Secure Composition of PKIs with Public Key Protocols

Vincent Cheval, Véronique Cortier, Bogdan Warinschi

Secure composition of PKIs with public key protocols

Vincent Cheval*, Véronique Cortier*, Bogdan Warinschi†
*LORIA, CNRS/INRIA/UL, France
†University of Bristol, UK

e-mail: vincent.cheval@inria.fr, veronique.cortier@loria.fr, bogdan@compsci.bristol.ac.uk

***Abstract*—We use symbolic formal models to study the composition of public key-based protocols with public key infrastructures (PKIs). We put forth a minimal set of requirements which a PKI should satisfy and then identify several reasons why composition may fail. Our main results are positive and offer various trade-offs which align the guarantees provided by the PKI with those required by the analysis of protocol with which they are composed. We consider both the case of ideally distributed keys but also the case of more realistic PKIs.**

Our theorems are broadly applicable. Protocols are not limited to specific primitives and compositionality asks only for minimal requirements on shared ones. Secure composition holds with respect to arbitrary trace properties that can be specified within a reasonably powerful logic. For instance, secrecy and various forms of authentication can be expressed in this logic. Finally, our results alleviate the common yet demanding assumption that protocols are fully tagged.

1. Introduction

Modular analysis of cryptographic systems is an area of permanent concern in security research. Composition results are set either in the symbolic model (e.g. [22], [19], [20], [7], [30], [25], [17], [3]) or in the computational model (e.g. [15], [31], [16], [27], [26], [14], [10], [12]). Some of these results yield general frameworks where arbitrary components can be safely combined but, unsurprisingly, rely on particularly strong hypothesis. Other results provide narrower composition theorems tailored to specific cryptographic tasks but afford more relax and practical assumptions on the components.

This paper falls within the latter research direction. We study the composition of protocols for establishing public key infrastructures (PKIs) with arbitrary other protocols which require such keys. For two parties the question we adress is (using ad-hoc notation): when can a PKI protocol $P = P_1 \mid P_2$ that distributes public (and secret) keys be composed with a protocol $Q = Q_1 \mid Q_2$ that uses these keys. We are after a theorem which (using informal notation) guarantees that

$$P_1 \mid P_2 \models \phi_{\mathsf{PKI}} \quad \Rightarrow \quad P_1.Q_1 \mid P_2.Q_2 \models \mathsf{secrecy}(s)$$

provided that

$$Q_1(sk_A, \mathsf{pk}(sk_B)) \mid Q_2(sk_B, \mathsf{pk}(sk_A)) \models \mathsf{secrecy}(s)$$

That is, a secure PKI infrastructure P (as captured by some security property ϕ_{PKI}) can be safely used by a protocol Q provided that Q is secure (preserve the secrecy of some piece of data s) when analyzed with honestly generated keys. In the above theorem, think of P_1 as the PKI component which provides the secret key to user A and informs him of the public key of user B; protocol P_2 plays the converse role for user B.

This type of composition result is often simply assumed! For example, it is quite common for the analysis of protocols that use public keys to rely on the assumption that prior to their execution the PKI keys have been generated, distributed to all parties, and that the link between the identities of parties and their public keys is known to everyone. This convenient idealization of key distribution is often adopted by analysis via automatic tools (e.g. ProVerif[6], Scyther [21], Avispa [4], or Tamarin [29]) and reflects compelling inuition: PKI infrastructures (such as X.509) are designed to distribute and certify keys, *independently* of the protocols that will use such keys. This intuition is not supported by rigorous underpinnings and it may actually be wrong since the guarantees provided by the PKI are not always aligned with those assumed by the subsequent protocol. For example, PKIs do not guarantee that a malicious party registers the same key with different registration authorities (so the same user may have two different public keys), do not guarantee that different users do not share the key or, more generally, that each user has followed honestly the registration process. In fact, it is not clear even what minimal guarantees for a PKI can still ensure a composition result.

There has been relatively little research on the problem of secure composition of PKIs with other protocols and, interestingly, most work is set within computational model (see related work). In this paper we approach the problem using symbolic models. As we discuss later, the higher level of abstraction yields results of broader applicability. Moreover, our results are relevant for proofs that use automated tools since they usually work on top of some symbolic model.

In a nutshell, our results are as following. We reconfirm that the mismatch between assumptions made in the analysis of Q and the guarantees offered by some PKI protocols P leads to insecure composition. Our main results are then rigorous composition theorem which carefully account for the mismatch between what is assumed and what is guaranteed.

Below we highlight the main features of our work. A full version of this paper is available here [1].

THE PERILS OF IDEALIZED KEY DISTRIBUTION. At a high level the theorem we are after may fail because of implicit assumptions that underly the analysis of Q. As explained above, the assumption is usually that the keys of parties have been honestly generated and are in place before the execution of Q. For a variety of reasons, this assumption does not hold when keys are managed by a real PKI. For example, when parties generate their own secret/public keys and have them certified (e.g. the Verisign process for issuing certificates), keys are not necessarily honestly generated. Other causes include "confusion" of messages between the protocol for key registration and subsequent protocols that use these keys (which makes standalone analysis incomplete) as well as the classic message parsing errors. In Section 2 we illustrate through concrete counterexamples several obstacles which need to be carefully accounted for by any generic composition theorem.

COMPOSITION OF PKIS WITH ARBITRARY PROTOCOLS. The counterexamples that we identify inform our main results. We provide sufficient conditions to ensure that a protocol for distributing public keys P composes securely with a protocol Q which uses these keys, in the sense that the desired security properties of Q are guaranteed.

Our first theorem imposes only minimal requirements on P. These are formalized by formula ϕ_{PKI} (in a logic on traces which we provide) and demand that secret keys stay secret, that all parties have a consistent view of the public keys of *honest* parties, and that honest parties use distinct keys for signing and decryption. We emphasize that ϕ_{PKI} provides no guarantees on the keys held by dishonest parties. This weaker guarantees on the keys distributed by P translates into a correspondingly stronger requirement on Q. Its security needs to hold under a *permissive* key assignment (which only reflects the guarantees offered by ϕ_{PKI}). We refer to this version of Q as "permissive" Q.

In symbolic models, protocols are however never analyzed in their "permissive" form, but rather in the "ideal" form where all keys (including those of dishonest parties) are honestly generated and already predistributed. In this case the above theorem does not apply (and in fact we give counterexample protocols that shows that composition with realistic PKIs fails). Our next theorem recovers a composition result for such protocols, at the expense of stronger requirements on the PKI: we strengthen the assumption on P to require that besides ϕ_{PKI} it also satisfies ϕ_{ideal}. This additional assumption essentially asks that all honest users have a consistent view of the keys for encryption and verification that belong to other users, that keys of distinct agents are pairwise distinct and that encryption and verification keys are also distinct.

Our theorems share several features. First, they treat the properties of Q in a generic way. Composition preserves *any trace-based security property* which can be specified by a formula in a logic which we provide. In particular, (weak) secrecy and various forms of authentication properties can be specified in the logic. Interestingly, the security properties we consider for P (ϕ_{PKI} and ϕ_{ideal}) as well as permissive Q can be encoded in existing tools such as ProVerif.

Our theorems are rather *agnostic to the class of protocols* themselves. We consider arbitrary classes of protocols (possibly with else branches) which employ arbitrary cryptographic primitives (including e.g. Exclusive Or or Diffie-Hellman). While we require the minimal condition that P and Q do not share underspecified primitives we do permit that they both use standard ones (encryption, signatures, hashes, ...).

Sharing primitives between protocols leads to well-known difficulties due to cross-protocol attacks. The traditional solution is to assume that each use of these primitives is "tagged" [20], [19], [3]. This convenient technique helps to easily distinguish between messages of different protocols but is not supported by current practice. Our theorems show how one may *avoid full tagging of primitives*. We propose a more general property that avoids cross-protocol attacks. This property can be enforced by tagging mechanism but also through alternative restrictions on the protocols, e.g. that P and Q employ shared functions but always under different keys or by minimal tagging assumptions (e.g. that only occurrences of public keys, and not each individual use of the primitives are tagged). For example, a PKI protocol P may use the same signature (resp. encryption) scheme as Q provided that keys shared from P to Q are either used to sign (resp. encrypt) in P or Q but not both.

A CONVENIENT SPECIFICATION LANGUAGE. Our results are set within a symbolic model similar to those that underlie existing automated tools. It turns out that existing symbolic formalisms (e.g. those close to the applied pi-calculus [2]) are not convenient to specify scenarios like those we treat in this paper. For example, in the applied-pi calculus, it is surprisingly difficult to express persistent storage e.g. of a trusted server of symmetric keys shared by unbounded number of pairs of agents. Modeling such a server requires a heavy encoding using private channels. Tools like ProVerif bypass this encoding by extending their calculus to include tables. The problem is that the notion of "agent" is captured implicitly in existing calculi and this makes it difficult to reason in a simple manner about composition.

We design a new specification language. The main feature is a notion of parameterized agents and names which allows to conveniently talk about the different sessions of protocols that share the same parameters. For example, we can elegantly express that a server shares a symmetric key K_{AS} with any agent A by writing $k[A,S]$. Then one server talking to infinitely many agents can be simply described by a process of the form

$$!^i R_1(k[S,A[i]]) \mid R_2(k[S,A[i]])$$

where R_1 and R_2 represent respectively the role of the server and the agent. This extension can still be encoded in tools like ProVerif, for proof purposes.

RELATED WORK. Our work uses and extends techniques used in other existing composition result set withing symbolic models, and is close in scope with some recent works on the composition of PKIs that rely on computational models.

Relevant composition results within symbolic models include [20] which characterizes when two protocols run in parallel may share keys and [19] which studies what is a good key establishment protocol and how it can be used. These early works hold for trace properties. More recent results establish similar results in the context of equivalence properties, useful to model privacy properties [18], [3]. In [30], [25], [17], the authors study "vertical composition": when a protocol Q uses some secure, authenticated, or confidential channel, how such a channel can be securely realized? In contrast, our paper focuses here on PKI and studies what are the properties of a good PKI and how it can be used. Our proof techniques borrow from [3], [17]. However, in addition to considering a different type of composition (PKI), we establish the first composition result that does not require an explicit tagging scheme. In other words, we can now compose actual protocols instead of composing their tagged version. To establish such a general result, we had to considerably reshape the proofs developed e.g. in [3] or [17].

In the computational model there are several generic frameworks for compositional analysis [15], [31], [26], [14] all sharing the same underlying philosophy: components can be designed separately, yet their security is preserved when the components are used together, so composability comes somehow for free. The strength of this level of security also means that it may be difficult to achieve. Indeed, for public key infrastructures the model for PKI as introduced by Barak et al. [5] and later refined by [24] can only be achieved by registration protocols which essentially ensure that the PKI also learns the associated secret key.

The works of Boldyreva et al [8] and of Boyd et a. [9] are closest in spirit to ours in that they are exclusively concerned with the use of PKIs within other protocols and primitives. Boldyreva et al. [8] consider the security of asymmetric encryption and digital signatures in the presence of attackers that can also interfere with the registration process of long term keys. That work considers composition of PKIs with these two important primitives but leaves the study of implications to higher level protocols for future work. More recently, Boyd et al. [9] have looked at the use of PKIs within key exchange protocols. They extend standard cryptographic model for key exchange with adversarial capabilities that reflect potential PKI interference (like registering malformed keys). The models can then be used to construct key exchange protocols that protect against some weaknesses in the PKI. However, strictly speaking the results are not compositional results: there are no guarantees for when the PKI is instantiated with an actual protocol.

2. Why composing with a PKI is hard?

In this section we discuss in more details why composition with a PKI does not work so well in general, providing counter-examples and spelling out the assumptions we will consider in the rest of the paper.

2.1. Minimal assumptions on the PKI

We first state what we view as the minimal property that we believe a PKI should satisfy. Informally, we demand that:

- An honest agent has a unique public/private key pair and a unique verification/signing key pair.
- Honest agents have pairwise distinct private/signing keys.
- Keys are consistently distributed, that is, honest agents know each other public and verification keys.
- Decryption/signing keys of honest agents are private.

In this paper we explore whether these properties are actually sufficient: can a PKI that satisfies the requirements above be safely used together with any public key protocol Q?

2.2. Standard assumptions

Since composing a public-key protocol involves sharing key material, we of course face the same issues as existing composition results [20], [19], [18], [3]. In particular, one of the protocols could act as a decryption oracle for the other one. For example, assume that the PKI includes a challenge response phase where the authority checks that A knows her private key.

$$\begin{aligned} Auth \to A &: \{N\}_{\mathsf{pkA}} \\ A \to Auth &: N \end{aligned}$$

This challenge phase may occur during the registration of the key but also later, for example if A wishes to extend the validity of the certificate associated with her key. Such a PKI would break the security of most protocols that use public keys. Consider for example, the following simple protocol Q where B sends a secret to A using her public key.

$$B \to A : \{s\}_{\mathsf{pkA}}$$

Then Q executed in isolation with predistributed keys is secure (it does not compromise s) while Q composed with the PKI described above is insecure.

The standard way for preventing such behaviours [20], [19], [18], [3] consists in tagging the encryption scheme which a tag $\mathtt{t}_{pki}$ that is specific to the protocol.

$$\begin{aligned} Auth \to A &: \{\mathtt{t}_{pki}, N\}_{\mathsf{pkA}} \\ A \to Auth &: N \end{aligned}$$

In this paper we go one step further: we introduce a more general property which ensures that messages of different protocols are not confused. The usual tagging mechanism is only one way to enforce this property. We show that it suffices to ensure that functions shared between protocols use different keys. In particular, a PKI protocol P may use the same signature (resp. encryption) scheme as Q, if keys provided by P to Q are used to sign (resp. encrypt) either by P or Q but not by both. In practice, this condition is often satisfied and allows us to compose protocols without requiring a tagging scheme. A typical exception are protocols where the PKI protocol uses the keys it provides to also carry out a challenge response as proof of possession.

2.3. Confusing public keys with other material

Analysis of public key protocols typically assumes that the keys of all parties have been honestly generated and distributed. However, this assumption is not valid in all scenarios, e.g. in the Verisign process for issuing certificates where users generate their own secret/public keys and have them certified. The following example shows that the mismatch between assumption and reality may be problematic.

Assume a setting where public keys (or verification keys) are used to identify parties. Consider a simple protocol where A sends a message M to B, signed together with the identity of B, to indicate that M is meant for B.

$$A \to B \quad : \quad [\mathsf{pkB}, M]_{\mathsf{skA}}$$

When B receives the message M, he is convinced that A sent it to him. Consider an attacker C who registers the string $\mathsf{pkC} = \mathsf{pkB}, \mathsf{pkC}'$ as his public key. Note that he may not be able to decrypt message encrypted by pkC (or properly sign with the corresponding key) but this still allows him to mount an attack against the simple signing protocol.

$$\begin{array}{rcl} A \to C & : & [\mathsf{pkC}, M]_{\mathsf{skA}} \\ C(A) \to B & : & [\mathsf{pkB}, \mathsf{pkC}', M]_{\mathsf{skA}} \text{ since } \mathsf{pkC} = \mathsf{pkB}, \mathsf{pkC}' \end{array}$$

That is, when A initiates a session with some malicious party C, C can use the message in this session to impersonate A towards user B. In practice, it could be the case that A was requesting a service to C, and the attacker uses the corresponding message to request a service to B, in the name of A. So, while the protocol is secure when long term keys of parties are honestly generated, the protocol is insecure if parties manage to register malformed keys.

One way of circumventing this issue is to tag keys that are part of messages (i.e. not used for signing/encryption).

$$A \to B \quad : \quad [\mathsf{t}_{\mathsf{pkey}}(\mathsf{pkB}), M]_{\mathsf{skA}}$$

Such a tagging makes sense as soon as the format of messages ensures that public keys cannot be confused with other material. However examples in the following sections show that this countermeasure is not sufficient.

2.4. Confusing public keys with other keys

Even when public keys are not sent as payload, the adversary may chose his dishonest key so that he can trigger unexpected behaviors. Consider the case where A sends out a secret encrypted with C's public keys and (immediately) decrypts the message with her private key.

$$A \to B \quad : \quad \mathsf{adec}(\{s\}_{\mathsf{pkC}}, \mathsf{skA})$$

Then C may simply chose his public key to be the public key of A: $\mathsf{pkC} = \mathsf{pkA}$, yielding an attack in the protocol Q described above. A similar issue occurs if A uses fixed, long term, asymmetric keys. Our example is admittedly contrived but it conveys well the composition issue. Intuitively, some message may be encrypted at some point with a public key which provenance is not known (this encryption may be embedded in a safer encryption). And later, a decryption guarantees that the message is processed only if the used public key was an honest one. We could make it more realistic by complexifying Q.

2.5. Several public keys for the same identity

One problem which is not fixed by the use of tagging is that dishonest parties may register and use different keys to identify themselves to different users. This may yield composing issue as soon as Q contains (non trivial) else branches. Notice that a dishonest agent may register different keys for his identity. For example, A may believe C's public keys is pkC while B believes C's public keys is pkC'. This can created unexpected disequalities that undermine the security of Q, as illustrated by the following example.

$$\begin{array}{rcl} A \to B & : & [B, \mathsf{pkB}]_{\mathsf{skA}} \\ A : [x, y_1]_{\mathsf{skA}}, [x, y_2]_{\mathsf{skA}} \xrightarrow{y_1 \neq y_2} B & : & s \end{array}$$

A sends the public key of B as viewed by A, signed by A. Then whenever A receives two certificates (one from A and one from B), she may check that the two public keys coincide. By interacting with two sessions of the protocol (one with A and one with B), the attacker could obtain both $[C, \mathsf{pkC}]_{\mathsf{sigA}}$ and $[C, \mathsf{pkC}']_{\mathsf{sigB}}$, and send them to A to learn the secret s. While this example is again contrived, it presents well the intuition: in case two honest agents A and B do not share the same view on C, some unexpected behaviors may occur. To circumvent this issue, we have two options: either consider more demanding properties on the PKI (even dishonest keys should be consistently distributed) or we analyze a more flexible Q (see next sections). We explore both options in this paper. This issue did not surface in previous composition results since they either do not consider else branches [20], [19] or do not consider dishonest keys [18], [3].

2.6. Related keys

Even if a PKI guarantees that honest agents have pairwise distinct public keys, there is no guarantee that these keys are independent. Key dependencies may lead to insecurity, as exemplified by our next (pathological) example. Assume that, possibly in interaction with the PKI, A obtains k as private key ($\mathsf{skA} = k$) while B obtains $\langle k, k \rangle$ ($\mathsf{skB} = \langle k, k \rangle$). This would break the security of the following protocol Q.

$$\begin{array}{rcl} A \to B & : & \{N\}_{\mathsf{pkB}} \\ A : \mathsf{in}(x).\ \mathsf{let}\ y = \mathsf{adec}(x, \langle \mathsf{skA}, \mathsf{skA} \rangle) \xrightarrow{y=N} B & : & s \end{array}$$

Then A sends a fresh nonce N encrypted with B's public key and then expects a message x, decrypts it with $\langle \mathsf{skA}, \mathsf{skA} \rangle$ and leaks a secret (s) if she retrieves her nonce N. This protocol would clearly be insecure with the PKI sketched above while when keys are honestly and independently generated the protocol is clearly secure.

One way to circumvent this problem is to ensure through syntactical requirements that Q cannot break due

to dependencies of keys provided by P. Interestingly, this issue is not specific to public key distribution. However, previous results discarded such behaviors by either requiring disjoint primitives [19] (the two protocols may not both use concatenation) or requiring explicitly that keys established by P are atomic or at least viewed as atomic by Q [18], [3].

2.7. Permissive Q

The examples in the previous sections show that typical security assumptions on a PKI fail, in more than one way, to allow composition with arbitrary public key protocols. One option to recover composability is to require more from the PKI, in particular w.r.t. the dishonest keys. Another option is to analyze Q under a more "permissive" assumption which makes no restrictions on how keys of dishonest parties are created. For example, instead of analyzing sessions between A, B and a dishonest C with perfectly distributed keys:

$$Q_1(sk_A, \mathsf{pk}(sk_B)) \mid Q_2(sk_B, \mathsf{pk}(sk_A)) \mid Q_2(sk_B, \mathsf{pk}(sk_C))$$

we may let agents input dishonest keys from the adversary.

$$Q_1(sk_A, \mathsf{pk}(sk_B)) \mid Q_2(sk_B, \mathsf{pk}(sk_A)) \mid \mathsf{in}(x).Q_2(sk_B, x)$$

Indeed, we show that if this permissive Q is secure then it can be safely composed with a PKI, under much lighter assumptions. Interestingly, such a permissive Q can be easily encoded in existing tools (e.g. ProVerif, Tamarin, Scyther).

2.8. Summary

In this paper, we conduct a thorough analysis on how to compose safely protocols with a PKI. Our main results are summarized in Table 1. The rest of the paper is devoted to the formalization and proof of these results. Interestingly, we do not need to prove each result separately. Instead, we can derive them from a general composition result (which will not be fully stated in the paper, due to space constraints).

3. Framework

A cryptographic protocol describes how agents exchange messages over a network. A standard framework for modelling cryptographic protocols is a process algebra, such as the applied pi-calculus [2]. It is typical for existing approaches to have an implicit notion of agents: an honest agent is modeled as a process while dishonest agents are not described – their private keys are simply passed to the attacker. This is not sufficient to describe some trust scenarios like those underlying our results. We therefore introduce novel specification framework which enhances the traditional process algebra with an explicit notion of agent. In particular, our framework provides the user directly with an intuitive notion of honest and dishonest agents, discharging him from having to hard-code which keys are known to the attacker. We also believe that it can be used to specify more complex scenario such as subnetworks, e.g. the particular protocol topology described in the introduction where a server links an unbounded number of pairs of parties.

3.1. Messages and agents

We assume a set of *names* $\mathcal{N}$ used to represent keys, nonces, etc. We consider a set of *agents* $\mathcal{A} = \mathcal{A}_D \uplus \mathcal{A}_H$ where $\mathcal{A}_D$ (resp. $\mathcal{A}_H$) represents the dishonest (resp. honest) agents, a set of *integer variables* $\mathcal{X}_\mathbb{N}$ and a set of *variables* $\mathcal{X} = \mathcal{X}_t \uplus \mathcal{X}_a$ where $\mathcal{X}_t$ represents term variables and $\mathcal{X}_a$ agent variables. All these sets are infinite. Lastly, we consider a signature $\mathcal{F} = \mathcal{F}_c \uplus \mathcal{F}_d$ which consists of a finite set of function symbols and their arity. The subsets $\mathcal{F}_c$ and $\mathcal{F}_d$ represent constructor and destructor function symbols.

Many calculi (e.g. applied pi calculus [2]) rely extensively on the renaming of bounded variables and names in presence of replication. However, this renaming becomes an hindrance when one needs to refer to specific variables or names during the protocol execution. Moreover, the renaming does not allow a simple mapping between the different agents and their shared knowledge. We therefore replace part of the renaming with an alternative mechanism that relies on the notion of *parametrized agent* and *parametrized names*.

We define the set of parametrized agents $\overline{\mathcal{A}}$ as the set of elements from $\mathcal{X}_a$ or of the form $A[p_1, \ldots, p_n]$ where $A \in \mathcal{A}$, $n \in \mathbb{N}$ and $p_i \in \mathbb{N} \cup \mathcal{X}_\mathbb{N}$ for $i = 1 \ldots n$. We say that a parametrized agent $A[p_1, \ldots, p_n]$ is honest (resp. dishonest) when $A \in \mathcal{A}_H$ (resp. $\mathcal{A}_D$) and we denote by $\overline{\mathcal{A}}_H$ (resp. $\overline{\mathcal{A}}_D$) their set. When there is no parameter, we write A for $A[]$. For example, for a typical protocol, we will simply consider one honest parameterized agent $H[i]$ and one dishonest parameterized agent $D[i]$ to model honest agents $a_1, a_2 \ldots$ and dishonest agents $d_1, d_2 \ldots$. A local server talking to agents inside an internal network can be be modeled as $S[i]$ with agents $A[i, j]$ where agents $A[i,1], \ldots, A[i,n]$ only talk to $S[i]$.

Similarly, we define the set $\overline{\mathcal{N}}$ of parametrized names as the set of elements of the form $k[A_1, \ldots A_n]$ where $n \geq 1$, $k \in \mathcal{N}$ and $A_i \in \overline{\mathcal{A}}$ for $i = 1 \ldots n$. We say that a parametrized name $k[A_1, \ldots, A_n]$ is honest when $A_1, \ldots, A_n$ are all honest and is otherwise dishonest.

Terms are inductively defined as variables, names, parametrized names and agents, closed by application of function symbols (in a way that complies with arities). We say that a term t is a *constructor term* when t does not contain destructor function symbols. A term t is *ground* if it does not contain any variables and integer variables.

The destructor and constructor function symbols represent the cryptographic primitives used in the protocol. We model their behavior by means of a *rewriting system* $\mathcal{R}$ and an *equational theory* E that are standard rewriting techniques used in symbolic cryptographic models (e.g. [23]). In our model we require that the equations in E are between name-free constructor terms and that the rewrite rules in $\mathcal{R}$ are of the form $\mathsf{f}(t_1, \ldots, t_{n-1}) \to t_n$ where $t_1, \ldots, t_n$ are name-free constructor terms and $\mathsf{f} \in \mathcal{F}_d$. Moreover, we assume that $\mathcal{R}$ is convergent modulo E we denote by $u\downarrow$ the normal form of u modulo E. Lastly, we consider the predicate $Msg(u)$ which holds when the normal form modulo E of any subterm of u is a constructor term. In such a case, we say that u is a *message*. Thanks to our expressive modeling which considers both a rewrite system and an equational theory, we can model most

Q secure	P secure PKI	Additional hypotheses		Permanent hypotheses
$\mathcal{Q}_{perm}$	ϕ_{PKI}	Tagged Processes		• Only $\mathscr{F}^0$ as comon signature • Tagged private keys
		Disjoint keys		
$\mathcal{Q}_{ideal}$	$\phi_{\mathsf{PKI}} \wedge \phi_{ideal}$	Tagged Processes	• Tagged public keys	
		Disjoint keys	• Only PKI keys in asym. enc.	

Figure 1: Summary of our composition results

primitives and in particular rather complex primitives such as Exclusive Or, associative concatenation, Diffie-Hellman, or blind signatures.

Example 1. *We consider the signatures* $\mathscr{F} = \mathscr{F}_c \uplus \mathscr{F}_d$ *where* $\mathscr{F}_d = \{\mathsf{sdec}/2, \mathsf{rsdec}/2, \mathsf{adec}/2, \mathsf{radec}/2, \mathsf{check}/2, \mathsf{proj}_1/1, \mathsf{proj}_2/1\}$ *and* $\mathscr{F}_c = \{\mathsf{senc}/2, \mathsf{rsenc}/2, \mathsf{aenc}/2, \mathsf{raenc}/2, \mathsf{pk}/1, \mathsf{sign}/2, \mathsf{vk}/1, \langle\rangle/2, \mathsf{h}/1, \oplus/2, 0/0\}$. *They represent deterministic and randomized symmetric encryption as well as asymmetric encryption, signature, pairing, hash function and exclusive or. Their behavior can be modeled with the following rewriting system* $\mathscr{R}$*:*

$$\begin{array}{rcl}
\mathsf{sdec}(\mathsf{senc}(x,y),y) & \to & x \\
\mathsf{check}(\mathsf{sign}(x,y),\mathsf{vk}(y)) & \to & x \\
\mathsf{adec}(\mathsf{aenc}(x,\mathsf{pk}(y)),y) & \to & x \\
\mathsf{proj}_i(\langle x_1,x_2\rangle) & \to & x_i \text{ with } i \in \{1,2\}) \\
\mathsf{rsdec}(\mathsf{rsenc}(x,y,z),z) & \to & x \\
\mathsf{radec}(\mathsf{raenc}(x,y,\mathsf{pk}(z)),z) & \to & x
\end{array}$$

and the following equational theory E *that models Exclusive Or:*

$$x \oplus x = 0 \quad x \oplus (y \oplus z) = (x \oplus y) \oplus z$$
$$x \oplus 0 = x \qquad x \oplus y = y \oplus x$$

3.2. Processes

Processes in our framework are modeled using the grammar in Figure 2. We discuss its less standard aspects below, and we do so from the perspective of using this specification framework to formulate our results.

Before we go into the details, we introduce a useful refinement of how assignment is usually handled in processes. We are motivated by our composition scenario. A PKI infrastructure P assigns secret and public keys and these keys may then be passed through variable assignment to a process Q that depends on these keys. To indicate what type of terms is expected to be assigned in a variable, we introduce a typed variable assignment $[x :=_\tau u]$. Formally, we consider a set T of types that contains $\mathbb{sk}, \mathbb{pk}, \mathbb{vk}, \mathbb{sig}$, corresponding to types for resp. private, public, verification, and signing keys. Similarly to parametrized names, we also consider the infinite sets $\mathscr{X}_{ty}$ and $\overline{T}$ of type variables and parametrized types respectively. For example, a variable assignment with type $\mathbb{pk}[A,B]$ will typically refer to the public key of B as viewed by A. As we shall see later, this is very convenient to relate variables among different agents and different sessions. Given $\tau \in T$, we denote by $\overline{\tau}_H$ the set of parametrized types $\tau[A_1,\ldots,A_n]$ where $A_1,\ldots,A_n$ are honest.

$$\begin{array}{rll}
P,Q = & 0 & \text{null} \\
| & \mathsf{in}_A(c,x).P & \text{input} \\
| & \mathsf{out}_A(c,u).P & \text{output} \\
| & \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q & \text{conditional} \\
| & P \mid Q & \text{parallel} \\
| & !^i P & \text{replication} \\
| & \mathsf{new}_A k.P & \text{name restriction} \\
| & \mathsf{agt}(X,S).P & \text{agent selection} \\
| & [x :=_\tau u].P & \text{variable assignment}
\end{array}$$

where $A \cup \in \overline{\mathscr{A}}$, $S \subseteq \overline{\mathscr{A}}$, $X \in \mathscr{X}_a$, $x \in \mathscr{X}_t$, $i \in \mathscr{X}_{\mathbb{N}}$, $\tau \in \overline{T}$ and c,u,v are terms.

Figure 2: Grammar of processes

The grammar of our processes is provided in Figure 2 and explained below. Part of our grammar is classical in cryptographic process algebra. Note that we annotate inputs, outputs and name restrictions by the agent performing them. Moreover, a replication $!^i P$ is annotated by an integer i. Intuitively, P is parametrized by the variable i that will be instantiated at each replication by some (non necessarily fresh) integer $n \in \mathbb{N}$. This mechanism allows to differentiate between different replicas of P. For example, $!^i R_1(k[S,A[i]]) \mid R_2(k[S,A[i]])$ represents a server talking to infinitely many agents, each of them sharing a ket $k[S,A]$ with him. Even more interestingly, we can represent the case of an unbounded number of internal networks, where inside each network i, agents may only communicate among themselves and to a router $S[i]$, while routers may communicate between them. The corresponding process is $!^i R_1(S[i]).!^j R_2([S[i],A[i,j]])$, which denotes multiples sessions of R_1 and R_2 but where the $A[i,j]$ may only talk to the same $S[i]$. The process $\mathsf{agt}(X,S)$ selects an agent from S that instantiates X. Lastly, the process $[x :=_\tau u]$ assigns the term u to the variable x typed with τ.

Term variables are bound by input and variable assignment, agent variables are bound by agent selection and names are bound by name restriction. We say that a process P is *a role of* A if all outputs, inputs and name restrictions in P are done by A and all parametrized names and types in P contain A as agent.

Example 2. *We consider a PKI where agents generate their own private/public key pair and signing/verification key pair. They send both public key and verification key to a trusted server S to be signed. When an agent A wishes to establish a connection with another agent B, he will send a request to B along with his own certificate. Upon receiving the certificates*

*of B, A will check that they are signed by the server and they correspond to the public and verification keys of B. The role of the agent can be modeled by the following context process $P_A[\__A]$ where the hole $\__A$ corresponds to where the role of A in the composed protocol (*e.g. *Needham-Schroeder protocol) will be plugged.*

$$\begin{array}{ll}
\mathsf{out}_A(pc[S,A], \langle \mathsf{pk}(sk[A]), \mathsf{vk}(sig[A])\rangle). & \textit{Register}\\
\mathsf{in}_A(pc[S,A], x_{cert}).\\
[x_{skA} :=_{\mathbb{sk}[A]} sk[A]].[x_{sigA} :=_{\mathbb{sig}[A]} sig[A]].\\
\mathsf{out}_A(c, \langle\langle \mathsf{request}, B\rangle, x_{cert}\rangle).\mathsf{in}_A(c,z). & \textit{Request}\\
\mathsf{if}\ \mathsf{proj}_1(\mathsf{check}(z, \mathsf{vk}(sig[S]))) = B\ \mathsf{then} & \textit{Check}
\end{array}$$

$$\begin{array}{ll}
\mathsf{let}\ y = \mathsf{proj}_2(\mathsf{check}(z, \mathsf{vk}(sig[S])))\ \mathsf{in} & \\
[y_{pkB} :=_{\mathbb{pk}[A,B]} \mathsf{proj}_1(y)].[y_{vkB} :=_{\mathbb{vk}[A,B]} \mathsf{proj}_2(y)].__A & \textit{Assign}
\end{array}$$

We use the syntax $\mathsf{let}\ y = u\ \mathsf{in}\ P$ *as a syntactic sugar for* $P\{^u/_y\}$*. Note that in the registration phase, $sk[A]$ and $sig[A]$ represent the private and signing keys of A. Since they are parametrized names, they will remain the same through any different sessions. Furthermore, an attacker does not have directly access to them unless A is dishonest. Also note that the agent A and the server S are communicating through a parametrised channel $pc[S,A]$ meaning that this channel is only shared between A and S. After sending its request to B, the agent A is expecting a message of the form $\mathsf{sign}(\langle B, \langle t_1, t_2\rangle\rangle, sig[S])$ where t_1 and t_2 respectively correspond to the public and verification key of B. Once the agent A verifies the signature and the ownership of the certificate, he assigns the variables accordingly.*

The role of the receiver B is very similar to the one of A and can be modeled by the following context process $P_B[\__B]$. The process modeling the role of the server registering the key of an agent A, denoted $R(A)$, is described as follows:

$$\mathsf{in}_S(pc[S,A], x).\mathsf{out}_S(pc[S,A], \mathsf{sign}(\langle A, x\rangle, sig[S]))$$

A complete session of the PKI with a server S between two agents A and B can thus be modeled by the following context process $P[\__A, \__B] = P_A[\__A] \mid P_B[\__B] \mid R(A) \mid R(B)$. Furthermore, if we want to model unbounded number of sessions between honest and dishonest agents, with a unique trusted server S. It corresponds to the following context process

$$!^i\, \mathsf{agt}(A, \{H[i], D[i]\}).!^j\, \mathsf{agt}(B, \{H[j], D[j]\}).P[\__A, \__B]$$

where $S, H \in \mathscr{A}_H$ and $D \in \mathscr{A}_D$.

The following example models the well-known Needham-Schroeder-Lowe protocol [28].

Example 3. *Needham-Schroeder-Lowe protocol can be informally described as follows.*

$$\begin{array}{lcl}
A \to B & : & \{\mathsf{pkA}, N_a\}_{\mathsf{pkB}}\\
B \to A & : & \{\mathsf{pkB}, N_a, N_b\}_{\mathsf{pkA}}\\
A \to B & : & \{N_b\}_{\mathsf{pkB}}
\end{array}$$

The following process Q_A represents the role of the initiator A in the Needham-Schroeder protocol:

$$\begin{array}{l}
\mathsf{new}_A\, n_a.\mathsf{out}_A(c, \mathsf{aenc}(\langle \mathsf{pk}(x_{skA}), n_a\rangle, x_{pkB})).\mathsf{in}_A(c,y).\\
\quad \mathsf{if}\ n_a = \mathsf{proj}_1(\mathsf{proj}_2(\mathsf{adec}(y, x_{skA})))\ \mathsf{then}\\
\quad \mathsf{if}\ x_{pkB} = \mathsf{proj}_1(\mathsf{adec}(y, x_{skA}))\ \mathsf{then}\\
\quad \mathsf{out}_A(c, \mathsf{aenc}(\mathsf{proj}_2(\mathsf{proj}_2(\mathsf{adec}(y, x_{skA}))), x_{pkB}))
\end{array}$$

Note that x_{skA}, x_{pkB} are free in Q_A. Intuitively, these variables should be bound by a PKI infrastructure process P that assigns variables x_{skA} and x_{pkB} respectively with type $sk[A]$ and $pk[A,B]$ where B is the agent contacted by A.

Definition 1. *A configuration is a tuple $(\mathscr{E}; P; \Phi; \sigma; \mu)$ where:*

- *$\mathscr{E}$ is a set of names that corresponds intuitively to the private names of the process.*
- *P is a process where names and variables are bound only once.*
- *Φ and σ are both substitutions of term variables to ground terms. The variables of $\mathrm{dom}(\Phi)$ do not appear anywhere else in the configuration.*
- *μ is a mapping from $\overline{T}$ to sets of term variables.*

The set $\mathscr{E}$ represents the set of names that have been generated by honest agents. The substitution Φ, also called *frame*, represents the messages that have been sent on channels controlled by the attacker (and which the adversary therefore knows). The substitution σ represents the variables instantiated so far. Lastly, $\mu(\tau)$ for some τ is the set of variables that have been assigned with type τ. We sometimes write P instead of the initial configuration $(\emptyset; P; \emptyset; \emptyset; \emptyset)$.

The attacker can forge new messages by applying *recipes* to his knowledge that is by applying function symbols. He may also use names, except the names generated by honest agents. Formally, given a frame Φ and a set $\mathscr{E}$, we define $\mathsf{Recipe}(\Phi, \mathscr{E})$ as the set of terms M whose variables are of the domain of Φ, whose names are not in $\mathscr{E} \cup \overline{\mathscr{N}_H}$ and that satisfies $Msg(M\Phi)$.

3.3. Semantics

We define the operational semantics of configurations through a transition relation $\mathscr{K} \to \mathscr{K}'$ between configurations. The transition relation is defined by the rules given in Figure 3. We denote by $\to^*$ the transitive closure of $\to$.

The rules follow the intuition we gave when describing the grammar of of processes. Note that the rule NEW adds a name k restricted in $\mathsf{new}_A\, k.P$ to the set $\mathscr{E}$ only if A is honest. If A is dishonest, k becomes available to the attacker (since $k \notin \mathscr{E}$, it can be freely used in recipes by the attacker). Also note that the rule ASSIGN augments μ by adding x to the set $\mu(\tau)$ of variables of type τ.

Example 4. *Consider the process Q_A of Example 3 representing the role of the initiator in the Needham-Shroeder protocol. Recall that x_{skA} and x_{pkB} where free in Q_A. Assume that Q_B is a process representing the role of the receiver with x_{skB} and x_{pkA} as free variables. The following process*

$$(\mathscr{E};P \mid \mathsf{out}_A(c,u).Q;\Phi;\sigma;\mu) \to (\mathscr{E};P \mid Q;\Phi\cdot\{w\to u\sigma\};\sigma;\mu) \quad \text{(OUT)}$$
$$\text{if } w \text{ is fresh, } Msg(c\sigma),\ Msg(u\sigma) \text{ and } \exists M \in \mathsf{Recipe}(\mathscr{E},\Phi).M\Phi{\downarrow} = c\sigma{\downarrow}$$

$$(\mathscr{E};P \mid \mathsf{in}_A(c,x).Q;\Phi;\sigma;\mu) \to (\mathscr{E};P \mid Q;\Phi;\sigma\cdot\{x\to N\Phi\sigma\};\mu) \quad \text{(IN)}$$
$$\text{if } \exists M,N \in \mathsf{Recipe}(\mathscr{E},\Phi) \text{ s.t. } M\Phi{\downarrow} = c\sigma{\downarrow},\ Msg(c\sigma)$$

$$(\mathscr{E};P \mid \mathsf{in}_A(c,x).Q \mid \mathsf{out}_B(d,u).R;\Phi;\sigma;\mu) \to (\mathscr{E};P \mid Q \mid R;\Phi;\sigma\cdot\{x\to u\sigma\};\mu) \quad \text{(COMM)}$$
$$\text{if } Msg(c\sigma),\ Msg(d\sigma),\ Msg(u\sigma) \text{ and } c\sigma{\downarrow} = d\sigma{\downarrow}$$

$$(\mathscr{E};P \mid \mathsf{if}\ u=v\ \mathsf{then}\ Q\ \mathsf{else}\ R;\Phi;\sigma;\mu) \to (\mathscr{E};P \mid Q;\Phi;\sigma;\mu) \quad \text{if } u\sigma{\downarrow} = v\sigma{\downarrow},\ Msg(u\sigma),\ Msg(v\sigma) \quad \text{(THEN)}$$

$$(\mathscr{E};P \mid \mathsf{if}\ u=v\ \mathsf{then}\ Q\ \mathsf{else}\ R;\Phi;\sigma;\mu) \to (\mathscr{E};P \mid R;\Phi;\sigma;\mu) \quad \text{if } u\sigma{\downarrow} \neq v\sigma{\downarrow} \text{ or } \neg Msg(u\sigma) \text{ or } \neg Msg(v\sigma) \quad \text{(ELSE)}$$

$$(\mathscr{E};P \mid !^i Q;\Phi;\sigma;\mu) \to (\mathscr{E};P \mid !^i Q \mid Q\{i\to n\}\rho;\Phi;\sigma;\mu) \quad \text{(REPL)}$$
$$\text{if } n\in\mathbb{N} \text{ and } \rho \text{ is a fresh renaming of bound names and variables of } Q$$

$$(\mathscr{E};P \mid \mathsf{new}_A\, k.Q;\Phi;\sigma;\mu) \to (\mathscr{E}';P \mid Q;\Phi;\sigma;\mu) \quad \text{where } \mathscr{E}' = \mathscr{E}\cup\{k\} \text{ if } A\in\overline{\mathscr{A}}_H \text{ else } \mathscr{E}' = \mathscr{E} \quad \text{(NEW)}$$

$$(\mathscr{E};P \mid \mathsf{agt}(X,S).Q;\Phi;\sigma;\mu) \to (\mathscr{E};P \mid Q\{X\to A\};\Phi;\sigma;\mu) \quad \text{if } A\in S \quad \text{(AGENT)}$$

$$(\mathscr{E};P \mid [x :=_\tau u].Q;\Phi;\sigma;\mu) \to (\mathscr{E};P \mid Q;\Phi;\sigma\cdot\{x\to u\sigma\};\mu') \quad \text{(ASSIGN)}$$
$$\text{if } Msg(u\sigma),\ \mu'(\tau') = \mu(\tau') \text{ for } \tau'\neq\tau \text{ and } \mu'(\tau) = \mu(\tau)\cup\{x\}$$

Figure 3: Semantics of configuration

models an unbounded number of sessions of the Needham-Schoeder protocol between honest or dishonest agents where the public keys and private keys are ideally distributed.

$$!^i\,\mathsf{agt}(A,\{H[i],D[i]\}).!^j\,\mathsf{agt}(B,\{H[j],D[j]\}).(Q_A\{^{sk[A]}/_{x_{skA}},{}^{\mathsf{pk}(sk[B])}/_{x_{pkB}}\} \mid Q_B\{^{sk[B]}/_{x_{skB}},{}^{\mathsf{pk}(sk[A])}/_{x_{pkA}}\})$$

where $H\in\mathscr{A}_H$ and $D\in\mathscr{A}_D$.

3.4. Logical formulas

To express security property, we introduce a first order logic on configurations. It will be particularly convenient to specify the properties expected from a "good" PKI. We consider the following atomic formula.

- $u=v$ and $u\neq v$ where u,v are terms
- $x\stackrel{\circ}{=}y$ and $x\stackrel{\circ}{\neq}y$ where x,y are term variables
- $\tau_1=\tau_2$ where τ_1,τ_2 are parametrized types
- $\not\vdash x$ where x is a term variable

A valuation is a configuration $\mathscr{K}=(\mathscr{E};P;\Phi;\sigma;\mu)$. The satisfaction relation $\models_c$ of atomic formulas is defined as

$$\begin{array}{lll}
\mathscr{K}\models_c x\stackrel{\circ}{=}y & \text{iff} & x=y\\
\mathscr{K}\models_c u=v & \text{iff} & u\sigma{\downarrow}=v\sigma{\downarrow}\\
\mathscr{K}\models_c \not\vdash x & \text{iff} & \forall M\in\mathsf{Recipe}(\mathscr{E},\Phi).Msg(M\Phi)\\
 & & \quad\text{implies } M\Phi{\downarrow}\neq x\sigma{\downarrow}\\
\mathscr{K}\models_c \tau_1=_{\mathscr{A}}\tau_2 & \text{iff} & \exists\gamma_1,\gamma_2\in T.\exists A_1,\ldots,A_n\in\overline{\mathscr{A}}.\\
 & & \tau_1=\gamma_1[A_1,\ldots,A_n]\wedge\tau_2=\gamma_2[A_1,\ldots,A_n]
\end{array}$$

and is lifted as usual to logic formulas with boolean connectors $\wedge,\vee$ and universal and existential quantification of parametrized agent and type variables. Moreover, we consider universal quantification of term variables over parametrized type: $\forall x\in\tau.\phi$ with $x\in\mathscr{X}_t$, $\tau\in\overline{T}$. Its satisfaction relation is defined as: $\mathscr{K}\models_c\forall x\in\tau.\phi$ iff $\forall x\in\mu(\tau),\mathscr{K}\models_c\phi$. Similarly, we also consider existential quantification of term variables.

We say that a process P satisfies a formula ϕ, denoted $P\models\phi$, if ϕ holds in an accessible configuration, that is, $\mathscr{K}\models_c\phi$ for any configuration $\mathscr{K}$ such that $P\to^*\mathscr{K}$.

Example 5. *Consider a type $\mathbb{sk}$ and a process $!^i\,\mathsf{agt}(X,\{H[i],D[i]\}).P$ where P contains a single assignment variable $[x:=_{\mathbb{sk}(X)}u]$. The formula*

$$\forall A\in\overline{\mathscr{A}}_H.\forall y,z\in\mathbb{sk}(A).y=z$$

expresses that any two sessions of some honest agent always assign x to the same term. Note that the formula does not say anything about sessions of dishonest agents.

Secrecy. To model secrecy preservation, we consider an additional type $\mathbb{secret}$, yielding a set of types T that contains at least $\{\mathbb{sk},\mathbb{pk},\mathbb{sig},\mathbb{vk},\mathbb{secret}\}$. We assume that variables that should remain confidential are assigned the type $\mathbb{secret}$. Then secrecy can be generically defined by the following formula ϕ_{sec}.

$$\forall\tau\in\overline{\mathbb{secret}}_H.\forall x\in\tau.\not\vdash x$$

This formula states that any variable of type $\mathbb{secret}_A$ for A honest should not be deducible (for any of its instantiations).

Example 6. *Continuing Example 3, we can require secrecy of the nonce n_a generated by A and the nonce n_b as received by A by simply modifying process Q_A as follows.*

$$\begin{array}{l}
\mathsf{new}_A\,n_a.[z:=_{\mathbb{secret}[A,B]}n_a].\\
\quad\mathsf{out}_A(c,\mathsf{aenc}(\langle\mathsf{pk}(x_{skA}),n_a\rangle,y_{pkB})).\mathsf{in}_{-}(c,y)\\
\quad\mathsf{if}\ n_a=\mathsf{proj}_1(\mathsf{proj}_2(\mathsf{adec}(y,x_{skA})))\ \mathsf{then}\\
\quad\mathsf{if}\ y_{pkB}=\mathsf{proj}_1(\mathsf{adec}(y,x_{skA}))\ \mathsf{then}\\
\quad[z':=_{\mathbb{secret}[A,B]}\mathsf{proj}_2(\mathsf{proj}_2(\mathsf{adec}(y,x_{skA})))].\\
\quad\mathsf{out}_A(c,\mathsf{aenc}(\mathsf{proj}_2(\mathsf{proj}_2(\mathsf{adec}(y,x_{skA}))),y_{pkB}))
\end{array}$$

Authentication. In the literature, authentication properties are usually modeled using events. In our formalism, variables assignments can play such a role. Consider for example two types $\mathbb{ev}_1$ and $\mathbb{ev}_2$ contained in T. The authentication property modeling a correspondence between the two types can be defined by the following formula ϕ_{auth}.

$$\forall\tau\in\overline{\mathbb{ev}_1}_H.\forall x\in\tau.\exists\tau'\in\overline{\mathbb{ev}_2}_H.\exists y\in\tau'.\tau=_{\mathscr{A}}\tau'\wedge x=y$$

Informally, the formula indicates that whenever a variable x of type $\mathbb{ev}_1[A_1,\ldots,A_n]$ is assigned a term with honest agents $A_1,\ldots,A_n$ then there exists a variable y of type $\mathbb{ev}_2[A_1,\ldots,A_n]$ must have been assigned previously with the same term. We could also consider injective authentication by further requiring the variable y to be unique, which can be expressed by $\forall z \in \tau'.y \neq z \vee y \mathrel{\mathring{=}} z$.

Example 7. *Continuing Example 3, authentication can be expressed through ϕ_{auth} and adding in Q_A the assignment $[x :=_{\mathbb{ev}_2[A,B]} \langle n_A, \mathsf{proj}_2(\mathsf{proj}_2(\mathsf{adec}(y, x_{skA})))\rangle]$ and similarly in Q_B but with the type $\mathbb{ev}_1$.*

Composable properties. In this paper, we will show that our composition result preserves any *composable property*, that is, a closed formula from our logic where quantification of agents are over honest agents, i.e. of the form $\forall A \in S$ and $\exists A \in S$ with $S \subseteq \overline{\mathscr{A}}_H$, and where any atomic formula $u = v$ and $u \neq v$ involves only variables ($u, v \in \mathscr{X}$).

4. Composition hypotheses

In this section we formalize the hypothesis that underlie our composition theorem. Since the development is rather technical we include here a small roadmap for this section. We begin (Section 4.1) with formalizing the guarantees that we view as minimal for any PKI as sketched in Section 2.1. Next, in Section 4.2, we formalize the composition of an arbitrary PKI protocol P with an arbitrary other protocol Q. Most of the development here consists of syntactical restrictions which essentially force that the composition between protocol P and Q.

The rest of the section deals with more subtle interactions between P and Q. In Section 4.3 we explain how to deal with private keys: they should only be used as key material in cryptographic algorithms and, if sent as payload, they should be tagged. We discuss in Sections 4.3 and 4.4 two distinct approaches to ensure that the use of common primitives does not lead to unwanted interference between the two composed components. Our theorem can employ either of the two approaches.

We discuss each of the more subtle hypothesis that we require through the prism of an example that motivates it. For simplicity, both the discussions and the formalism that we develop in this paper consider the case of two party protocols; our results can be lifted to n-party protocols.

4.1. PKI properties

We consider PKIs that establish keys both for encryption and signatures. We model these types of keys through a set T of types for assignment variables that includes the types $\mathbb{sk}, \mathbb{pk}, \mathbb{sig}$ and $\mathbb{vk}$ respectively for asymmetric private keys, asymmetric public keys, signing keys, and verification keys. Agents do not necessarily share the same values for keys, in particular, an agent A may think that C's public key is $\mathsf{pk}C$ while B believes that C's public key is $\mathsf{pk}C'$. We do not want to discard this possibility (since as discussed in Section 2.5, this is a possible attack against a PKI). We model this possibility using our notion of parametrized types. Specifically, we consider types of the form $\mathbb{sk}[A]$, $\mathbb{pk}[A,B]$, $\mathbb{sig}[A]$ and $\mathbb{vk}[A,B]$ where $\mathbb{pk}[A,B]$ (resp. $\mathbb{vk}[A,B]$) represents the asymmetric public (resp. verification) key of B as viewed by A.

As stated in Section 2.1, we informally demand that a PKI satisfies the following properties.

- An honest agent has a unique public/private key pair and a unique verification/signing key pair.
- Honest agents of course have pairwise distinct private/signing keys.
- Keys are consistently distributed, that is, honest agents know each other's public and verification keys.
- Private/signing keys of honest agents are indeed private.

For asymmetric encryption keys, these properties are captured through the formula ϕ_{asy} below. Each line corresponds to a bullet above.

$$\begin{aligned}\phi_{asy} \mathrel{\hat{=}}\ & \forall A,B \in \overline{\mathscr{A}}_H.\forall x,y \in \mathbb{sk}[A].x = y \\ & \wedge \forall x \in \mathbb{sk}[A].\forall y \in \mathbb{sk}[B].A = B \vee x \neq y) \\ & \wedge \forall x \in \mathbb{sk}[A].\forall y \in \mathbb{pk}[B,A].\mathsf{pk}(x) = y) \\ & \wedge \forall x \in \mathbb{sk}[A].\not\forall x)\end{aligned}$$

We model the analogous property of a good PKI w.r.t. signing/verification keys with a formula ϕ_{sig} obtained from ϕ_{asy} by replacing $\mathbb{sk}$, $\mathbb{pk}$ and pk respectively by $\mathbb{sig}$, $\mathbb{vk}$ and vk.

Finally, the overall guarantee that a PKI should offer are the two properties above together with the requirement that the keys used for signing/verification are different from those used for encryption/decryption:

$$\phi_{\mathsf{PKI}} \mathrel{\hat{=}} \phi_{asy} \wedge \phi_{sig} \wedge \forall A,B \in \overline{\mathscr{A}}_H.\forall x \in \mathbb{sk}[A].\forall y \in \mathbb{sig}[B].x \neq y$$

The last part of ϕ_{PKI} indicates that signing keys and private asymmetric keys should be pairwise distinct.

Example 8. *Consider the PKI protocol modeled by the process P in Example 2. The protocol satisfies our security requirement: $C[P[0,0]] \models \phi_{\mathsf{PKI}}$ where $C[\_]$ is the context $!^i\,\mathsf{agt}(A,\{H[i],D[i]\}).!^j\,\mathsf{agt}(B,\{H[j],D[j]\}).\_$ that declares the agents.*

4.2. Composition Setup

In the previous section we introduced the security guarantee which we require from protocol P (when analyzed in isolation). From this point onwards we consider the interaction between P and Q. We start by defining the composition between a PKI protocol P and an arbitrary protocol Q. Formally, we first consider a process $P[\__A,\__B]$ representing a PKI protocol that establishes long term keys for two agents A and B. Second, we consider two processes Q_A and Q_B modeling the roles of a two-agents protocol Q in which keys are assumed to be already distributed. Our goal is to identify on which conditions on P and Q their combination remains secure. Formally, the combination of Q using the PKI P is expressed by the following process.

$$!^i\,\mathsf{agt}(A,\{H[i],D[i]\}).!^j\,\mathsf{agt}(B,\{H[j],D[j]\}).P[Q_A,Q_B]$$

This process models an unbounded number of sessions between honest and dishonest agents.

Since we consider two-agent protocols, we assume w.l.o.g. the following properties on P and Q.

H$_1$. $x_{skA}, y_{pkB}, x_{sigA}, y_{vkB}$ are the only possible free variables of Q_A, Q_A is a role of A and the hole $\_A$ in P is in the scope of $[x_{skA} :=_{\mathtt{sk}[A]} u]$, $[y_{pkB} :=_{\mathtt{pk}[A,B]} v]$, $[x_{sigA} :=_{\mathtt{sig}[A]} w]$, $[y_{vkB} :=_{vk[A,B]} r]$ for some u, v, w, r. We require the analogous hypothesis for Q_B and $\_B$. Moreover, the sets of bound names and free names in P and Q are distinct.

This hypothesis simply formalises the setting and ensures that the process $P[Q_A, Q_B]$ avoids name clashes and is closed, meaning that the free variables of Q_A and Q_B will be instantiated.

Moreover, we demand that the only shared keys are those that the PKI protocol P generates and passes to Q. Both protocols may generate other keys, but these cannot be shared. In particular, P and Q may not share long term keys.

H$_2$. for all $n[A_1, \ldots, A_p] \in pn(P[\_A, \_B])$, for all $m[B_1, \ldots, B_q] \in pn(Q_1, Q_2)$, $n \neq m$.

4.3. Tagging

As discussed in Section 2, a PKI infrastructure and a protocol Q do not immediately yield a secure composition. We first need to get rid of the behaviors explained in Section 2.2 where the PKI infrastructure P intereferes with a protocol Q as they use the same primitives and the same keys.

Similarly to the approach of Arapinis, Cheval, and Delaune [3] we consider a setting where P and Q may use arbitrary primitives, except for the shared ones, that should be the standard primitives. Formally, we consider the following common signature $\mathscr{F}^0 = \mathscr{F}^0_c \uplus \mathscr{F}^0_d$ where $\mathscr{F}^0_d = \{\mathsf{sdec}/2, \mathsf{rsdec}/2, \mathsf{adec}/2, \mathsf{radec}/3, \mathsf{check}/2, \mathsf{proj}_1/1, \mathsf{proj}_2/1\}$ and $\mathscr{F}^0_c = \{\mathsf{senc}/2, \mathsf{rsenc}/3, \mathsf{aenc}/2, \mathsf{raenc}/3, \mathsf{pk}/1, \mathsf{sign}/2, \mathsf{vk}/1, \langle\rangle/2, \mathsf{h}/1\}$. The associated rewriting system $\mathscr{R}^0$ has been defined in Example 1.

We also consider two disjoints signatures for P and Q, namely $\mathscr{F}^P, \mathscr{F}^Q$, as well as their associated rewriting systems $\mathscr{R}^P, \mathscr{R}^Q$ and equational theories $\mathrm{E}^P, \mathrm{E}^Q$.

4.3.1. Tagging of private keys. We first need to guarantee that Q does not manipulare the structure of private keys, to avoid the "related keys" example of Section 2.6. Such related keys will be tolerated under the condition that Q never "opens" a private keys nor sends them as payload unless they are tagged (in principle, private keys should not be sent in payload anyway). Formally, assume that $\mathscr{F}^Q$ contains two function symbols tagk and untagk such that $\mathsf{untagk}(\mathsf{tagk}(x)) \to x \in \mathscr{R}^Q$ and $\mathsf{tagk}, \mathsf{untagk}$ do not appear in any other rewrite rules in $\mathscr{R}^Q$ or in E^Q. The next hypothesis states how private keys are tagged when used as payload:

H$_3$. Process $P[\_A, \_B]$ is built over $\mathscr{F}^P \cup \mathscr{F}^0$, process Q_A, Q_B are built over $\mathscr{F}^Q \cup \mathscr{F}^0 \setminus \{\mathsf{untagk}\}$ and in Q_A and Q_B, the private and signing keys provided by the PKI can only be used in key position with $\mathsf{adec}, \mathsf{radec}, \mathsf{sign}$ respectively or as the argument of tagk.

4.3.2. Tagging processes. As illustrated in Section 2.2, primitives shared between P and Q should be tagged. For instance, tagging an encryption $\mathsf{senc}(u,k)$ may be done by encrypting u alongside some constant t, *i.e.* $\mathsf{senc}(\langle \mathsf{t}, u\rangle, k)$. As for the tags on private keys, we do not wish to specify exactly how tags are implemented. Therefore, for all $i \in \{P,Q\}$, we assume the existence of two function symbols tag_i and untag_i such that $\mathsf{untag}_i(\mathsf{tag}_i(x)) \to x \in \mathscr{R}^Q$ and $\mathsf{tag}_i, \mathsf{untag}_i$ do not appear in any other rewrite rules in $\mathscr{R}^i$ or in E^i.

Definition 2 (Tagged terms and processes)**.** *Let $i \in \{P,Q\}$. We define the set of i-*tagged terms*, denoted* $\mathrm{TAGT}(i)$*, as the smallest set of term built on $\mathscr{F}^i \cup \mathscr{F}^0$ such that for all $u_1, \ldots, u_n \in \mathrm{TAGT}(i)$, for all $\mathsf{f}/n \in \mathscr{F}^i \cup \mathscr{F}^0$,*

- $u \in \mathrm{TAGT}(i)$ *if* $u \in \mathscr{N} \cup \overline{\mathscr{N}} \cup \mathscr{X}_t$
- $\mathsf{untag}_i(\mathsf{f}(u_1, \ldots, u_n)) \in \mathrm{TAGT}(i)$ *if* $\mathsf{f} \in \{\mathsf{sdec}, \mathsf{adec}, \mathsf{rsdec}, \mathsf{radec}, \mathsf{check}\}$
- $\mathsf{f}(\mathsf{tag}_i(u_1), u_2, \ldots, u_n) \in \mathrm{TAGT}(i)$ *if* $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{rsenc}, \mathsf{raenc}, \mathsf{sign}\}$
- $\mathsf{f}(u_1, \ldots, u_n) \in \mathrm{TAGT}(i)$ *when* $\mathsf{f} \in \{\mathsf{h}, \langle\ \rangle, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{vk}, \mathsf{pk}\} \cup \mathscr{F}^i \setminus \{\mathsf{untag}_i\}$

*A process is said to be i-tagged if for all terms u contained in an action of the process (*i.e. *input, output, conditional and variable assignment), $u \in \mathrm{TAGT}(i)$.*

We can now state our condition for tagged processes:

C$_1$. $P[\_A, \_B]$ is P-tagged and Q_A, Q_B are Q-tagged

4.4. Disjoint public keys

Considering tagged protocols P and Q is an efficient way to ensure that honest agents do not confuse messages from P with messages from Q. While simple, such a tagging assumption is rarely met in practice. Even simple protocols such as the Needham-Schroeder-Love protocol would not be covered by our composition result. We propose here an alternative assumption that is best explained going back to the example described in Section 2.2 where the executions of P and Q may interfere with each other. To avoid such interferences, we need to make sure that the set of keys that are used for encryption and signatures in P is disjoint from the set of keys used in Q. Such a condition may however be difficult to check. We therefore formulate a stronger assumption, usually met in practice.

Specifically, we assume that the only public/private keys *dynamically* generated by P and Q are those passed from P to Q. This assumption is often met in practice, e.g. for our NSL example. This is informally stated as follows.

C$_2$. All terms in $P[\_A, \_B]$ used in key positions w.r.t. $\mathscr{F}^0$ are ground, that is, P only uses keys known in advance.

C$_3$. All terms in Q_A, Q_B used in key positions w.r.t. $\mathscr{F}^0$ are either ground and disjoint from the terms in $P[\_A, \_B]$ used in key positions w.r.t. $\mathscr{F}^0$ or in $\{x_{skA}, y_{pkB}, x_{sigA}, y_{vkB}, x_{skB}, y_{pkA}, x_{sigB}, y_{vkA}\}$. Similarly, Q uses only known keys or keys from P.

Note that these assumptions only restrict keys used in the common signatures. P and Q may of course freely create keys provided they use a different encryption/signing scheme.

It then remains to ensure that the keys established by P (and therefore shared with Q) are distinct from the other keys used by P. This property can be expressed in our logic by considering a special type $\mathtt{test} \in T$, and a process P' obtained from P by adding assignments of the form $[x :=_{\mathtt{test}} t]$ where x a fresh variable, for any t appearing in P as key position w.r.t. the common signature $\mathscr{F}^0$. None of these shared terms should collide with the PKI keys established by P.

C_4. $P' \models \forall x \in \mathtt{test}.\forall y \in \overline{\mathtt{sk}}_H.\forall z \in \overline{\mathtt{sig}}_H.x \neq y \wedge x \neq z \wedge x \neq \mathsf{pk}(y) \wedge x \neq \mathsf{vk}(z)$.

Example 9. *Continuing Example 2, the only keys used in key position are $sig[S]$ (in the role of the server) and $\mathsf{vk}(sig[S])$ (when A and B check the certificates). Since none of these keys is freshly bound, the process $P'[\__A,\__B]$ can simply be the process $P[\__A,\__B] \mid [z_1 :=_{\mathtt{test}} sig[S]] \mid [z_2 :=_{\mathtt{test}} \mathsf{vk}(sig[S])]$. Moreover, it is easy to show that no honest agent can be assigned the public key nor the verification key of the server, meaning that $P[\__A,\__B]$ satisfies the condition C_4.*

As previously mentioned, to ensure secure composition, we need either our protocols to verify the disjoint-keys hypotheses or the tagging hypotheses. Therefore, we can state the following hypothesis that gathers both cases:

H_4. either condition C_1 holds or conditions C_2, C_3 and C_4 hold.

5. Composition results

The previous section lists necessary assumptions to avoid interferences between a PKI protocol P and a subsequent protocol Q. We are therefore ready to state our result: if Q is secure and if P is a good PKI then Q may securely use P for the establishment of its keys. In fact, we note that such a composition result (still) does not hold in general since a good PKI provides weaker guarantees than an ideal distribution of the keys as usually assumed in the analysis of Q as exemplified in Section 2. We therefore need to introduce a more *permissive* Q which is the final ingredient to our main composition result.

5.1. Permissive Q

Our "confusing material" example in Section 2.3 shows that Q should be analysed without assuming dishonest keys to be honestly generated and distributed. Instead, a *permissive* Q should be considered, where dishonest keys are simply provided by the attacker. The goal of this section is to formally define permissive Q. We assume $d \in \mathcal{N}$ to be a fresh public channel (that is, a name not used elsewhere) and that the names *sk* and *sig* do not occur in Q_A and Q_B.

Recall that $Q = Q_A \mid Q_B$ and $x_{skA}, y_{pkB}, \ldots$ are the only possible free variables of Q (cf H_1). The ideal instantiation of the private variables and public variables of an agent A are $\sigma_A^{\mathsf{priv}} = \{{}^{sk[A]}/_{x_{skA}}, {}^{sig[A]}/_{x_{sigA}}\}$ and $\sigma_B^{\mathsf{pub}} = \{{}^{\mathsf{pk}(sk[B])}/_{y_{pkB}}, {}^{\mathsf{vk}(sig[B])}/_{y_{vkB}}\}$ respectively. Similarly, we define σ_B^{priv} and σ_A^{pub}. However, if A is dishonest, he should be able to chose his public and verification keys freely, i.e., they are under the control of the attacker. Formally, we define

- $I_A^{\mathsf{pub}}[\_] = \mathsf{in}_B(d, y_{pkA}).\mathsf{in}_B(d, y_{vkA}).\_$
- $I_B^{\mathsf{pub}}[\_] = \mathsf{in}_A(d, y_{pkB}).\mathsf{in}_A(d, y_{vkB}).\_$

In the previous section, we stated all the hypotheses that we rely on to ensure a secure composition between the PKI P and the protocol Q. For our first main result, we consider that a *permissive* Q satisfies the secrecy property.

Permissive Q is the protocol Q where honest keys are ideally distributed (and private keys remain private) while dishonest ones are under the control of the attacker.

Definition 3 (Permissive Q)**.** *Let $Q = Q_A \mid Q_B$ be a process satisfying assumptions H_1 and H_2. We define* permissive Q*, denoted Q_{perm}, as the following process:*

$$!^i\,\mathsf{agt}(A,\{H[i]\}).!^j\,\mathsf{agt}(B,\{H[j]\}).\\ (O_A[Q_A]\sigma_A^{\mathsf{priv}}\sigma_B^{\mathsf{pub}} \mid O_B[Q_B]\sigma_B^{\mathsf{priv}}\sigma_A^{\mathsf{pub}})\\ \mid !^i\,\mathsf{agt}(A,\{D[i]\}).!^j\,\mathsf{agt}(B,\{H[j]\}).I_A^{\mathsf{pub}}[O_B[Q_B]]\sigma_B^{\mathsf{priv}}\\ \mid !^i\,\mathsf{agt}(A,\{H[i]\}).!^j\,\mathsf{agt}(B,\{D[j]\}).I_B^{\mathsf{pub}}[O_A[Q_A]]\sigma_A^{\mathsf{priv}}$$

where $O_A[\_] = \mathsf{out}_A(d, \langle \mathsf{pk}(x_{skA}), y_{pkB}, \mathsf{vk}(x_{sigA}), y_{vkB}\rangle).[z_1 :=_{\mathtt{secret}[A]} x_{skA}].[z_2 :=_{\mathtt{secret}[A]} x_{sigA}].\_$ *with z_1, z_2 fresh and similarly for $O_B[\_]$.*

The process $O_A[\_]$ simply outputs the public keys of A and B as viewed by A and indicates that the private keys of A should stay secret, and similarly for $O_B[\_]$. The first part of Q_{perm} corresponds to sessions between honest agents, where all keys are ideally distributed while the second (resp. third) part of Q_{perm} corresponds to sessions between an honest B (resp. A) and a dishonest A (resp. B).

If permissive Q is secure, then Q can safely be composed with a good PKI.

Theorem 1. *Let $P[\__A,\__B]$ be a context process and $Q = Q_A \mid Q_B$ be a process such that P and Q satisfy hypotheses H_1 to H_4. Let ϕ be a composable property.*

If the following conditions are satisfied

- $!^i\,\mathsf{agt}(A,\{H[i],D[i]\}).!^j\,\mathsf{agt}(B,\{H[j],D[j]\}).P[O_A,O_B] \models \phi_{\mathsf{PKI}}$ *(that is, P is a secure PKI)*
- $Q_{perm} \models \phi_{sec} \wedge \phi$ *(that is, Q is a secure protocol)*

then $P[Q_A,Q_B]$ is secure, that is

$$!^i\,\mathsf{agt}(A,\{H[i],D[i]\}).!^j\,\mathsf{agt}(B,\{H[j],D[j]\}).P[Q_A,Q_B] \models \phi$$

where $\phi_{sec} \triangleq \forall \tau \in \overline{\mathtt{secret}}_H.\forall x \in \tau.\not\vdash x$.

Sketch of proof. A complete version of the proof can be found in [1]. The first step of the proof is quite standard w.r.t. existing composition results. It is well known that reachability properties compose well when processes do no share any secret. Hence from our hypothesis H_2, we obtain that $C[P[O_A,O_B]] \mid Q_{perm} \models \phi_{sec}$ where $C[\_] = !^i\,\mathsf{agt}(A,\{H[i],D[i]\}).!^j\,\mathsf{agt}(B,\{H[j],D[j]\}).\_$. From there, we reason by contradiction. If $C[P[Q_A,Q_B]] \not\models \phi_{sec}$ then it is easy to see that $C[P[O_A[Q_A],O_B[Q_B]]] \not\models \phi_{sec}$ since O_A and

O_B are just outputs. Applying the same reasoning with fresh inputs, we can build two processes R_{real} and R_{perm} such that

- $C[P[Q_A, Q_B]] \not\models \phi_{sec}$ implies $R_{real} \not\models \phi_{sec}$
- $C[P[O_A, O_B]] \mid Q_{perm} \models \phi_{sec}$ implies $R_{perm} \models \phi_{sec}$

Intuitively, R_{real} and R_{perm} only differ by the messages in the process. For example, any occurrence of a variable x_{skA} of an honest agent A in R_{real} corresponds to an occurrence of $sk[A]$ at the same position in R_{perm}. The key part of the proof (and the main difficulty) consists in showing that $R_{real} \not\models \phi_{sec}$ in fact implies $R_{perm} \not\models \phi_{sec}$. For this, we consider a transformation δ on terms whose purpose is to dynamically replace the occurrences of the instantiation of x_{skA}, x_{sigB}, etc in a configuration of R_{real} by their counterpart in R_{perm}. In particular we show that

- for any configuration $R_{real} \to^* (\mathscr{E}; P; \Phi; \sigma; \mu)$, we have $R_{perm} \to^* (\mathscr{E}; P; \delta(\Phi); \delta(\sigma); \delta(\mu))$
- $(\mathscr{E}; P; \Phi; \sigma; \mu) \models_c \vdash u$ implies $(\mathscr{E}; P; \delta(\Phi); \delta(\sigma); \delta(\mu)) \models_c \vdash \delta(u)$.

This proof technique is similar to the work in [20], [19], [3]. However, we improve over this previous work by relaxing the tagging condition. Interestingly, our core result is general enough to captures both the tagging and the disjoint key conditions in a single result. Moreover, we no longer assume restriction on the form of the shared keys (signing, verification, private and public keys), in contrast to e.g. [3] where private keys are assumed to be nonces. □

Note that we require $P[O_A, O_B]$ to satisfy ϕ_{PKI} and not just P. This is because we need to make sure that P remains a secure PKI even when the public keys are indeed public.

Interestingly, the permissive version of a protocol can easily be encoded and analysed in ProVerif. This is the first lesson learned from our work: if you wish to analyze a protocol Q independently of the underlying PKI, you should analyze permissive Q instead of the ideal (standard) Q. As we shall see in the next section, it may be sufficient to analyse the ideal version of Q, at the price of additional assumptions on either P or Q.

5.2. Composition with an "ideal Q"

As far as we know, in symbolic models protocols are never analyzed in their "permissive" version. Instead, all existing libraries consider all keys to be properly generated and distributed, including those of dishonest parties. We will say that libraries consider *ideal* protocols. As illustrated by our "confusing material" example in Section 2.3, such ideal protocols *are* indeed too abstract and may be flawed when used in conjunction with a true PKI. So a natural question arrises: what about the hundreds of protocols that have already been analyzed? Should all these analysis start over? In this section, we study under which conditions it is sufficient to analyze an ideal protocol Q. Clearly, secure composition requires to a corresponding strenghtening of the guarantees of the PKI.

We first define formally "ideal Q". It consists of the protocol Q where all keys are ideally distributed.

Definition 4 (Ideal Q)**.** *Let $Q = Q_A \mid Q_B$ be a process satisfying assumptions H_1 and H_2. We define* ideal Q*, denoted Q_{ideal}, as the following process:*

$$!^i\,\mathsf{agt}(A, \{H[i], D[i]\}).!^j\,\mathsf{agt}(B, \{H[j], D[j]\}).\\ (O_A[Q_A]\sigma_A^{\mathsf{priv}}\sigma_B^{\mathsf{pub}} \mid O_B[Q_B]\sigma_B^{\mathsf{priv}}\sigma_A^{\mathsf{pub}})$$

where $O_A[\_], O_B[\_], \sigma_A^{\mathsf{priv}}, \sigma_B^{\mathsf{priv}}\sigma_A^{\mathsf{pub}}\sigma_B^{\mathsf{pub}}$ have been defined in Section 5.1.

Process Q_A is instantiated by the expected private keys σ_A^{priv} and public keys σ_B^{pub} of B and similarly for Q_B.

5.2.1. Tagged public keys. As illustrated by our "confusing material" example in Section 2.3, when public keys are used as payload, they may interfere with other parts of the protocol. To avoid such intereferences, we need public keys to be "isolated". So, similarly to the case of private keys (Assumption H_9), we now require that public keys used as payload are isolated within a tag. We further require that only PKI keys may be used for asymmetric encryption/decryption in Q. This is more formally stated as follows.

H_5. In the processes Q_A and Q_B, the public and verification keys provided by the PKI can only be used in key position with $\mathsf{aenc}, \mathsf{raenc}, \mathsf{check}$ respectively or below a tag tagk. Moreover, only the private, public, signing, verification keys provided by the PKI can be used in key position with the common signature or below a tag tagk.

Such an assumption is trivially satisfied when public keys are not used as payload but only for encryption. However, a lesson learned from our analysis is that for protocols that use public key as payloads then either permissive Q should be analyzed or tagging public keys is necessary.

5.2.2. Ideal PKI. Even if public keys are properly used in Q, the attacker can control dishonest public keys and interferes with Q's behavior, as exemplified in Section 2.5 where we show unexpected behaviors if honest agents do not share the same view of dishonest keys. Therefore, we consider an additional property which ensures that public and verification keys of dishonest agents are consistently distributed among honest agents. This is formally captured by formula ϕ_{ideal} as follows.

$$\begin{aligned}\phi_{ideal} \hat{=}\ & \forall A, B \in \overline{\mathscr{A}}_H.\forall C, D \in \overline{\mathscr{A}}.\\ & \forall x \in \mathbb{pk}[A, C].\forall y \in \mathbb{pk}[B, D].C = D \Leftrightarrow x = y\\ & \wedge\ \forall x \in \mathbb{vk}[A, C].\forall y \in \mathbb{vk}[B, D].C = D \Leftrightarrow x = y\\ & \wedge\ \forall x \in \mathbb{pk}[A, C].\forall y \in \mathbb{vk}[A, C].x \neq y\end{aligned}$$

The first line ensures that all agents share the same public key for a given agent C and that conversely, public keys of distinct agents are pairwise distinct. The second lines states the same property for verification keys. Finally, public and verification keys should of course be distinct.

Such strong guarantees are typically met when public keys are issued by a (trusted) authority, for example a governmental agency that issues electronic IDs.

The next theorem establishes that analysis of the ideal version of a protocol Q still enables secure composition with a PKI protocol P, provided that P satisfies ϕ_{PKI} and ϕ_{ideal}.

Theorem 2. *Let $P[\__A, \__B]$ be a context process and $Q = Q_A \mid Q_B$ be a process such that P and Q satisfy hypotheses H_1 to H_5. Let ϕ be a composable property.*

If the following conditions are satisfied

- $C[P[O_A, O_B]] \models \phi_{\mathsf{PKI}} \wedge \phi_{ideal}$ *(that is, P is an ideal PKI)*
- $Q_{ideal} \models \phi_{sec} \wedge \phi$ *(that is, Q is an ideal secure protocol)*

then $P[Q_A, Q_B]$ is secure, that is $C[P[Q_A, Q_B]] \models \phi$ where $C[\_] = !^i\,\mathsf{agt}(A, \{H[i], D[i]\}).!^j\,\mathsf{agt}(B, \{H[j], D[j]\}).\_$.

Sketch of proof. We first apply the transformation as in Theroem 1, constructing two processes R_{real} and R_{perm} such that $R_{real} \not\models \phi_{sec}$ implies $R_{perm} \not\models \phi_{sec}$. The second step of the proof consists in building a third process R_{ideal} such that $C[P[O_A, O_B]] \mid Q_{ideal} \models \phi_{sec}$ implies $R_{ideal} \models \phi_{sec}$. Intuitively, any occurence of a variable y_{pkA} of a dishonest agent A in R_{perm} corresponds to an occurence of $\mathsf{pk}(sk[A])$ at the same position in R_{ideal}. We conclude by showing that $R_{perm} \not\models \phi_{sec}$ implies $R_{ideal} \not\models \phi_{sec}$. □

6. Conclusion

Standalone analysis of protocols that rely on long term asymmetric keys typically assumes idealized key distribution through some PKI. Yet, this property is not naturally guaranteed by standard PKIs. We have therefore initiated a study of the conditions under which the composition of PKIs (for both asymmetric encryption and digital signatures) with arbitrary protocols that require such keys yields a secure system.

We have shown that this is possible through modular analysis which considers the two protocols separately and requires minimal, easy to implement and verify conditions on how the two components of the composition interact. In short, we have identified several useful recommendations. To deal with the weaker guarantees offered by PKIs, the *permissive* version of the protocol that uses PKI keys should be analyzed rather than its *ideal* version. In addition, to eliminate unwanted interference between the two components of the composition the protocols should not share any keys (beyond those that the PKI distributes). In fact, standards already suggest that this should be the case – our analysis confirms that this guarantee helps guarantee the desired composability between protocols. Finally, we have shown that under some conditions, security analysis of protocols that assumes idealized key distribution is sound if the PKI also guarantees a consistent distribution of dishonest keys.

Even if most our counter-examples to composition are contrived, the lesson learned from our study is that the properties offered by a typical PKI provide less guaranties than what is typically expected by a protocol *using* a PKI. When designing a PKI, it is difficult, if not impossible, to predict how it will be used in subsequent protocols. It is therefore important to strengthen it as much as possible and our property ϕ_{ideal} provides some guidance for this. Conversely, when analyzing a protocol that assumes a PKI, it is tempting to abstract the PKI in a too simple way. We provide here a sound abstraction that can be used in tools like ProVerif.

In our study, we also identified several cases where composition is not secure, and provided examples. Some of these examples are contrived. As future work, we plan to explore whether real case protocols cannot indeed be composed, or alternatively, identify why "realistic" examples do not run into the same issues and formalize the corresponding theorems.

Much of the work set within computational models assumes a PKI that provides keys to parties but is not concerned with the details of how public keys are generated and distributed. Unsurprisingly, our negative results set within symbolic models also hold in such models. A fascinating question is whether in such models, positive results analogous to those we present here are possible. As outlined in the section on related work, existing results make unrealistic assumptions on PKI procedures (i.e. that they use a concurrently secure zero-knowledge proofs of knowledge). In turn, this assumptions can be traced to the simulation-based paradigm which facilitate general composition results. An interesting avenue for future research is to consider other definitional paradigms (e.g. cryptographic games) where less general composition results but with more realistic assumptions on the components are possible [13], [11], [32].

Acknowledgments

This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC).

References

[1] Secure composition of pkis with public key protocols. https://www.dropbox.com/s/hhsqqxqm8sm1gl3/CSF2017-Secure_Compo_PKI-long_version.pdf?dl=0.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.

[3] Myrto Arapinis, Vincent Cheval, and Stéphanie Delaune. Composing security protocols: from confidentiality to privacy. In Riccardo Focardi and Andrew Myers, editors, *Proceedings of the 4th International Conference on Principles of Security and Trust (POST'15)*, Lecture Notes in Computer Science, London, UK, April 2015. Springer Berlin Heidelberg.

[4] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

[5] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, pages 186–195, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. CSFW'01*, 2001.

[7] Florian Bohl and Dominique Unruh. Symbolic universal composability. In *Proceedings of the 2013 IEEE 26th Computer Security Foundations Symposium*, CSF '13, pages 257–271, Washington, DC, USA, 2013. IEEE Computer Society.

[8] Alexandra Boldyreva, Marc Fischlin, Adriana Palacio, and Bogdan Warinschi. A closer look at PKI: security and efficiency. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 458–475. Springer, 2007.

[9] Colin Boyd, Cas Cremers, Michele Feltz, Kenneth G. Paterson, Bertram Poettering, and Douglas Stebila. ASICS: authenticated key exchange security incorporating certification systems. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 381–399. Springer, 2013.

[10] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, 12(4):267–297, 2013.

[11] Christina Brzuska, Marc Fischlin, Nigel P Smart, Bogdan Warinschi, and Stephen C Williams. Less is more: Relaxed yet composable security notions for key exchange. *International Journal of Information Security*, 12(4):267–297, 2013.

[12] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of bellare-rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 51–62. ACM, 2011.

[13] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C Williams. Composability of bellare-rogaway key exchange protocols. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 51–62. ACM, 2011.

[14] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. Universal composition with responsive environments. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 807–840, 2016.

[15] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[16] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. Time-bounded task-pioas: A framework for analyzing security protocols. In Shlomi Dolev, editor, *DISC*, volume 4167 of *Lecture Notes in Computer Science*, pages 238–253. Springer, 2006.

[17] Vincent Cheval, Véronique Cortier, and Eric le Morvan. Secure Refinements of Communication Channels. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*, volume 45 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 575–589, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[18] Céline Chevalier, Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Composition of password-based protocols. *Formal Methods in System Design*, 43(3):369–413, 2013.

[19] Ştefan Ciobâcă and Véronique Cortier. Protocol composition for arbitrary primitives. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 322–336. IEEE Computer Society Press, July 2010.

[20] Véronique Cortier and Stéphanie Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, February 2009.

[21] Cas Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, volume 5123/2008 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.

[22] Stéphanie Delaune, Steve Kremer, and Olivier Pereira. Simulation based security in the applied pi calculus. In Ravi Kannan and K. Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 169–180. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.

[23] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 243–320. Elsevier,, 1990.

[24] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 146–162, Berlin, Heidelberg, 2009. Springer-Verlag.

[25] Thomas Gibson-Robinson, Allaa Kamil, and Gavin Lowe. Verifying layered security protocols. *Journal of Computer Security*, 23(3), 2015.

[26] Dennis Hofheinz, Eike Kiltz, and Victor Shoup. Practical chosen ciphertext secure encryption from factoring. *J. Cryptology*, 26(1):102–118, 2013.

[27] Ralf Kusters. Simulation-based security with inexhaustible interactive turing machines. In *Proceedings of the 19th IEEE Workshop on Computer Security Foundations*, CSFW '06, pages 309–320, Washington, DC, USA, 2006. IEEE Computer Society.

[28] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, march 1996.

[29] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc.*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[30] Sebastian Moedersheim and Luca Viganò. Sufficient conditions for vertical composition of security protocols. In *ASIACCS*, pages 435–446, 2014.

[31] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati, editors, *CCS 2000, Proceedings of the 7th ACM Conference on Computer and Communications Security, Athens, Greece, November 1-4, 2000.*, pages 245–254. ACM, 2000.

[32] Thomas Shrimpton, Martijn Stam, and Bogdan Warinschi. A modular treatment of cryptographic apis: The symmetric-key case. In *Annual Cryptology Conference*, pages 277–307. Springer, 2016.

Appendix A.
Some more examples

A.1. Continuing Example 2

We explain here in more details why the PKI protocol presented in Example 2 is indeed secure.

Example 10 (Example 8 continued)**.** *Continuing Example 2, if we denote $C[\_]$ the context $!^i\,\mathsf{agt}(A,\{H[i],D[i]\}).!^j\,\mathsf{agt}(B,\{H[j],D[j]\}).\_$ that declares the agents then we have $C[P[0,0]] \models \phi_{\mathsf{PKI}}$. Consider ϕ_{asy} (ϕ_{sig} being symmetrical). We can first notice that in $P[\__A,\__B]$, the only assignments of type $\mathbb{sk}$ are $[x_{skA} :=_{\mathbb{sk}[A]} sk[A]]$ and $[x_{skB} :=_{\mathbb{sk}[B]} sk[B]]$. Hence for all $C[P[0,0]] \to^* (\mathscr{E};P;\Phi;\sigma;\mu)$, given an honest agent $H[i]$, a variable $x \in \mu(\mathbb{sk}[H[i]])$ will necessarily be instantiated by $sk[H[i]]$ hence ensuring that $C[P[0,0]]$ models the first conjunct of ϕ_{asy}, i.e.,*

$$C[P[0,0]] \models \forall A \in \overline{\mathscr{A}}_H.\forall x,y \in \mathscr{X}_t.(x,y \overline{\in} \mathbb{sk}[A] \Rightarrow x = y)$$

Similarly, considering that $sk[A] = sk[B]$ if and only if $A \neq B$, $C[P[0,0]]$ models the second conjunct of ϕ_{asy}. The fourth conjunct is intuitively satisfied since the secret keys are never sent in plain text.

The third conjunct is less obvious than the previous ones. Intuitively, it relies on the fact that we assumed the server trusted, i.e. $S \in \mathscr{A}_H$. As such, the channel $pc[S,H[i]]$ will remain unaccessible to the attacker and in particular only known between S and $H[i]$. Thus, during the registration phase, it ensures that the variable x_{cert} will be instantiated by the signature $\mathsf{sign}(\langle H[i],\langle \mathsf{pk}(sk[H[i]]),\mathsf{vk}(sig[H[i]])\rangle\rangle, sig[S])$. Moreover, only the server signs with $sig[S]$ and it is always on messages of the form $\langle A,\langle t_1,t_2\rangle\rangle$ where A is the agent with whom he communicates. Thus, even if A is dishonest, e.g. $A = D[j]$ for some j, then the signature he will get from the server will necessarily be of the form $\mathsf{sign}(\langle D[i],\langle t_1,t_2\rangle\rangle, sig[S])$ which cannot be confused with $\mathsf{sign}(\langle H[i],\langle \mathsf{pk}(sk[H[i]]),\mathsf{vk}(sig[H[i]])\rangle\rangle, sig[S])$. Thus, the honest agent $H[j]$ checking the identity of the certificate supposedly coming from the honest agent $H[i]$, e.g. the conditional "$\mathsf{if}\ \mathsf{proj}_1(\mathsf{check}(z,\mathsf{vk}(sig[S]))) = B\ \mathsf{then}$" in the process $P_A[\__A]$, is guaranteed that his assignment variable of type $\mathbb{pk}[H[j],H[i]]$ will be instantiated by $\mathsf{pk}(sk[H[i]])$.

Note that even though ϕ_{asy} only focuses on the honest agent, it is important to also reason about the certificates that dishonest agents can obtain. Indeed, consider a loose PKI that could provide certificates with two different formats, e.g., for different protocols: $\mathsf{sign}(\langle A,\langle pk,vk\rangle\rangle, sig[S])$ and $\mathsf{sign}(\langle vk,\langle pk,A\rangle\rangle, sig[S])$. An attacker could then sends to the server $\langle \mathsf{pk}(D[j]), H[i]\rangle$ through the channel $pc[S,D[j]]$ and obtain the certificate $\mathsf{sign}(\langle H[i],\langle \mathsf{pk}(D[j]), D[j]\rangle\rangle, sig[S])$. An honest agent receiving such certificate would then assume that $\mathsf{pk}(D[j])$ is the public key of $H[i]$.

The following example illustrates why a (contribed) PKI could be secure w.r.t. ϕ_{PKI} while its security breaks as soon as the public keys are revealed.

Example 11. *Consider a mock PKI $P[\__A,\__B] \hat{=} R(A,B) \mid R(B,A) \mid P_A[\_] \mid P_B[\_]$ where $R(A,B)$ and $P_A[\_]$ are defined as follows :*

$$\begin{aligned} R(A,B) \hat{=}\ & \mathsf{in}_S(c,z).\mathsf{if}\ z = \mathsf{pk}(sk[B])\ \mathsf{then} \\ & \quad \mathsf{new}_S k.\mathsf{out}_S(pc[S,A],\langle sk[A],\mathsf{pk}(k)\rangle). \\ & \quad \mathsf{out}_S(c,sk[B]) \\ & \mathsf{else}\ \mathsf{out}_S(pc[S,A],\langle sk[A],\mathsf{pk}(sk[B])\rangle) \end{aligned}$$

$$\begin{aligned} P_A[\_] \hat{=}\ & \mathsf{in}_A(pc[S,A],y).[x_{skA} :=_{\mathbb{sk}[A]} \mathsf{proj}_1(y)]. \\ & [y_{pkB} :=_{\mathbb{pk}[A,B]} \mathsf{proj}_2(y)] \end{aligned}$$

The processes $R(B,A)$ and $P_B[\_]$ are defined analogously. Intuitively, the server is expected to send to the agents private and public keys of the form $sk[A]$ and $\mathsf{pk}[B]$. However, when A wants to get his private key and the public key of B, the server will first check whether the agent already knows the public key $\mathsf{pk}(sk[B])$. If so, we will send to A a fresh public key for B, i.e., $\mathsf{pk}(k)$, and reveal the previous private key. Otherwise, he sends the public key $\mathsf{pk}(sk[B])$. This mock PKI is obviously not realistic but one can notice that even when A and B are two honest agents $P[0,0] \models \phi_{\mathsf{PKI}}$ since the intruder never learn any public key of honest agents and so the test "$z = \mathsf{pk}(sk[B])$" will never hold.

However if you consider the process $Q_A = \mathsf{new}_A s.[z :=_{\mathbb{secret}[A]} s].\mathsf{out}_A(c,\mathsf{pk}(x_{skA})).\mathsf{out}_A(c,\mathsf{aenc}(s,\mathsf{pk}(x_{skA})))$ and the process $Q_B = 0$, then $P[Q_A,Q_B] \not\models \phi_{sec}$ since the attacker can first inject in $R(A,B)$ some nonce which will yield x_{skA} and y_{pkB} to be instantiated by $sk[A]$ and $\mathsf{pk}(sk[B])$ respectively. Then, he can inject the public key $\mathsf{pk}(sk[A])$ obtained from Q_A into $R(B,A)$. In such a case, the test "$z = \mathsf{pk}(sk[A])$" in $R(B,A)$ will succeed and so the secret key $sk[A]$ would be revealed which will allow the intruder to decrypt $\mathsf{aenc}(s,\mathsf{pk}(x_{skA}))$ and obtain s.

Note that this example focuses only on one session of the PKI and would not work for unbounded number of sessions, i.e.,

$$!^i\,\mathsf{agt}(A,\{H[i],D[i]\}).!^j\,\mathsf{agt}(B,\{H[j],D[j]\}).P[0,0] \not\models \phi_{\mathsf{PKI}}$$

However, we can create a more complex process $P[\__A,\__B]$ based on the same idea and relying on counters that would illustrate the unbounded case.

A.2. Permissive Needham-Schroeder protocol

Example 12. *Consider process Q_A from the Needham-Schroeder protocol in Example 3. The process representing the role of an honest agent A talking with an honest agent B in permissive Q, that is, $O_A[Q_A]\sigma_Q^{\mathsf{priv}}\sigma_B^{\mathsf{pub}}$, the usual process where all keys are distributed as expected.*

$$\begin{aligned} & \mathsf{out}_A(d,\mathsf{out}_A(d,\langle \mathsf{pk}(sk[A]),\mathsf{pk}(sk[B])\rangle)). \\ & \mathsf{new}_A n_a.[z :=_{\mathbb{secret}[A,B]} n_a]. \\ & \quad \mathsf{out}_A(c,\mathsf{aenc}(\langle \mathsf{pk}(sk[A]),n_a\rangle,\mathsf{pk}(sk[B]))).\mathsf{in}_{-}(c,y) \\ & \quad \mathsf{if}\ n_a = \mathsf{proj}_1(\mathsf{proj}_2(\mathsf{adec}(y,sk[A])))\ \mathsf{then} \\ & \quad \mathsf{if}\ x_{pkB} = \mathsf{proj}_1(\mathsf{adec}(y,x_{skA}))\ \mathsf{then} \\ & \quad [z' :=_{\mathbb{secret}[A,B]} \mathsf{proj}_2(\mathsf{proj}_2(\mathsf{adec}(y,sk[A])))]. \\ & \quad \mathsf{out}_A(c,\mathsf{aenc}(\mathsf{proj}_2(\mathsf{proj}_2(\mathsf{adec}(y,sk[A]))),\mathsf{pk}(sk[B]))) \end{aligned}$$

Since Q_A and Q_B do not use signing and verification keys, we safely omitted them.

The role of an honest agent A talking with a dishonest agent B in permissive Q, i.e., $I_B^{\mathsf{pub}}[O_A[Q_A]]\sigma_A^{\mathsf{priv}}$, is obtained from Q_A by letting the attacker freely chose the public key of B.

$$
\begin{array}{l}
\mathsf{in}_A(d, y_{pkB}).\mathsf{out}_A(d, \mathsf{pk}(sk[A])).\\
\mathsf{new}_A\, n_a.[z :=_{\mathtt{secret}[A,B]} n_a].\\
\quad \mathsf{out}_A(c, \mathsf{aenc}(\langle \mathsf{pk}(sk[A]), n_a\rangle, x_{pkB})).\mathsf{in}_.(c,y)\\
\quad \mathsf{if}\ n_a = \mathsf{proj}_1(\mathsf{proj}_2(\mathsf{adec}(y, sk[A])))\ \mathsf{then}\\
\quad \mathsf{if}\ y_{pkB} = \mathsf{proj}_1(\mathsf{adec}(y, sk[A]))\ \mathsf{then}\\
\quad [z' :=_{\mathtt{secret}[A,B]} \mathsf{proj}_2(\mathsf{proj}_2(\mathsf{adec}(y, sk[A])))].\\
\quad \mathsf{out}_A(c, \mathsf{aenc}(\mathsf{proj}_2(\mathsf{proj}_2(\mathsf{adec}(y, sk[A]))), y_{pkB}))
\end{array}
$$

We again omitted the unnecessary parts regarding signing/verification keys.