5.3:    $Clk_\iota^t \geq \psi_r.\rho[\Delta\mathsf{ReTRA}].clk + \Delta \wedge$
5.4:    $\left[Alg_\iota^t, Inp_\iota^t\right] = [\mathcal{P}.\mathsf{Query}, \psi.id] \wedge$
5.5:    $\nexists\psi \in Out_\iota^t\ s.t.\ Core(\psi_r) = Core(\psi) \wedge \mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{REV})$
```

Revocation Transparency: Requirement 9 presented Algorithm 6. The $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\Delta\mathsf{ReTRA},\mathcal{M}}(1^\kappa, \mathsf{N})$ game executes adversary $\mathcal{A}$ using $\textbf{Exec}$, verifies that $\mathcal{A}$ followed the model $\mathcal{M}$, and extracts the execution details as described in $\textbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{ACC},\mathcal{M}}$.

The adversary wins the game if four requirements are met. First, ψ_p must be a valid transparency pending certificate (line 6.1). Second, authority x and the transparency issuer of ψ_p

must be an honest authority (line 6.2). Notice that the game checks that the transparency issuer is honest and not the issuer of the certificate itself (i.e., accountability issuer), since even if the issuer of the certificate is corrupted, this does not prevent the transparency issuer from broadcasting the certificate to the rest of the authorities. The third requirement is that there is an honest authority x that is not aware of the transparency of ψ_p on time t (line 6.3). This requirement is validated by verifying that on time t authority x was instructed to perform $\mathcal{P}$.Query with $\psi_p.id$, but the output of this invocation did not contain a transparent certificate based on ψ_p (line 6.3). Since the $\mathcal{P}$.Query algorithm outputs all the certificate associated with $\psi_p.id$, this is a proof that an honest authority is not aware of ψ_p. The fourth and final requirement ensures that time t was sometime after the time that ψ_p was supposed to be transparent $\psi.\rho[\Delta\mathsf{TRA}].t$ (line 6.4). This requirement ensures that the evidence the adversary obtained of winning the game is indeed relevant, since until time $\psi.\rho[\Delta\mathsf{TRA}].t$ not all honest authorities are required to know ψ_p.

E. *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 PII 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. Let LiveC_{attr} be the *liveness-conditions predicate* over runs, s.t. if $\mathsf{LiveC}_{attr}(R) = \top$, then operations should complete successfully.

In PII, the only operation that upgrades certificates from 'pending' is the $\mathcal{P}$.Upgrade algorithm. Upgrading is applied to pending certificate ψ_p with respect to a specific attribute, and different attributes may have different liveness conditions. We denote the liveness condition of attribute $attr$ by LiveC_{attr}.

Requirement 10. *Liveness of pending certificate upgrade for attribute attr.* For every valid pending certificate ψ_p as input to $\mathcal{P}$.Upgrade on an honest authority ι, that the LiveC_{attr} algorithm decides satisfies the liveness criteria, the algorithm outputs a valid upgraded certificate ψ'. See Algorithm 7.

Algorithm 7 $\mathbf{LivenessExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{Upgrade},\mathsf{LiveC}_{attr},\mathcal{M}}(1^\kappa, \mathsf{N}, f)$

```
1: $[t, Out_{\mathcal{A}}, \iota, R] \leftarrow \mathbf{Exec}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f)$
2: **if** $\mathcal{M}(R) = \perp$ **then** Return $\perp$
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: $[\psi, attr, \alpha] \leftarrow Inp_t^{\iota}$
5: Return:
      // $\iota$ is an honest authority and $\mathcal{P}$.Upgrade was invoked on $\iota$ on round $t$
5.1:  $\iota \in \mathsf{N} - \mathbf{Corrupted}(R) \wedge Alg_\iota^t = \mathcal{P}.\mathsf{Upgrade} \wedge$
      // $\psi$ 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., $\iota$ did not upgrade $\psi$
5.3:  $\mathsf{LiveC}_{attr}(R) \wedge \mathcal{P}.\mathsf{WasValid}(Out_\iota^t, attr) \neq \top$
```

V. *United-π*: A Provably-Secure PII System

We now describe the *United-π* system, a provably-secure 'proof-of-concept' PII scheme. *United-π* is designed for simplicity rather than efficiency or deployability. In particular, *United-π* requires $|\mathsf{N}| > 3f$, while, at the same time, adopting an 'extreme' system model, as described next, although in this system model, it is easy to achieve an alternative implementation with $|\mathsf{N}| > 2f$ and with better efficiency.

United-π provably meets all PII requirements (see Appendix A for sketches of the proofs and the full version of this work [25] for rigorous proofs and definitions).

A. *System Model $\mathcal{M}$*

United-π assumes a synchronous model with reliable communication and synchronized clocks, i.e., all sent messages are received in one time unit. The adversary is a standard active adversary, that might control up to f compromised authorities out of the N authorities, as long as $|\mathsf{N}| > 3f$.

In *United-π*, the only upgrade operation that involves pending certificates is non-equivocation. Hence, the liveness criteria for non-equivocation upgrade $\mathsf{LiveC}_{\mathsf{NEQ}}(R)$ is that at least $3f + 1$ authorities out of N approved a pending certificate and they did so before Δ time has expired.

B. *Underlying cryptographic schemes*

United-π 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, however, for their security properties, see the citations.

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.

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 $\mathsf{Ver}(vk, m, \sigma) \to (\top/\bot)$, 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 ($\bot$).

A *robust (t,n)-threshold-signature* scheme $\mathcal{TS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Ver})$ [40], [41] consists of the following probabilistic algorithms:

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

C. United-π: Overview

The simplest form of a valid certificate in *United-π* is an accountable certificate; authorities directly issue certificates with the ACC attribute.

In *United-π*, upon request for transparency upgrade, the authority immediately outputs a transparent certificate without a need for a *pending* intermediate response. The authority immediately broadcasts the upgraded certificate; since *United-π* assume synchronous model with reliable communication, by the end of the following round, all authorities are aware of the transparent certificate, as required (with $\Delta = 1$). Note that a malicious authority could fail to perform this broadcast; the transparency requirements in current PKIs does not prevent this, e.g., consider a malicious logging-authority in CT that fails to disseminate a certificate. A stronger version of transparency could be defined (and supported), but we avoided this in this work to maintain focus and clarity.

Similarly to the 'issue' operation, upon a request to revoke a certificate, the authority simply adds to the certificate the REV attribute. To upgrade the certificate to ΔReTRA, the authority also needs to broadcast the certificate, with the ΔReTRA attribute, similarly to the transparency upgrade.

United-π achieves all of the aforementioned attributes using a standard signature scheme, where certificate ψ has attribute $attr$ if $\psi.\rho[attr].\sigma$ is a valid signature over ψ and $attr$ signed by $\psi.\rho[attr].\iota$. Every authority can provide certificates with these attributes; recall that relying parties are expected to ignore certificates or attributes where the signer is not authorized - the 'authorization' aspect, e.g., naming-constraints, is not part of the framework.

However, to achieve *non-equivocation*, a sufficient number of honest authorities must cooperate - which is why we use the term *united* (for *United-π*). In *United-π*, non-equivocation is achieved using *threshold signatures*. Namely, the NEQ attribute requires that $|\mathsf{N}| - f$ honest authorities would generate a verifiable group signature over ψ, including the NEQ attribute. Assuming that $|\mathsf{N}| > 3f$, this ensures that there is no other *unequivocal* certificate (i.e., a certificate with the NEQ attribute) for the same id. Efficiency and threshold could be easily improved, but *United-π* is only a proof of concept designed for simplicity, not efficiency.

The process of *non-equivocation upgrade* in *United-π* is as follows. When authority i is requested to upgrade certificate ψ with the NEQ attribute, the authority first returns, *and broadcasts* the *pending certificate*. Upon receiving a NEQ-pending certificate, an authority checks for any conflicts, i.e., any certificate known to it with the same identifier but conflicting public information, which also has NEQ- or NEQ-pending- attribute. If there is no conflict, then the authority uses its share of the threshold-signing key to send back a partial-signature for the certificate upgraded with the NEQ (non-pending) attribute. If a conflict is detected, then the conflicting-certificate is sent in response, which aborts the upgrade process, returning failure ($\bot$). Note that the model ensures this will be done within $\Delta = 2$ time units.

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 NEQ attribute.

Note, that systems that use *United-π* can decide for themselves what type of certificates they are willing to support. That is to say, that although non-equivocation is the strongest property suggested by *United-π*, systems can definitely accept and trust certificates which are 'only' accountable, transparent or pending; however, they 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.

D. United-π Algorithms

$\mathsf{Gen}(1^\kappa)$ (Algorithm 8). This algorithm receives as input security parameter 1^κ, and uses it to generate cryptographic encryption and signing keys.

Algorithm 8 $\mathsf{Gen}(1^\kappa)$

```
   // 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 $(PrivInfo = (dk, sk), PubInfo = (ek, vk))$
```

$\mathsf{GroupGen}(1^\kappa, \mathsf{N}, f, \{PubInfo_i\}_{i \in \mathsf{N}})$ (Algorithm 9). This algorithm receives as input security parameter 1^κ, the set N, maximal number of faulty parties f, 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. 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. The algorithm concludes by outputting the public information, which consists of the set of hints h_i, the group verification key vk, and the set of public keys ek_i, vk_i.

Algorithm 9 $\mathsf{GroupGen}(1^\kappa, \mathsf{N}, f, \{PubInfo_i\}_{i\in\mathsf{N}})$

```
    // Generate a group verification key $tvk$ and partial signing keys $tsk_i$
 1: $(tvk, \{tsk_i\}_{i=1}^{|\mathsf{N}|}) \leftarrow \mathcal{TS}.\mathsf{Gen}(1^\kappa, |\mathsf{N}|, |\mathsf{N}| - f)$
    // 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 $PubInfo$
```

$\mathsf{Issue}(id, pub, sd, ed)$ (Algorithm 10). Inputs are the certificate details: identity id, public information (incl. key) pub, and start, end dates sd, ed. Output is the certificate ψ.

Algorithm 10 $\mathsf{Issue}(id, pub, sd, ed)$

```
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 $\psi$
```

$\mathsf{Query}(id)$ (Algorithm 11). 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 11 $\mathsf{Query}(id)$

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

$\mathsf{Revoke}(\psi)$ (Algorithm 12) - used to revoke 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. In *United-π*, we require that certificates are revoked by their issuers; other implementations may require differently. Then, the algorithm checks if the certificate might have been already revoked. if this is the case, the revoked certificate is returned. 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.

$\mathsf{IsRevoked}(\psi)$ (Algorithm 13) - returns ψ_r if ψ was revoked, a non-revoked-certificate ψ' if not revoked, and $\perp$ if ψ is not a valid not expired certificate issued by this authority.

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 *United-π* that send messages.

Algorithm 12 $\mathsf{Revoke}(\psi)$

```
    // Verify that $\psi$ was issued by the authority and $\psi$ 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 $\psi$ was already revoked, return it
 2: if $\exists \psi_r \in S.certs[\psi.id]\ s.t.$
        $\mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \ \wedge\ Core(\psi) = Core(\psi_r)$
2.1:    Return $\psi_r$
 3: $\psi_r \leftarrow \psi$
    // Revoke $\psi$
 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 $\psi_r$ to the local state
 7: $S.certs[\psi_r.id] \mathrel{+}= \psi_r$
 8: Return $\psi_r$
```

Algorithm 13 $\mathsf{IsRevoked}(\psi)$

```
    // Verify that $\psi$ was issued by the authority and $\psi$ 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 $\psi$ was already revoked, return it
 2: if $\exists \psi_r \in S.certs[\psi.id]\ s.t.$
        $\mathcal{P}.\mathsf{WasValid}(\psi_r, \mathsf{REV}) \ \wedge\ Core(\psi) = Core(\psi_r)$
2.1:    Return $\psi_r$
 3: $\psi$' $\leftarrow \psi$
    // Add the non-revocation proof to $\psi$'
 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 $\psi$'
```

For simplicity, *United-π* sends most messages m to all entities, which we write as $Broadcast(m)$.

$\mathsf{Upgrade}(\psi, attr, [\alpha])$ (Algorithm 14): receives as input a certificate ψ, attribute $attr$ and optionally additional information α, and, if possible, upgrades ψ with $attr, \alpha$. If ψ was already previously upgraded with $attr$, the algorithm outputs the upgraded certificate.

United-π supports three types of upgrades: transparency ($\Delta\mathsf{TRA}$), revocations transparency ($\Delta\mathsf{ReTRA}$), and non-equivocation (NEQ). To upgrade ψ with $\Delta\mathsf{TRA}$ or $\Delta\mathsf{ReTRA}$ attribute, the algorithm simply signs ψ with the corresponding attribute and sends ψ to all authorities. However, upgrading a ψ with a NEQ attribute is more tricky, since we must ensure non-equivocation.

We designed *United-π* to ensure non-equivocation using a simple, albeit inefficient, method. Namely, the algorithm requires the approval of the rest of the (non-malicious) authorities in N. Therefore, the algorithm generates a pending certificate (as discussed in Section IV-A), and broadcasts ψ to the rest of the authorities (as a certificate that wants to be unequivocal). The rest of the non-equivocation upgrade logic is handled by the $\mathcal{P}.\mathsf{Incoming}$ algorithm. When $\mathcal{P}.\mathsf{Upgrade}$ receives as input a *pending* non-equivocation certificate ψ, the algorithm outputs the final upgraded certificate, if such exists. If the upgrade failed, the algorithm outputs $\perp$.

To check if a certificate is valid or has an attribute,

Algorithm 14 $\mathsf{Upgrade}(\psi, attr, [\alpha])$

```
        // Verify that $\psi$ is a valid, not expired certificate
1:    if $\mathcal{P}.\mathsf{WasValid}(\psi) \neq \top \ \vee \ \psi.ed < clk$ then return $\bot$
        // 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 $\exists \psi' \in S.certs \ s.t. \ Core(\psi) = Core(\psi') \ \wedge \ \mathcal{P}.\mathsf{WasValid}(\psi', attr)$
2.1:      Return $\psi'$
3:    $\psi' \leftarrow \psi$
4:    switch $attr$
            // Transparency upgrade
4.1:      case $\Delta\mathsf{TRA} \ \vee \ \Delta\mathsf{ReTRA}$
                // Add the transparency proof to $\psi'$
4.1.1:        $data \leftarrow (Core(\psi), attr, clk)$
4.1.2:        $\sigma = \mathcal{S}.\mathsf{Sign}(S.PrivInfo.sk, data)$
4.1.3:        $\psi'.\rho[attr] \leftarrow (\sigma, S.\iota, clk)$
4.1.4:        $Broadcast(\psi')$
            // Non-equivocation upgrade for a non-pending certificate
4.2:      case $(\mathsf{NEQ} \wedge \mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{NEQ}) = \bot)$
                // Add the $Pend\mathsf{NEQ}$ attr to $\psi'$
4.2.1:        $data \leftarrow (Core(\psi), Pend\mathsf{NEQ}, clk)$
4.2.2:        $\sigma = \mathcal{S}.\mathsf{Sign}(S.PrivInfo.sk, data)$
4.2.3:        $\psi'.\rho[\mathsf{NEQ}] \leftarrow (\sigma, S.\iota, clk)$
4.2.4:        $Broadcast(\psi')$
            // Non-equivocation upgrade for pending certificate
4.3:      case $\mathsf{NEQ} \wedge \mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{NEQ}) = Pending$
                // If upgrade time ($\Delta = 2$) did not pass, return the pending certificate
4.3.1:        if $clk < \psi.\rho[\mathsf{NEQ}].clk + \Delta$ then return $\psi$
                // Upgrade time passed already yet still 'pending': failure
4.3.2:        Return $\bot$
        // Add the certificates to the local set of certificates
5:    $S.certs[\psi.id] \mathrel{+}= \{\psi, \psi'\}$
6:    Return $\psi'$
```

Algorithm 15 $\mathsf{WasValid}(\psi, [attr, tms])$

```
1:    $\chi \leftarrow \psi.\rho[\mathsf{ACC}]$
        // Verify that $\chi.\iota$ (the issuer of $\psi$) is authorized to issue $\psi$ and that the accountability proof is cryptographically valid. These are the basic validity requirements in United-$\pi$
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 non-equivocation attribute
5:    if $attr = \mathsf{NEQ}$
            // Check if $\psi$ is pending
5.1:      if $\mathcal{S}.\mathsf{Ver}(S.PubInfo_{\eta.\iota}.vk, (Core(\psi), Pend\mathsf{NEQ}, \eta.clk), \eta.\sigma)$
5.1.1:        Return $Pending$
            // Certificate is not pending, check if the group proof is cryptographically valid
5.2:      Return $\mathcal{TS}.\mathsf{Ver}(S.PubInfo.vk_U, (Core(\psi), \mathsf{NEQ}), \eta.\sigma)$
        // For the rest of the attributes
6:    if $attr \in \{\Delta\mathsf{TRA}, \Delta\mathsf{ReTRA}, \mathsf{REV}, \mathsf{NR}\}$
            // Check that the proof is cryptographically valid
6.1:      Return $\mathcal{S}.\mathsf{Ver}(S.PubInfo_{\eta.\iota}.vk, (Core(\psi), attr, \eta.clk), \eta.\sigma)$
7:    Return $\bot$
```

the $\mathcal{P}.\mathsf{WasValid}(\psi, [attr, tms])$ algorithm is used (see Algorithm 15). The algorithm first checks if the inputted certificate is a valid certificate. In *United-π*, a valid certificate is a 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 and that the proof of accountability is cryptographically valid. If the inputted $attr$ argument has a value which is different than ACC, the algorithm now examines whether the certificate has the $attr$ attribute. Namely, the algorithm checks whether the certificate contains valid cryptographic proofs (according to the requested $attr$) and output $\top$ or $\bot$ accordingly.

The $\mathcal{P}.\mathsf{Time}$ algorithm is not required in *United-π*, since in *United-π* there are no time-based events.

The $\mathcal{P}.\mathsf{Incoming}$ algorithm handles all incoming messages (see Algorithm 16). In *United-π*, there are three possible incoming messages: (1) a certificate broadcast , (2) a non-equivocation rejection, and (3) a non-equivocation approval. When a certificate arrives, it is added to the local set of certificates. If the arriving certificate is pending non-equivocation, the algorithm checks against the local state whether there is a conflicting certificate. Such conflicting certificate can be either an existing unequivocal certificate or a pending non-equivocation certificate. If there is no conflict, the algorithm sends back a partial signature approving the non-equivocation upgrade.

When the algorithm receives a partial signature, it stores the partial signature locally; when enough partial signature have arrived, we combine them using the threshold signature scheme's $\mathcal{TS}.\mathsf{Combine}$ algorithm. If the $\mathcal{TS}.\mathsf{Combine}$ algorithm was successful, the algorithm stores the upgraded certificate.

VI. Analysis

In this section, we prove that *United-π* achieves PII's security and liveness properties. Due to space constraints, we include here only sketches of the proofs and the definitions of the model $\mathcal{M}$ and of the liveness conditions. For rigorous proofs and definitions, see Appendix A.

A. Accountability, Transparency, Revocation Accountability and Revocation Transparency

Claim 1. If *United-π* uses a secure signature scheme $\mathcal{S}$, then it achieves $\{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{ReACC}\}$ and $\Delta\mathsf{ReTRA}$.

Proof sketch. Assume, to the contrary, that there exists an adversary $\mathcal{A}$ that negates the claim that *United-π* achieve property $\xi \in \{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{ReACC}, \Delta\mathsf{ReTRA}\}$. Namely:

$$Pr[\mathbf{SafetyExp}^{\xi,\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

'Winning' in experiment $\mathbf{SafetyExp}^{\xi,\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}$ requires that $\mathcal{P}.\mathsf{WasValid}(\psi, \xi)$ returns true. From the implementation of $\mathcal{P}.\mathsf{WasValid}$ as described in Alg. 15, this requires that

$$\mathcal{S}.\mathsf{Ver}(S.PubInfo_{\eta.\iota}.vk, (Core(\psi), \xi, \eta.clk), \eta.\sigma) = \top$$

Algorithm 16 $\mathsf{Incoming}(M)$

```
       // Process each of the messages
1:  for each $m' \in M$
          // If $m'$ is a broadcasted certificate
1.1:      if $\psi \leftarrow m'$ s.t. $\mathcal{P}.\mathsf{WasValid}(\psi)$
              // Add the certificate to the local state
1.1.1:        $S.certs[\psi.id] \mathrel{+}= \psi$
              // If $\psi$ is pending non-equivocation
1.1.2:        if $\mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{NEQ}) = Pending$
                  // If a conflicting certificate is known, return it to abort
1.1.3:            if $\exists \psi' \in S.certs\ s.t.\ \psi.id = \psi'.id \wedge$
                  $Core(\psi) \neq Core(\psi') \ \wedge\ \mathcal{P}.\mathsf{WasValid}(\psi', \mathsf{NEQ}) \neq \bot$
                      // Prepare a rejection response with the conflicting certificates
1.1.4:                $res \leftarrow (\psi, \psi')$
                  // No conflicting certificate - approve the request for non-equivocation
1.1.5:            else
                      // Send partial signature
1.1.6:                $res \leftarrow (\psi, \sigma = \mathcal{TS}.\mathsf{Sign}(S.PrivInfo.tsk, (Core(\psi), \mathsf{NEQ})))$
1.1.7:            Send $res$ to authority $\psi.\rho[\mathsf{NEQ}].\iota$
1.1.8:        else if $(\psi, \sigma) \leftarrow m'\ s.t.\ \mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{NEQ}) = Pending$
                  $S.toUpgrade[\psi.id] \mathrel{+}= \sigma$
                  // Check to see if enough semi-proofs have arrived
1.1.9:            if $|S.toUpgrade[\psi.id]| < |CAs| - f$ then continue
                  // Enough semi-proofs have arrived - try to combine them
1.1.10:           $\psi' \leftarrow \psi$
1.1.11:           $\psi'.\rho[\mathsf{NEQ}].\sigma \leftarrow \mathcal{TS}.\mathsf{Combine}(S.toUpgrade[\psi.id])$
                  // Check if the upgrade was successful
1.1.12:           if $\mathcal{P}.\mathsf{WasValid}(\psi', \mathsf{NEQ})$ then $S.certs[\psi'.id] \mathrel{+}= \psi'$
```

for $\eta = \psi.\rho[\xi]$.

This allows us to use $\mathcal{A}$ to construct an adversary $\mathcal{A}$' that breaks the existential unforgeability of $\mathcal{S}$- a contradiction to the assumption the $\mathcal{S}$ is secure. Namely, $\mathcal{A}$' runs against the existential unforgeability experiment of $\mathcal{S}$; let v be the public key that $\mathcal{A}$' receives (as result of key generation $(s, v) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ by the experiment). $\mathcal{A}$' runs the $\mathbf{Exec}$ algorithm with *United*-π and $\mathcal{A}$, and with two modifications. First, $\mathcal{A}$' modifies the $\mathcal{P}.\mathsf{Gen}$ algorithm such that the public verification key of authority ι (the honest authority chosen by the adversary) is the public verification key v received from the unforgeability experiment; for other authorities and other functions of *United*-π, there is no change, and in particular, $\mathcal{A}$' 'knows' all the other private keys (except s, which should be used to sign for ι). Second, $\mathcal{A}$' replaces every call to $\mathcal{S}.\mathsf{Sign}$ needed to be performed by authority ι, with a matching oracle call to $\mathcal{S}_s$ in the unforgeability experiment. $\mathcal{A}$' takes the output of the adversary $\psi \leftarrow Out_{\mathcal{A}}$, and outputs (m, σ) such that:

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

Following the output of $\mathcal{A}$', we get

$$\mathcal{S}.\mathsf{Ver}(v, m, \sigma) \equiv \mathcal{S}.\mathsf{Ver}(S.PubInfo_{\eta.\iota}.vk, (Core(\psi), \xi, \eta.clk), \eta.\sigma)$$

Namely, $\mathcal{A}$' was given a public verification key v and was able to generate a message m and a valid signature σ over m, without knowing v's matching secret signing key s, thus contradicting the existential unforgeability of $\mathcal{S}$.

Therefore, when using a secure signature scheme, PII achieves property ξ. $\square$

B. Non-Equivocation

Non-equivocation in *United*-π depends on the security of encryption scheme $\mathcal{E}$ *and* of threshold signature scheme $\mathcal{TS}$. Hence, we split our argument into two steps. We first define an execution model where non-equivocation relies solely on a secure threshold signature $\mathcal{TS}$ (Def. 2) and show that *United*-π securely achieves non-equivocation under this model (see Claim 12). Then, we show that the security argument also holds under the original $\mathbf{Exec}$ execution model, see Claim 13.

Definition 2. Execution model $\mathbf{Exec}'_{\mathcal{A},\textit{United-}\pi}(1^\kappa, \mathsf{N}, f)$ is a variant of the $\mathbf{Exec}$ execution model described in Alg. 1 with *United*-π and an adversary $\mathcal{A}$, where instead of giving the 'real' output of the $\mathcal{P}.\mathsf{GroupGen}$ algorithm to the adversary (Alg.1 line 6), we give the adversary an encryption of zero (or any known string). Namely, in $\mathbf{Exec}'_{\mathcal{A},\textit{United-}\pi}(1^\kappa, \mathsf{N}, f)$, the execution model generates a special *PubInfo* for the adversary, by substituting line 2 of Alg. 9 with the following line:

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

Experiment $\mathbf{SafetyExp}'$ is identical to experiment $\mathbf{SafetyExp}$, except for the use of $\mathbf{Exec}$' instead of $\mathbf{Exec}$. We say that a PII scheme *achieves non-equivocation'* if for every PPT $\mathcal{A}$ holds:

$$Pr[\mathbf{SafetyExp'}^{\mathsf{NEQ},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa, \mathsf{N}, f) = 1] \in Negl(1^\kappa)$$

Claim 2. If *United*-π uses a secure threshold signature scheme $\mathcal{TS}$, then either *United*-π *achieves non-equivocation'*, as defined above, or $\mathcal{TS}$ is not a secure threshold signature scheme.

Proof sketch. Assume to the contrary that there exists an efficient (PPT) adversary $\mathcal{A}$ that negates the claim that *United*-π achieve non-equivocation', i.e.:

$$Pr[\mathbf{SafetyExp'}^{\mathsf{NEQ},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

We next show that this means that we can use $\mathcal{A}$ to construct a PPT adversary $\mathcal{A}$' that breaks the unforgeability of $\mathcal{TS}$.

Let $(v, \{s_i\}_{i \in \mathsf{N}}) \leftarrow \mathcal{TS}.\mathsf{Gen}(1^\kappa, |\mathsf{N}|, f)$. $\mathcal{A}$' executes $\mathbf{Exec}'_{\mathcal{A},\textit{United-}\pi}$ algorithm, using $\mathcal{A}$, *United*-π as 'black box', with two modifications. First, $\mathcal{A}$' modifies the $\mathsf{GroupGen}$ algorithm by replacing the public group verification key vk with the generated public verification key v. Second, $\mathcal{A}$' replaces every call to $\mathcal{TS}.\mathsf{Sign}$ in the $\mathsf{Upgrade}$ and $\mathsf{Incoming}$ algorithms with a matching oracle operation to $\mathcal{TS}$. $\mathcal{A}$' takes the output of the adversary $\psi \leftarrow Out_{\mathcal{A}}$, and outputs $(m, \sigma)\ s.t.\ m = (\psi, \mathsf{NEQ})$ and $\sigma = \psi.\rho[\mathsf{NEQ}].\sigma$.

To 'win' in the $\mathbf{SafetyExp}'$ experiment, the conditions in lines 6.1 to 6.3 of the experiment must hold. In particular, we should have two certificates, ψ and ψ', with the same identifier $\psi.id = \psi'.id$ but different public information, i.e., $\psi.pub \neq \psi'.pub$, which are both 'valid', i.e., $\mathcal{P}.\mathsf{WasValid}(\psi, \mathsf{NEQ}) \wedge \mathcal{P}.\mathsf{WasValid}(\psi', \mathsf{NEQ})$. Hence we have:

$$\mathcal{TS}.\mathsf{Ver}(v, m, \sigma) \equiv \mathcal{TS}.\mathsf{Ver}(S.vk, (\psi, \mathsf{NEQ}), \psi.\rho[\mathsf{NEQ}].\sigma)$$

$\mathcal{TS}.\mathsf{Ver}(v, m, \sigma) \equiv \mathcal{TS}.\mathsf{Ver}(S.vk, (\psi\text{'}, \mathsf{NEQ}), \psi\text{'}.\rho[\mathsf{NEQ}].\sigma)$

We will show how $\mathcal{A}$' 'wins' the forgery experiment of $\mathcal{TS}$. *United-π* uses parameters $|\mathsf{N}|$ for the number of shares and $|\mathsf{N}|-f$ for the threshold, i.e., exactly $|\mathsf{N}|-f$ shares are required to construct a valid group signature; recall that $|\mathsf{N}| > 3f$. The only place in *United-π* where authorities use their share of the group signing key, is in line 1.1.4 of the Incoming algortihm 16, where an authority generates a share for an NEQ certificate ψ. However, this line is executed only if the check in 1.1.3 passes, i.e., there is no conflicting certificate ψ' in the $S.certs$ repository, with same public information, different identifier - and a valid or pending NEQ attribute. Namely, each honest authority would only execute line 1.1.4, i.e., generate their partial group-signature, for *either ψ or ψ'*, not for both. Let n_ψ ($n_{\psi\text{'}}$) denote the number of honest authorities partially-signing ψ (resp., ψ'). Then surely $n_\psi + n_{\psi\text{'}} \leq |\mathsf{N}| - f$. Assume, WLOG, that $n_\psi \geq |\mathsf{N}| - 2f$, which will mean that there are sufficiently signed signature-shares to combine into a validly-signed version of ψ with the NEQ attribute. It follows that $n_{\psi\text{'}} \leq |CAs| - f - (|CAs| - 2f) = f$; hence the total number of shares of signatures for ψ' is at most $2f$ ($n_{\psi\text{'}}$, plus f malicious authorities). But we used threshold of $|\mathsf{N}| - f > 3f - f = 2f$; hence, there are not enough partial signatures to combine into a valid NEQ certificate-upgrade for ψ'.

Since we assumed, to the contrary, that $\mathcal{A}$ is able to get both of these NEQ certificates, it follows that we can use it to generate a threshold signature - without the threshold number ($|\mathsf{N}| - f$) of partial-signatures. Therefore, when using a secure threshold signature scheme $\mathcal{TS}$, *United-π* achieves non-equivocation under the $\mathbf{Exec}\text{'}_{\mathcal{A},\textit{United-}\pi}$ model. ☐

Claim 3. If *United-π* uses a secure encryption scheme $\mathcal{E}$ and secure threshold-signature scheme $\mathcal{TS}$, then it achieves non-equivocation.

Proof sketch. The difference between the **Exec** and the **Exec**' models is that the adversary has an advantage in **Exec**, because it also receives the encrypted secret information generated by GroupGen using a secure encryption scheme $\mathcal{E}$. Claim 12 shows that *United-π* is secure under the **Exec**' model, since $\mathcal{TS}$ is secure; we now show that the security under the **Exec** model also holds.

Assume to the contrary that although $\mathcal{E}$ is secure; there exists an adversary $\mathcal{A}$ that negates the claim that *United-π* achieve non-equivocation under the **Exec** model, namely:

$$Pr[\mathbf{SafetyExp}^{\mathsf{NEQ},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

Since the only difference between **Exec** and **Exec**' is the use of $\mathcal{E}$ to encrypt the individual secret information, it means that $\mathcal{A}$ uses this advantage to win the experiment. Moreover, this means that we can use $\mathcal{A}$ to construct an adversary $\mathcal{A}$' that breaks the *indistinguishability experiment* of $\mathcal{E}$.

Recall the indistinguishability experiment of $\mathcal{E}$ with an adversary $\mathcal{A}$. The experiment randomly chooses $b \in \{0, 1\}$. If $b = 0$, we execute $\mathcal{A}$ under the $\mathbf{Exec}\text{'}_{\mathcal{A},\textit{United-}\pi}$ model, and if $b = 1$, we execute $\mathcal{A}$ under the $\mathbf{Exec}_{\mathcal{A},\textit{United-}\pi}$ model. The adversary wins the game if it outputs b' such that $b = b'$.

Consider an adversary $\mathcal{A}$' that simulates $\mathcal{A}$, and outputs $b' = 1$ if $\mathcal{A}$ wins the $\mathbf{SafetyExp}^{\mathsf{NEQ},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa, \mathsf{N}, f)$ experiment (since we conclude it is an execution under the **Exec** model, where $\mathcal{A}$ has an advantage), and outputs $b' = 0$ otherwise (since it is probably an execution under the **Exec**' model). Consequently, if such $\mathcal{A}$ exists then we are able to construct $\mathcal{A}$' that wins the aforementioned indistinguishability experiment with a non-negligible probability, thus contradicting the indistinguishability of $\mathcal{E}$.

Therefore, *United-π* achieves non-equivocation (also under the **Exec** model). ☐

C. Liveness

Claim 4. *United-π* achieves liveness of pending certificate upgrade for attribute $attr$.

Proof sketch. Assume to the contrary that *United-π* does not achieves liveness of pending certificate upgrade for attribute $attr$. In *United-π*, the only case when a pending certificate is issued is when upgrading a certificate with the non-equivocation attribute. Therefore, there exists an adversary $\mathcal{A}$ where

$$Pr[\mathbf{LivenessExp}^{\mathsf{Upgrade},\mathsf{LiveC}_{\mathsf{NEQ}},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

Thus, following experiment $\mathbf{LivenessExp}^{\mathsf{Upgrade},\mathsf{LiveC}_{\mathsf{NEQ}},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}$, it means that although the liveness criteria $\mathsf{LiveC}_{\mathsf{NEQ}}(R)$ was met, an upgraded certificate was not issued by an honest authority ι. However, if $\mathsf{LiveC}_{\mathsf{NEQ}}(R) = \top$, this means that at least $3f + 1$ authorities out of N sent approvals to ι and they did so before Δ time has expired. Since there are only f malicious authorities, ι had enough partial proofs to combine and create the upgraded certificate before Δ expires and would have generated a valid combined proof of non-equivocation. Therefore, since ι is an honest authority, ι would have outputted an upgraded certificate on any time $Clk^t_\iota \geq \psi.\rho[attr].clk + \Delta$, thus contradicting the assumption that such $\mathcal{A}$ exists.

Therefore, when using a secure signature scheme, *United-π* achieves liveness of pending certificate upgrade for attribute $attr$.

☐

VII. Conclusions and Future Work

Public key infrastructure (PKI) is critical to the security of many systems. The ability to define and prove security requirements for PKIs is important, especially that over the last several years, there have been significant progress in design of PKIs, and their security properties has become more complex and non-trivial to understand, formalize, analyze, and prove.

In this work we present a framework which allows to define precisely security requirements for PKI schemes - and we show that these definitions are usable, by presenting a *United-π*, a simple, 'proof of concept' PKI system, satisfying all requirements presented. We hope that our work can provide good foundation and starting point for such collaborative effort by the cryptographic and security communities, to 'debug' and extend these definitions, improve upon them and extend them - as was the case for other basic cryptographic schemes, e.g., encryption. Some of these extensions are already mentioned in this work, e.g., a strong-transparency requirement, that will *ensure* publication of a certificate, beyond ensuring *accountability* for failure of publication (as in Certificate Transparency and other existing schemes). Other directions for further research include (1) analysis of existing (and new) PKI schemes with respect to our framework, and in particular, designing optimized variant of *United*-π, which was presented in a simplified, sub-optimal design and (2) adopting the framework to define and study security properties of other complex systems, (3) extending the framework to support secure compositions, e.g. following UC [42], and specifically [43] (which present a basic UC model for certification).

References

[1] O. Goldreich, *Foundations of cryptography*. Cambridge University Press, 2009.

[2] A. Freier, P. Karlton, and P. Kocher, "The secure sockets layer (SSL) protocol version 3.0," IETF, Tech. Rep., 2011.

[3] J. Prins, "Diginotar certificate authority breach 'operation black tulip," 2011.

[4] Comodo™, "Incident Report," Published online, http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html, March 2011.

[5] J. Dyer, "China accused of doling out counterfeit digital certificates in 'serious' web security breach," VICE News, April 2015.

[6] C. Evans, C. Palmer, and R. Sleevi, "Rfc 7469: Public key pinning extension for http," *IETF. URl: https://tools. ietf. org/html/rfc7469*, 2015.

[7] I. Dacosta, M. Ahamad, and P. Traynor, "Trust no one else: Detecting mitm attacks against ssl/tls without third-parties," in *European Symposium on Research in Computer Security*. Springer, 2012, pp. 199–216.

[8] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," IETF, Tech. Rep., 2013.

[9] B. Laurie and E. Kasper, "Revocation transparency," *Google Research, September*, 2012.

[10] M. D. Ryan, "Enhanced certificate transparency and end-to-end encrypted mail." in *NDSS*, 2014.

[11] P. Eckersley, "Sovereign key cryptography for internet domains," https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=HEAD, 2012.

[12] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "Coniks: Bringing key transparency to end users," in *USENIX Security Symposium*, 2015, pp. 383–398.

[13] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor, "Accountable key infrastructure (aki): a proposal for a public-key validation infrastructure," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 679–690.

[14] P. Szalachowski, S. Matsumoto, and A. Perrig, "Policert: Secure and flexible tls certificate management," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 406–417.

[15] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "ARPKI: Attack resilient public-key infrastructure," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 382–393.

[16] J. Yu, V. Cheval, and M. Ryan, "DTKI: A new formalized PKI with verifiable trusted parties," *The Computer Journal*, vol. 59, no. 11, pp. 1695–1713, 2016.

[17] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities" honest or bust" with decentralized witness cosigning," in *Security and Privacy (SP), 2016 IEEE Symposium on*. Ieee, 2016, pp. 526–545.

[18] S. Matsumoto and R. M. Reischuk, "IKP: Turning a PKI around with decentralized automated incentives," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 410–426.

[19] C. Fromknecht, D. Velicanu, and S. Yakoubov, "A decentralized public key infrastructure with identity retention." *IACR Cryptology ePrint Archive*, vol. 2014, p. 803, 2014.

[20] L. Axon and M. Goldsmith, "Pb-pki: A privacy-aware blockchain-based pki," in *SECRYPT*, 2017.

[21] A. Tomescu and S. Devadas, "Catena: Efficient non-equivocation via bitcoin," in *2017 38th IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 393–409.

[22] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "Certledger: A new pki model with certificate transparency based on blockchain," *arXiv preprint arXiv:1806.03914*, 2018.

[23] International Telecommunication Union, "Recommendation ITU-T X.509, OSI – the directory: Public-key and attribute certificate frameworks," 2016. [Online]. Available: https://www.itu.int/rec/T-REC-X.509-201610-I/en

[24] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5280.txt

[25] "Provable Security for PKI Schemes - Full Version," available from authors.

[26] E. Rescorla, *SSL and TLS Designing and Building Secure Systems*. Addison-Wesley, 2006.

[27] S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, and L. Repka, "S/MIME Version 2 Message Specification," RFC 2311 (Historic), RFC Editor, Fremont, CA, USA, pp. 1–37, Mar. 1998. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2311.txt

[28] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile," Tech. Rep., 2008.

[29] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and D. C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," RFC 6960, Jun. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6960.txt

[30] S. B. Roosa and S. Schultze, "The" certificate authority" trust model for ssl: a defective foundation for encrypted web traffic and a legal quagmire," *Intellectual property & technology law journal*, vol. 22, no. 11, p. 3, 2010.

[31] H. Asghari, M. Van Eeten, A. Arnbak, and N. A. van Eijk, "Security economics in the https value chain," in *Twelfth Workshop on the Economics of Information Security (WEIS 2013), Washington, DC*, 2013.

[32] J. Hruska, "Apple, microsoft buck trend, refuse to block unauthorized chinese root certificates," ExtremeTech, April 2015.

[33] R. Kotcher, Y. Pei, P. Jumde, and C. Jackson, "Cross-origin pixel stealing: timing attacks using css filters," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 1055–1062.

[34] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving ssh-style host authentication with multi-path probing." in *USENIX Annual Technical Conference*, vol. 8, 2008, pp. 321–334.

[35] E. F. F. (EFF). The eff ssl observatory. [Online]. Available: https://www.eff.org/observatory

[36] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the https protocol," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 78–81, 2009.

[37] B. Dowling, F. Günther, U. Herath, and D. Stebila, "Secure logging schemes and certificate transparency," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 140–158.

[38] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 696–701.

[39] "Gossiping in CT draft-ietf-trans-gossip-04," https://tools.ietf.org/html/draft-ietf-trans-gossip-04.

[40] P.-A. Fouque and J. Stern, "Fully distributed threshold rsa under standard assumptions," *Advances in CryptologyASIACRYPT 2001*, pp. 310–330, 2001.

[41] J. L. 0002, T. H. Yuen, and K. Kim, "Practical threshold signatures without random oracles," in *Provable Security, First International Conference, ProvSec 2007, Wollongong, Australia, November 1-2, 2007, Proceedings*, ser. Lecture Notes in Computer Science, W. Susilo, J. K. Liu, and Y. M. 0001, Eds., vol. 4784. Springer, 2007, pp. 198–207.

[42] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Foundations of Computer Science, 2001.*

Proceedings. 42nd IEEE Symposium on. IEEE, 2001, pp. 136–145, online at https://eprint.iacr.org/2000/067.pdf, last updated Dec. 2018.

[43] R. Canetti, D. Shahaf, and M. Vald, "Universally composable authentication and key-exchange with global pki," Cryptology ePrint Archive, Report 2014/432, 2014, https://eprint.iacr.org/2014/432.

Appendix A
Analysis of *United-π*

In this section, we provide reduction-based proofs showing that *United-π* achieves its safety and liveness properties. We first show that *United-π* achieves accountability, transparency, revocation accountability and revocation transparency by reduction to the existential unforgeability of a secure signature scheme. We then show that *United-π* also achieves non-equivocation by reduction to the existential unforgeability of a secure threshold signature scheme. We conclude by showing that *United-π* achieves the liveness properties of PII.

A. Preliminaries

We briefly recall the definitions of the security games for secure encryption and signature schemes used by *United-π* (see Section V-B). We start with the definition of CPA-indistinguishability of a secure encryption scheme $\mathcal{E}$.

Definition 3. An encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is CPA-indistinguishable (CPA-IND), if for every PPT adversary $\mathcal{A}$:

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

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

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) $\mathcal{A}$ receives vk and $c = \mathcal{E}.\mathsf{Enc}(sk, m_b)$ and outputs a value b' $\in \{0, 1\}$.

$\mathcal{A}$ wins the game if $b = b$'.

We now recall the definition of existential unforgeability of a secure signature scheme $\mathcal{S}$.

Definition 4. A signature scheme $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is existentially unforgeable if for every PPT adversary $\mathcal{A}$:

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

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

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.

$\mathcal{A}$ wins the game if $\mathcal{S}.\mathsf{Ver}(vk, m, \sigma) = \top$ and $\mathcal{A}$ did not use the oracle access on m.

We now recall the definition of existential unforgeability of a secure (t,n)-threshold-signature scheme.

Definition 5. 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 $\mathcal{A}$:

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

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

1) $(tvk, \{tsk\}_{i\in\mathsf{N}}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa, \mathsf{N}, t)$
2) Adversary $\mathcal{A}$ receives vk and has an oracle access to $\mathcal{TS}$ to sign any message it desires.
3) $\mathcal{A}$ outputs message m and signature σ.

$\mathcal{A}$ wins the game if $\mathcal{TS}.\mathsf{Ver}(tvk, m, \sigma) = \top$ and $\mathcal{A}$ did not use the oracle access on m.

We conclude with the definition of robustness of a (t,n)-threshold-signature scheme.

Definition 6. A (t,n)-threshold-signature scheme $\mathcal{TS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Ver})$ is robust, if there is no PPT adversary $\mathcal{A}$ that controls up to t players and has an oracle access to $\mathcal{TS}$ that can prevent honest parties from combining t' partial signatures from t' players into a single threshold signature, where $t' \geq 2t + 1$.

B. Proofs of Accountability, Transparency, Revocation Accountability and Revocation Transparency

Proof methodology. We argue that *United-π* achieves the following four security properties: accountability, transparency, revocation accountability and revocation transparency. We do so using a reduction to the existential unforgeability of the secure signature scheme $\mathcal{S}$ that each of these algorithms use. Our proof methodology is as follows:

1) Given a security property $\xi \in \{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{ReACC}, \Delta\mathsf{ReTRA}\}$, we assume to the contrary that *United-π does not* achieve ξ.
2) Hence, there must exist a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ that wins the security experiment defined for ξ with a non-negligible probability.
3) Then, we show how to use $\mathcal{A}$ to build an adversary $\mathcal{A}$' that breaks the existential unforgeability of the secure signing scheme $\mathcal{S}$, thus contradicting that such an adversary $\mathcal{A}$ exists.
   a) We first define an execution model **Exec**' (see Def. 7), which is a variant of the **Exec** execution model described in Alg. 1.
   b) Then, we construct an adversary $\mathsf{AdvEU}^{\mathcal{S}}_{\mathcal{A}}$ (see Alg. 17) that executes the given adversary $\mathcal{A}$ in the *United-π* under **Exec**' and outputs a message m and signature σ over m.
   c) Finally, we argue that if adversary $\mathcal{A}$ prevents *United-π* from achieving property ξ, then adversary $\mathsf{AdvEU}^{\mathcal{S}}_{\mathcal{A}}$ breaks the existential unforgeability of $\mathcal{S}$ (see Claim 5).

We start by defining the **Exec**' execution model.

Definition 7. Let $\mathcal{S}$ be a secure signature scheme and let $(sk', vk') \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$. Execution model **Exec**' is a variant of the **Exec** execution model, with *United-π* as PII implementation and an adversary $\mathcal{A}$ with the following changes:

1) The Gen algorithm has the following *additional* code:

```
if $S_\iota.chosen = \top$ then
    Return $(PrivInfo = (dk, nil), PubInfo = (ek, vk'))$
```

between line 1 and line 2, where vk' is the public verification key generated by $\mathcal{S}$.Gen.

2) In Issue, Upgrade, Revoke, IsRevoked algorithms, *replace* the following:

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

with the following code:

```
if $S_\iota.chosen = \top$ then
    Generate $\sigma$ by signing $data$ using the oracle access to $\mathcal{S}$.
else
    $\sigma = \mathcal{S}.\mathsf{Sign}(S.PrivInfo.sk, data)$
```

We now describe algorithm $\mathsf{AdvEU}_{\mathcal{A}}^{\mathcal{S}}$ (see Alg. 17). $\mathsf{AdvEU}_{\mathcal{A}}^{\mathcal{S}}$ takes as input a security parameter 1^κ, an attribute ξ, a public verification key vk, a set of authorities N and the number f of malicious authorities in N. The algorithm is using a secure signature scheme $\mathcal{S}$ and executes the adversary $\mathcal{A}$ in *United-π* under **Exec**' in order to generate a message m and a signature σ, such that σ is a valid signature over m which can be verified by vk, i.e., $\mathcal{S}.\mathsf{Ver}(vk, m, \sigma) = \top$. Note, that $\mathsf{AdvEU}_{\mathcal{A}}^{\mathcal{S}}$ has only access to vk and has no access to vk's matching signing key sk.

Algorithm 17 $\mathsf{AdvEU}_{\mathcal{A}}^{\mathcal{S}}(1^\kappa, \xi, vk, \mathsf{N}, f)$

```
1: $[t, Out_{\mathcal{A}}, \iota, R] \leftarrow \mathbf{Exec'}_{\mathcal{A},United\text{-}\pi}(1^\kappa, \mathsf{N}, f)$
2: if $\mathcal{M}(R) = \perp$ then Return $\perp$
3: $\psi \leftarrow Out_{\mathcal{A}}$
4: Output $(m, \sigma)$ $s.t.$ $m = (Core(\psi), \xi, \psi.\rho[\xi].clk)$ and $\sigma = \psi.\rho[\xi].\sigma$
```

We now argue that if there exists an adversary $\mathcal{A}$ that can break the security of *United-π*, we can use algorithm $\mathsf{AdvEU}_{\mathcal{A}}^{\mathcal{S}}$ described in Alg. 17 with adversary $\mathcal{A}$ to break the existential unforgeability of $\mathcal{S}$.

Claim 5. For every PPT adversary $\mathcal{A}$ that achieves

$$Pr[\mathbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\xi,\mathcal{M}}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

where $\xi \in \{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{ReACC}, \Delta\mathsf{ReTRA}\}$, there exists a PPT adversary $\mathcal{A}$' that achieves

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

Proof. To prove this claim, we demonstrate that if such adversary $\mathcal{A}$ exists, then there exists an adversary $\mathcal{A}$' that given a public verification key vk is able to generate a message m and a signature σ such that

$$\mathcal{S}.\mathsf{Ver}(vk, m, \sigma) = \top \tag{1}$$

without knowing the matching secret signing key sk, thus contradicting the existential unforgeability of $\mathcal{S}$.

For each attribute $\xi \in \{\mathsf{ACC}, \Delta\mathsf{TRA}, \mathsf{ReACC}, \Delta\mathsf{ReTRA}\}$, the safety experiment $\mathbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\xi,\mathcal{M}}(1^\kappa, \mathsf{N}, f)$ contains the algorithm call $\mathcal{P}.\mathsf{WasValid}(\psi, \xi)$. Following the implementation described in Alg. 15, the algorithm checks if

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

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

$$Pr[\mathbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\xi,\mathcal{M}}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

it must hold that

$$\mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota}, (Core(\psi), \xi, \eta.clk), \eta.\sigma) = \top \tag{2}$$

However, this means that we can construct an adversary $\mathcal{A}$' using $\mathcal{A}$ and AdvEU. Namely, $\mathcal{A}$' executes AdvEU with $\mathcal{A}$ and outputs AdvEU's outputs. First, since $\mathcal{A}$ is polynomial then $\mathcal{A}$' is also polynomial. Second, since AdvEU simulates $\mathcal{A}$ with the public verification key vk, then $vk = S.PubInfo.pk_{\eta.\iota}$, and following Alg. 17 the output of AdvEU is message $m = (Core(\psi), \xi, \psi.\rho[\xi].clk)$ and signature $\sigma = \psi.\rho[\xi].\sigma$, we get

$$\begin{aligned} \mathcal{S}.\mathsf{Ver}(S.PubInfo.pk_{\eta.\iota}, (Core(\psi), \xi, \eta.clk), \eta.\sigma) &\equiv \\ \mathcal{S}.\mathsf{Ver}(vk, m, \sigma) &\equiv \top \end{aligned} \tag{3}$$

and therefore, following Eq. 1, 2 and 3, we constructed an adversary $\mathcal{A}$' that satisfies:

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

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

Proofs. We now apply Claim 5 to argue that *United-π* achieves accountability, transparency, revocation accountability and revocation transparency. Since *United-π* uses the signature scheme $\mathcal{S}$ slightly differently to achieve these properties, we provide a separate claim for each property. In each claim we first explain how *United-π* uses the signature scheme $\mathcal{S}$ to achieve the specific property and then employ the reduction described in Claim 5 to argue that *United-π* indeed achieves the specific property.

Claim 6. If *United-π* uses a signature scheme $\mathcal{S}$, then either *United-π* achieves accountability, or $\mathcal{S}$ is not a secure signature scheme.

Proof. In *United-π*, the only way to generate a valid accountable certificate ψ, is by invoking the $\mathcal{P}$.Issue algorithm on authority ι which is authorized to issue ψ. According to the implementation described in Alg. 10, the algorithm $\mathcal{P}$.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 *United-π does not* achieve accountability. Namely:

$$Pr[\mathbf{SafetyExp}_{\mathcal{A},\mathcal{P}}^{\mathsf{ACC},\mathcal{M}}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

However, following Claim 5, if such adversary $\mathcal{A}$ exists, we can use $\mathcal{A}$ to build $\mathcal{A}$' that breaks the existential unforgeability of the secure signature scheme $\mathcal{S}$.

Therefore, PII achieves accountability. □

Claim 7. If *United-π* uses a signature scheme $\mathcal{S}$, then either *United-π* achieves transparency, or $\mathcal{S}$ is not a secure signature scheme.

Proof. In *United-π*, the only way to generate a valid transparent certificate ψ, is by invoking the $\mathcal{P}$.Upgrade algorithm on authority ι. According to the implementation described in Alg. 14, the $\mathcal{P}$.Upgrade algorithm uses the secure $\mathcal{S}$.Sign algorithm to generate the proof that ψ is a transparent certificate.

Assume to the contrary that *United-π does not* achieve transparency. Namely:

$$Pr[\mathbf{Exp}^{\Delta\mathsf{TRA}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

However, following Claim 5, if such adversary $\mathcal{A}$ exists, we can use $\mathcal{A}$ to build $\mathcal{A}$' that breaks the existential unforgeability of the secure signature scheme $\mathcal{S}$.

Therefore, PII achieves transparency. □

Claim 8. If *United-π* uses a signature scheme $\mathcal{S}$, then either *United-π* achieves revocation accountability, or $\mathcal{S}$ is not a secure signature scheme.

Proof. In *United-π*, the only way to revoke a certificate ψ, is by invoking the $\mathcal{P}$.Revoke algorithm on $\psi.\rho[\mathsf{ACC}].\iota$ (the issuer of ψ). According to the implementation described in Alg. 12, the $\mathcal{P}$.Revoke algorithm uses the secure $\mathcal{S}$.Sign algorithm to generate the proof that ψ was revoked by ι.

Assume to the contrary that *United-π does not* achieve revocation accountability. Namely:

$$Pr[\mathbf{Exp}^{\mathsf{ReACC}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

However, following Claim 5, if such adversary $\mathcal{A}$ exists, we can use $\mathcal{A}$ to build $\mathcal{A}$' that breaks the existential unforgeability of the secure signature scheme $\mathcal{S}$.

Therefore, PII achieves revocation accountability. □

Claim 9. If *United-π* uses a signature scheme $\mathcal{S}$, then either *United-π* achieves revocation transparency, or $\mathcal{S}$ is not a secure signature scheme.

Proof. In *United-π*, the only way to achieve revocation transparency is by invoking the $\mathcal{P}$.Upgrade algorithm on authority ι. According to the implementation described in Alg. 14, the $\mathcal{P}$.Upgrade algorithm uses the secure $\mathcal{S}$.Sign algorithm to generate the proof that ψ is transparently revoked by ι.

Assume to the contrary that *United-π does not* achieve revocation transparency. Namely:

$$Pr[\mathbf{Exp}^{\Delta\mathsf{ReTRA}}_{\mathcal{A},\mathcal{P}}(1^\kappa, \mathsf{N}, f) = 1] \notin Negl(1^\kappa)$$

However, following Claim 5, if such adversary $\mathcal{A}$ exists, we can use $\mathcal{A}$ to build $\mathcal{A}$' that breaks the existential unforgeability of the secure signature scheme $\mathcal{S}$.

Therefore, PII achieves revocation transparency. □

C. Proof of Non-Equivocation

Proving that *United-π* achieves non-equivocation is different from proving the other properties, because *United-π* implements non-equivocation using both a secure encryption scheme $\mathcal{E}$ and a secure and 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 *United-π* achieves non-equivocation, we use the following methodology:

1) We define an execution model **Exec**' where non-equivocation relies solely on a secure (t,n)-threshold-signature scheme $\mathcal{TS}$ (Def. 8) and define a matching variant of the non-equivocation property called non-equivocation', which is identical to non-equivocation but under the **Exec**' model (Def. 10).
2) Then, we show that *United-π* achieves non-equivocation' under the **Exec**' model, see Claims 10,11,12.
3) Finally, we show that the security argument also holds for (original) non-equivocation property under the original **Exec** execution model, see Claim 13.

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 in non-equivocation, the aforementioned proof methodology separates the two by defining the **Exec**' execution model, where encryption is not used. The fact that **Exec**' only slightly varies from **Exec**, allows us to prove that non-equivocation can be achieved in *United-π* 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 encryption to **Exec**', thus ending up the with the original **Exec** model, the security argument that *United-π* achieves non-equivocation still holds, as long as the encryption scheme is secure.

We start by defining the **Exec**' execution model.

Definition 8. Let $\mathcal{E}$ be a secure signature scheme and let $(ek', dk') \leftarrow \mathcal{E}.\mathsf{Enc}(1^\kappa)$. The execution model **Exec**' is a variant of the **Exec** execution model described in Alg. 1, with *United-π* as PII implementation and an adversary $\mathcal{A}$ with the following changes:

1) The Gen algorithm has the following *additional* code:

```
if $S_\iota.chosen = \top$ then
    Return $(PrivInfo = (nil, sk), PubInfo = (ek', vk))$
```

between line 2 and line 3, where ek' is the public encryption key generated by $\mathcal{E}$.Enc.

2) In GroupGen algorithm, *replace* the following line:

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

with the following:

$$H = \{h_i \leftarrow \mathcal{E}.\mathsf{Enc}(PubInfo_i.ek, tsk_i)\}_{i\in\mathsf{N}-\iota}$$
$$H = H \ \cup \ \{h_\iota \leftarrow \mathcal{E}.\mathsf{Enc}(PubInfo_\iota.ek, \text{'0'})\}$$

Note the two modifications that happen in **Exec**' as opposed to **Exec**. First, authority ι chosen by the adversary, does not generates random encryption/decryption keys (like the rest of the authorities), but rather use a predetermined encryption key ek'. Second, ι receives a 'useless' string ('0') instead of

ι's share of the signing key tsk_ι. These two modifications essentially eliminate the part that the encryption scheme $\mathcal{E}$ plays in non-equivocation, and allows the argument made in Claim 13. We now define matching definitions for safety experiments and safety properties under the **Exec**' model.

Definition 9. Safety experiment **SafetyExp**' is identical to safety experiment **SafetyExp** defined in Sec. IV-D, except for the use of the **Exec**' execution model instead of **Exec**.

Definition 10. *United-π achieves non-equivocation'* if for every PPT adversary $\mathcal{A}$ holds:

$$Pr[\mathbf{SafetyExp'}^{\mathsf{NEQ},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa,\mathsf{N},f)=1]\in Negl(1^\kappa)$$

We now start the second phase of our proof process. We begin by showing that *United-π* 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 NEQ attribute, it will not partially sign any other certificate ψ' with the same identifier ($\psi.id = \psi$'$.id$) that its validity period overlaps ψ's validity period, since these two certificates are in conflict.

Claim 10. Let $\mathcal{TS}$ be a (t,n)-threshold-signature scheme. If *United-π* uses $\mathcal{TS}$ with $n = |\mathsf{N}| > 3f$ as the number of shares, f the number of malicious parties and threshold t as $t = |\mathsf{N}| - f$, then no PPT adversary can *abuse United-π* to generate two conflicting certificates ψ,ψ' with the non-equivocation attribute.

Proof. The only place in *United-π* where authorities use their share of the group signing key, is in line 1.1.6 of the Incoming algorithm (Alg. 16), where an authority generates a share for a non-equivocal certificate ψ, i.e., ψ has the NEQ attribute. 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 NEQ 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:

$$n_\psi + n_{\psi'} \leq |\mathsf{N}| - f \tag{4}$$

Assume, without loss of generality, that $n_\psi \geq |\mathsf{N}| - 2f$, i.e., there are enough signature-shares from honest authorities to combine into a valid certificate ψ with the NEQ attribute. Following Eq. 4:

$$n_{\psi'} \leq |\mathsf{N}| - f - (|\mathsf{N}| - 2f) = f$$

hence the total number of shares of signatures for ψ' is at most $2f$ ($n_{\psi'}$, plus f malicious authorities). However, since we used a threshold of $|\mathsf{N}| - f > 3f - f = 2f$; hence, there are not enough partial signatures to combine into a valid non-equivocal certificate-upgrade for ψ'. ☐

We now argue that if there exists an adversary $\mathcal{A}$ that can win the non-equivocation' safety experiment under **Exec**', we can use $\mathcal{A}$ to contradict the unforgeability of $\mathcal{TS}$.

Claim 11. For every PPT adversary $\mathcal{A}$ that achieves

$$Pr[\mathbf{SafetyExp'}^{\mathsf{NEQ},\mathcal{M}}_{\mathcal{A},\mathcal{P}}(1^\kappa,\mathsf{N},f)=1]\notin Negl(1^\kappa)$$

there exists a PPT adversary $\mathcal{A}$' that achieves

$$Pr[\mathbf{Exp}^{EU}_{\mathcal{A}',\mathcal{TS}}(1^\kappa)=1]\notin Negl(1^\kappa)$$

Proof. To prove this claim, we demonstrate that if such adversary $\mathcal{A}$ exists, then there exists an adversary $\mathcal{A}$' that given a public group verification key v is able to generate a message m and a signature σ such that

$$\mathcal{TS}.\mathsf{Ver}(v,m,\sigma)=\top$$

without knowing more than f of the partial signing keys, thus contradicting the unforgeability of $\mathcal{TS}$.

First, note that following Claim 10, such adversary $\mathcal{A}$ cannot abuse *United-π* in a way that would cause honest authorities to generate conflicting partial-signatures for it.

However, assume to the contrary that such adversary $\mathcal{A}$ exists. Since $\mathcal{A}$ 'win' in the **SafetyExp**' experiment, the conditions in lines $6.1 - 6.3$ of the experiment must hold. In particular, $\mathcal{A}$ managed to produce two certificates ψ and ψ', with the same identifier $\psi.id = \psi$'$.id$ but different public information, i.e., $\psi.pub \neq \psi$'$.pub$, which are both 'valid', i.e., $\mathcal{P}.\mathsf{WasValid}(\psi,\mathsf{NEQ}) \wedge \mathcal{P}.\mathsf{WasValid}(\psi',\mathsf{NEQ})$. Hence, we have:

$$\begin{aligned}\mathcal{TS}.\mathsf{Ver}(v,m,\sigma) &\equiv\\ \mathcal{TS}.\mathsf{Ver}(S.PubInfo.tvk,(Core(\psi),\mathsf{NEQ}),\psi.\rho[\mathsf{NEQ}].\sigma) &\equiv\\ \mathcal{TS}.\mathsf{Ver}(S.PubInfo.tvk,(Core(\psi'),\mathsf{NEQ}),\psi'.\rho[\mathsf{NEQ}].\sigma) &\equiv \top\end{aligned} \tag{5}$$

Consider an adversary $\mathcal{A}$' that executes $\mathcal{A}$. Since $\mathcal{A}$ can generate two conflicting certificates ψ,ψ' as described in Eq. 5 with non-negligible probability, and following claim 10 that ψ' was not 'honestly' generated by *United-π* honest authorities, it shows that $\mathcal{A}$' is able to generate a message m and signature σ over m with only the knowledge of v and up to f of the signing key shares, thus contradicting the unforgeability of $\mathcal{TS}$. ☐

We now complete the second phase of our proof by arguing that *United-π* achieve non-equivocation'.

Claim 12. If *United-π* uses a secure threshold signature scheme $\mathcal{TS}$, then either *United-π achieves non-equivocation'*, or $\mathcal{TS}$ is not a secure threshold signature scheme.

Proof. Assume to the contrary that there exists an efficient (PPT) adversary $\mathcal{A}$ that negates the claim that *United-π* achieve non-equivocation', i.e.:

$$Pr[\mathbf{SafetyExp'}^{\mathsf{NEQ},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa,\mathsf{N},f)=1]\notin Negl(1^\kappa)$$

However, following Claim 11, this means that we can use $\mathcal{A}$ to construct a PPT adversary $\mathcal{A}$' that breaks the unforgeability of $\mathcal{TS}$.

Therefore, when using a secure threshold signature scheme $\mathcal{TS}$, *United-π* achieves non-equivocation', i.e., non-equivocation under the **Exec**' model. $\square$

We complete the proof that *United-π* achieves non-equivocation with the last phase of our proof methodology. We already showed that *United-π* achieves non-equivocation'. To prove that *United-π* also achieves non-equivocation, we need to show that the **Exec** execution model does not provide any advantage to the adversary in comparison to **Exec**'. Namely, the fact that *United-π* uses the secure encryption scheme $\mathcal{E}$ in **Exec** (yet not in **Exec**'), does not provide the adversary with an advantage.

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

1) The game randomly chooses $b \in \{0,1\}$.
2) If $b = 0$, we execute $\mathcal{A}$ under the $\mathbf{Exec'}_{\mathcal{A},\textit{United-}\pi}$ model, and if $b = 1$, we execute $\mathcal{A}$ under the $\mathbf{Exec}_{\mathcal{A},\textit{United-}\pi}$ model.
3) $\mathcal{A}$ outputs b' $\in \{0,1\}$.

$\mathcal{A}$ wins the game if $b = b$'.

We now argue that *United-π* achieves non-equivocation.

Claim 13. If *United-π* uses a secure encryption scheme $\mathcal{E}$ and secure threshold-signature scheme $\mathcal{TS}$, then it achieves non-equivocation.

Proof. The difference between the **Exec** and the **Exec**' models is that the adversary has an advantage in **Exec**, because it also receives the encrypted secret information generated by $\mathsf{GroupGen}$ using a secure encryption scheme $\mathcal{E}$. Claim 12 shows that *United-π* is secure under the **Exec**' model, since $\mathcal{TS}$ is secure; we now show that the security under the **Exec** model also holds.

Assume to the contrary that although $\mathcal{E}$ is secure; there exists an adversary $\mathcal{A}$ that negates the claim that *United-π* achieve non-equivocation under the **Exec** model, namely:

$$Pr[\mathbf{SafetyExp}^{\mathsf{NEQ},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa,\mathsf{N},f)=1] \notin Negl(1^\kappa)$$

Since the only difference between **Exec** and **Exec**' 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 $\mathbf{Exp'}^{CPA-IND}_{\mathcal{A},\mathcal{E}}(1^\kappa)$ indistinguishability game, and outputs $b' = 1$ if $\mathcal{A}$ wins the $\mathbf{SafetyExp}^{\mathsf{NEQ},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^\kappa,\mathsf{N},f)$ experiment (since we conclude it is an execution under the **Exec** model, where $\mathcal{A}$ has an advantage), and outputs $b' = 0$ otherwise (since it is probably an execution under the **Exec**' model). Consequently, if such $\mathcal{A}$ exists, then we are able to construct $\mathcal{A}$' that wins the $\mathbf{Exp'}^{CPA-IND}_{\mathcal{A},\mathcal{E}}(1^\kappa)$ experiment with a non-negligible probability, thus contradicting the indistinguishability of $\mathcal{E}$.

Therefore, *United-π* achieves non-equivocation (also under the **Exec** model). $\square$

D. Proof of Liveness

Recall that *United-π* employs an *immediate response* approach, where every algorithm's execution produces an immediate non-pending response, except for upgrading a certificate with the non-equivocation attribute; in such a case, a pending certificates is generated first. As a result, we only show that *United-π* satisfies the liveness requirements of non-equivocation, as the rest of the liveness properties are trivially achieved by *United-π*'s immediate response approach.

We show that *United-π* achieves liveness of non-equivocation in a two steps process. First, we argue that in any valid execution of *United-π*, 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 14). Then, we show that an adversary under *United-π*'s threat model cannot prevent a valid non-equivocation upgrade, thus resulting in *United-π* satisfying the liveness requirements of non-equivocation (see Claim 15).

Claim 14. Let $[t,\psi,\iota,R] \leftarrow \mathbf{Exec}_{\mathcal{A},\textit{United-}\pi}(1^\kappa,\mathsf{N},f)$ be an execution of the **Exec** execution model, where R are the execution details, ι is the honest authority chosen by the adversary $\mathcal{A}$ and ψ a pending non-equivocation certificate. If the liveness criteria of non-equivocation in *United-π* is met, i.e., $\mathsf{LiveC}_{\mathsf{NEQ}}(R) = \top$, then on any time $t > \psi.\rho[\mathsf{NEQ}].clk + \Delta$, ι outputs an upgraded non-equivocal certificate ψ' such that $Core(\psi) = Core(\psi') \ \wedge \ \mathcal{P}.\mathsf{WasValid}(\psi',\mathsf{NEQ}) = \top$.

Proof. Assume to the contrary that there exists an adversary $\mathcal{A}$ that is able to cause **Exec** to output execution R and time $t > \psi.\rho[\mathsf{NEQ}].clk + \Delta$ where ι outputs ψ' such that $\mathcal{P}.\mathsf{WasValid}(\psi',\mathsf{NEQ}) \neq \top$, even though the liveness requirement was met, i.e., $\mathsf{LiveC}_{\mathsf{NEQ}}(R) = \top$. However, following the liveness criteria of non-equivocation in *United-π* described in Section V-A, if $\mathsf{LiveC}_{\mathsf{NEQ}}(R) = \top$ then at least $3f + 1$ authorities out of N sent approvals to ι and they did so before Δ time has expired. Since there are only f malicious authorities and *United-π* do not accept more than one partial proof per authority, ι had at least $2f + 1$ (i.e., enough) valid partial proofs from honest authorities to combine before Δ expires, and no more than f invalid partial proofs; hence, ι would have generated a valid combined proof of non-equivocation. Therefore, since ι is an honest authority, ι would have outputted an upgraded certificate on any time $Clk^t_\iota \geq \psi.\rho[attr].clk + \Delta$, thus contradicting the assumption that such $\mathcal{A}$ exists. $\square$

Claim 15. If *United-π* uses a robust threshold signature scheme $\mathcal{TS}$, then either *United-π* achieves liveness of pending certificate upgrade for non-equivocation attribute, or $\mathcal{TS}$ is not a robust threshold signature scheme.

Proof. A robust (t,n)-threshold-signature scheme $\mathcal{TS}$ ensures that performing $\mathcal{TS}.\mathsf{Combine}$ on $t' \geq 2t+1$ partial signatures where at most t of them are invalid outputs a valid group signature. Since in *United-π* $n > 3f$, any call to $\mathcal{TS}.\mathsf{Combine}$ is with at least $2f+1$ valid partial signatures and thus outputs a valid group signature, as long as $\mathcal{TS}$ is indeed a robust (t,n)-threshold-signature.

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

$$Pr[\mathbf{LivenessExp}^{\mathsf{Upgrade},\mathsf{LiveC}_{\mathsf{NEQ}},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}(1^{\kappa},\mathsf{N},f)=1]\notin Negl(1^{\kappa})$$

Thus, following experiment $\mathbf{LivenessExp}^{\mathsf{Upgrade},\mathsf{LiveC}_{\mathsf{NEQ}},\mathcal{M}}_{\mathcal{A},\textit{United-}\pi}$, it means that although the liveness criteria $\mathsf{LiveC}_{\mathsf{NEQ}}(R)$ was met, an upgraded certificate was not issued by an honest authority ι. However, following Claim 14 and *United*-π's attack and communication model, we know that if $\mathsf{LiveC}_{\mathsf{NEQ}}(R)=\top$, 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 $\mathcal{TS}$.Combine enough valid partial signatures (at least $2f+1$) into a valid group signature with only f invalid partial signatures under model $\mathcal{M}$ that ensures reliable communication, thus contradicting the robustness of $\mathcal{TS}$.

Therefore, when using a robust threshold signature scheme, *United*-π achieves liveness of pending certificate upgrade for non-equivocation attribute. □

Appendix B
Additional Explanations for Safety Requirements

Below we provide explanations for the remaining four safety requirements from Section IV-D.

A. Transparency: Requirement 6 presented in Algorithm 3.

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

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 5.1). Notice that the game checks that the transparency issuer is honest and not the issuer of the certificate itself (i.e., accountability issuer), 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. The third requirement is that there is an honest authority ι that is not aware of the transparency of ψ on time t (lines 5.2-5.3). This requirement is validated by verifying that on time t authority ι was instructed to perform $\mathcal{P}$.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 $\mathcal{P}$.Query algorithm outputs all certificates associated with $\psi.id$, this is a proof that an honest authority is not aware of ψ.

B. Non-equivocation: Requirement 7 presented in Algorithm 4.

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

The adversary wins the game if three requirements are met. First, ψ and ψ' must be valid certificates (ψ provided by the adversary and ψ' by some ι, lines 4-5) with the non-equivocation attribute (line 6.1). Second, both certificates have overlapping validity periods (line 6.2). Third, ψ must also be a non-revoked certificate (line 6.3).

C. Revocation Accountability: Requirement 8 presented Algorithm 5.

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

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

D. Revocation Transparency: Requirement 9 presented Algorithm 6.

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

The adversary wins the game if four requirements are met. First, ψ_p must be a valid transparency pending certificate (line 6.1). Second, authority x and the transparency issuer of ψ_p must be an honest authority (line 6.2). Notice that the game checks that the transparency issuer is honest and not the issuer of the certificate itself (i.e., accountability issuer), since even if the issuer of the certificate is corrupted, this does not prevent the transparency issuer from broadcasting the certificate to the rest of the authorities. The third requirement is that there is an honest authority x that is not aware of the transparency of ψ_p on time t (line 6.3). This requirement is validated by verifying that on time t authority x was instructed to perform $\mathcal{P}$.Query with $\psi_p.id$, but the output of this invocation did not contain a transparent certificate based on ψ_p (line 6.3). Since the $\mathcal{P}$.Query algorithm outputs all the certificate associated with $\psi_p.id$, this is a proof that an honest authority is not aware of ψ_p. The fourth and final requirement ensures that time t was sometime after the time that ψ_p was supposed to be transparent $\psi.\rho[\Delta\mathsf{TRA}].t$ (line 6.4). This requirement ensures that the evidence the adversary obtained of winning the game is indeed relevant, since until time $\psi.\rho[\Delta\mathsf{TRA}].t$ not all honest authorities are required to know ψ_p.