Secure Protocol Composition under Dynamic Corruption: Scaling Up Symbolic Analysis for Real-World Security Properties

Cas Cremers
CISPA Helmholtz Center for Information Security

Erik Pallas
CISPA Helmholtz Center for Information Security

Aleksi Peltonen
CISPA Helmholtz Center for Information Security

Abstract

Although automated symbolic protocol verification has proven valuable and effective, current approaches begin to reach their limits: While small protocols can be analyzed automatically, the most complex case studies often require substantial expert time and resources. There have been many attempts to solve this problem by compositional verification, but they rely on unrealistic protocol assumptions and do not support real-world security properties like Forward Secrecy.

In this work, we enable compositional symbolic analysis for real-world security protocols with respect to modern security properties. We develop a composition result in the Applied π-Calculus that holds even in the presence of attackers capable of dynamic corruption if the protocols satisfy a disjointness requirement.

We demonstrate the applicability and effectiveness of our result on the composition of a data exchange protocol with a Diffie-Hellman key exchange and a compositional analysis of Forward Secrecy in TLS 1.3 within the scope of RFC 8446 and the ECH extension. While monolithic analyses of TLS 1.3 with ECH fail to deliver a result in 10% of cases, all compositional analyses succeed. Additionally, runtime decreases by 71% and memory usage by 86% on average.

1 Introduction

Automated verification of security protocols has proven to be a very valuable and effective approach to security analysis over the last decades. From the seminal discovery of the person-in-the-middle attack on the Needham-Schroeder protocol [50, 51], the extensive usage in the standardization of TLS 1.3 [12, 35, 36] to the analysis of large and complex protocols like WPA2 [37] and SPDM [34], automated tools have led to various significant contributions.

However, the analyses of the latter case studies also show that current approaches reach their limits as protocols become ever larger and more complex. In practice, this is typically reflected in the need for increasing amounts of expert manual guidance for the tools (e.g., specifying invariants/reusable lemmas in Tamarin, recently also implemented in ProVerif).

One way to remedy this is *compositional analysis*: Instead of verifying the properties of a protocol at large, it is divided into smaller subprotocols. These subprotocols are then analyzed separately, and the combined results provide guarantees for the entire protocol. This not only allows for simpler proofs but also reusing protocol models and their analyses when investigating other protocols. Unfortunately, the composition of protocols is not secure in general. Even if two protocols are secure in isolation, an adversary can use the behavior of or knowledge gained from one protocol to attack the other.

In the context of symbolic protocol analysis, several results exist that state under which conditions secure composition is possible, covering a multitude of composition types (e.g., parallel and sequential [31] or vertical [44]), protocol classes (e.g., password-based protocols [40] or public-key infrastructures [28]), and security properties (e.g., trace properties [53] or privacy properties [4, 5]). One would therefore expect that modern protocol analyses, notably those specified in the Applied π-Calculus, such as [13, 17, 27], would use such composition results. It turns out that these composition results are in fact not used in practice. There are two main reasons for this. First, the composition results only apply to restricted protocol classes, which prevents them from being used for real-world protocols. Second, they only provide guarantees for static corruption threat models, whereas modern protocol analyses aim to prove stronger guarantees.

Consider, for example, TLS 1.3. One of its main achievements compared to its predecessors is that it ensures Forward Secrecy in all situations, i.e., even if a long-term key gets leaked, all previously exchanged data remains secret. The enforcement of Forward Secrecy in the standard has been strongly argued for (see, e.g., [6] for a related discussion on the TLS 1.3 mailing list), and it is is now best-practice that key-exchange protocols guarantee Forward Secrecy.

To analyze protocols for Forward Secrecy and similar properties we need to consider threat models with *dynamic corruption* of values, which is out of scope for the previously

mentioned results. Dynamic corruption refers to an adversary's ability to obtain secrets at some point in the lifetime of the system, as opposed to static corruption, which only allows a secret to be never compromised, or always. Dynamic corruption allows us to specify stronger security guarantees, such as Forward Secrecy, that limit the temporal impact of a compromise.

Our core contribution now is a general composition result in the Applied π-Calculus that allows us to retain security, even with respect to dynamic corruption, when composing protocols. The protocols may use any cryptographic primitives that can be modeled using equational theories. Our result can be used with tools such as ProVerif [14] or SAPIC+ [30].

Additionally, whe show that unlike previous results, our result applies to real-world protocols. We illustrate our result by composing a Diffie-Hellman key exchange with a data exchange protocol and with a compositional analysis of Forward Secrecy in TLS 1.3 ECH as defined in RFC 8446 and the ECH extension. Compared to its latest state-of-the-art analysis in [13], where the authors could not prove 63% of properties, we are able to prove all properties of interest with our result. Additionally, we see improvements in time and memory consumption of up to two orders of magnitude.

En route, we also generalize previous results: We prove that two protocols that have no common signature, equational theory or variables can be securely composed, regardless of the cryptographic primitives used. Earlier results only allowed for concatenation, symmetric and asymmetric encryption, signatures and hash functions [33, 45].

1.1 Overview

Below, we first expand some of the details and restrictions of our composition result. In Section 2 we discuss related work. We give background on corruption, disjointness, and the Applied π-Calculus in Section 3. In Section 4 we define our corruption model and provide our formal definitions for composed processes and shared data, which allows us to formulate our composition result in Section 5. We show how our result can be used in practice in Section 6 and conclude in Section 7. In the appendix, we give extensive case study results.

We provide the ProVerif models of our case studies and the benchmarking script in [38]. There, we also provide the Full Version of this work, including all theoretical preliminaries and the full proofs.

1.2 Composition Result Properties

We provide a composition result that allows us to compose protocols so that they retain trace properties in the presence of an attacker able to dynamically corrupt arbitrary values. Previous results were restricted to corrupting non-shared data, i.e., information that is used in only one of the protocols.

Protocols may use any cryptographic primitives representable by an equational theory even if it is not confluent, as long as the protocols do not use the same primitives. The protocols are specified in the Applied π-Calculus and unrestricted in their control flow: They may communicate over channels, feature non-trivial else-branches and incorporate replication. Other results often lack at least one of these features. The two protocols may also pass data unidirectionally: One protocol may pass data to the other, the second protocol may not. Many previous results do not allow this form of communication or, if they do, come with other restrictions.

We achieve this in a framework that allows us to specify if and when information has been leaked to the adversary, making it possible to formalize notions like Forward Secrecy. Previous work either lacks this or at least one of the aforementioned features.

In summary, our result is the first that enables the secure composition of protocols that share and pass data in the presence of an attacker capable of dynamic corruption while having no restrictions on the control flow and the power to reference data leakage on the trace. With this, we are able to compose large classes of security protocols, e.g., key-exchange protocols with protocols using the resulting keys; these protocols can now be analyzed separately for properties like Forward Secrecy.

However, this result also comes with a few restrictions. First of all, it requires the composed protocols to have disjoint signatures, i.e., do not use the same cryptographic primitives. While this holds for many practically relevant scenarios, it just as often does not. In most cases, we can enforce disjoint signatures through the use of appropriate tagging.

Second, the theorem is only applicable to trace properties; it does not cover equivalence properties, which are often used to model privacy-type guarantees.

Third, data may only be transferred in one direction, i.e., only one of the protocols may pass information to the other. While this is sufficient in many scenarios, it would be helpful to provide input to the other protocol, e.g., pass information to a key-derivation protocol which then returns (session) keys.

Finally, on a more technical side, channel names cannot be passed between protocols. This restricts communication between processes but can often be circumvented by simply using the variables themselves as storage for shared data.

2 Related Work

Symbolic vs. Computational Model. Secure protocol composition has been investigated in both major models of security: the *symbolic* and the *computational model*. We achieve our results in the symbolic model. The benefit of symbolic approaches, which take a more abstract view of cryptography, is that they make it possible to scale better for analyzing larger parts of systems. The symbolic abstraction of cryptography considers abstract terms instead of concrete bitstrings,

considers an abstract attacker model that assumes that cryptographic primitives provide idealized black-box guarantees, and does not include any notions of key or nonce length. Thus, symbolic analyses typically focus more on logical attacks at the protocol level and do not cover attacks that rely on the low-level details of the cryptographic primitives.

Symbolic Composition Results. Existing composition results in the symbolic model lack the capability to reason about dynamic corruption and can thus not handle security properties like Forward Secrecy. Our results enable reasoning about dynamic corruption. They are also *generic* in that they are not tailored to any specific protocol or class of protocols.

Generic composition results typically rely on a *disjointness requirement* on the protocols to compose; in [33, 45, 46], protocols can be securely composed as long as they do not share data and use different cryptographic primitives. The primitives are limited to encryption, concatenation, signatures and hash functions. Our results also require this kind of *disjoint primitives*, but can be used with arbitrary primitives.

In [31] it is shown that the protocols to compose may also share keys as long as they remain secret. This result is lifted from trace properties to equivalence properties in [4, 5]. None of these works provide a model of dynamic corruption, although the results of the latter two hold if non-shared variables are dynamically compromised. Our results enable dynamic corruption of all variables.

Various other disjointness notions are used throughout the literature. The composition theorems for sequential [3] and parallel composition [20] can be emulated by our results. The results of [39] on key exchange-protocols, [29, 42, 44, 56, 57] on secure channels and [28] on public key-infrastructures require slightly different disjointness criteria than our theorems, but none of them covers dynamic corruption.

Computational Composition Results. Similar work has been done in the computational model, most notably with the *Universal Composition* framework [23], its variations and extensions [21, 24, 25, 43], and similar approaches like *Reactive Simulatability* [7–9], *GNUC* [48], *iUTC* [22], *IITM* [49], and *Oracle Simulation* [32]. Typically, the composition theorems in these frameworks are very restrictive and thus seldomly used in practice, and the methodology has no support for automation.

Computer-aided Verification. For computer-aided verification of security protocols, the OWL tool [41] and the DY* framework [10] provide forms of support for compositional reasoning. OWL does not support dynamic corruption, and hence cannot be used to prove properties such as Forward Secrecy. DY* on the other hand provides little to no automation and only allows the composition of so-called *layered protocols* [11].

TLS 1.3 Analyses. Finally, there are several formal and automated analyses of TLS 1.3, its extensions and their respective drafts (e.g., [13, 26, 35, 36]). None of them are compositional in that they rely on composability theorems to achieve their results, except for [12, 18]. Their analysis of Draft 18 of the TLS 1.3 standard makes use of compositional arguments developed for CryptoVerif [15] and its framework that are specific to the TLS 1.3 protocol.

3 Background

We first provide background on three key concepts we use: dynamic corruption, which is the threat model we assume; the type of protocol disjointness we require; and the Applied π-Calculus, the formalism in which we achieve our results.

3.1 Dynamic Corruption

Security properties are always relative to a threat model. Historically, it is often assumed that an adversary initially knows none of the encryption keys or other secrets owned by the parties executing a protocol. All information an adversary learns must be derived from the information learned by observing the messages exchanged by the parties.

However, this is a fairly strong assumption. In practice, an adversary might learn secrets through successful attacks on other protocols or systems, court orders, or simply user carelessness. Even if that happens, a protocol should ideally provide guarantees on past and future communication.

The literature contains multiple security properties that capture such guarantees, most notably *Forward Secrecy*, sometimes also called *Perfect Forward Secrecy*. First proposed in [47], it states that even if a long-term encryption key is leaked, all communication encrypted with a key derived from that long-term key remains secret. Forward Secrecy is a highly desired property, provided by the likes of TLS 1.3 [58] and the Signal protocol through its usage of X3DH [52].

Consider the following equivalent definition from [54].

Definition 3.1 (Forward Secrecy)**.** *A protocol is said to have* perfect forward secrecy *if compromise of long-term keys does not compromise past session keys.*

To formally analyze protocols for Forward Secrecy we need to model that keys can be compromised during the execution of a protocol. It requires an adversary to *dynamically corrupt* any given secret; we informally capture this as follows.

Definition 3.2 (Dynamic Corruption)**.** *Consider a protocol P that uses some secret data item d. An adversary is capable of* dynamic corruption *of d if they can gain knowledge of d at any point in time during the execution of P.*

Dynamic corruption of d is usually modeled by adding a step to a model of P that sends d directly to the attacker. We also follow this approach and formalize a generic transformation of P into a *dynamically corrupted process* in Section 4.1.

3.2 Protocol Disjointness

As we have seen in Section 2, composition results typically require some kind of *disjointness*: The protocols that are composed have to work differently enough so that they cannot critically confuse any of their own messages with those of the other protocol. Often, this is achieved through either *tagging* or *disjoint primitives*.

Tagging requires that messages sent over public channels are annotated in some way. This annotation often includes at least one of the sending party, the intended recipient, or the protocol or protocol mode that generated the message. On the other hand, two protocols use disjoint primitives if no cryptographic primitive is used in both protocols.

In this work, we assume the latter. Not only does it allow for more elegant protocol models, but it is also not a stronger restriction than tagging, as tagging can be used to achieve disjoint primitives: Simply tag every message with the protocol it has been sent from.

However, using disjoint primitives does not imply that protocols have to use strictly different cryptographic primitives – it merely requires that the usage of the primitives produces sufficiently different results. More precisely, we can consider two instantiations of the same cryptographic primitive disjoint if, given the same input, they return different outputs.

The technical reason for this lies in how we model cryptographic functions. Roughly, we abstract their output as a term containing their function name and all provided input. If two functions now have different output for the same input, we change the function name to differentiate the resulting terms.

We formalize our notion of disjoint primitives in the next subsection. As an example, consider a protocol that uses two hash functions, one of which produces a 256-bit value, while the other returns a 512-bit hash. For any input i they produce different results, so we can model the functions as H_{256} and H_{512} and their values as $H_{256}(i)$ and $H_{512}(i)$. Similarly, if one protocol step always computes *HKDF* with constant tag t, and another protocol step computes *HKDF* with constant tag t' ($t \neq t'$), we consider the *HKDF*s disjoint and model them as two different functions $HKDF_t$ and $HKDF_{t'}$.

3.3 The Applied π-Calculus

In the following, we introduce the key aspects of the Applied π-Calculus [2], using the *extended processes* notion of [1]. Due to space constraints, some ancillary definitions are omitted here; they can be found in the Full Version of this work, provided in [38].

3.3.1 Variables and Terms

The basic building block of our protocol models are *function symbols*, which represent cryptographic primitives. Every function symbol f has an arity $\mathrm{ar}(f)$; we write $f\langle\mathrm{ar}(f)\rangle$. All function symbols used in a protocol P form its *signature* $\Sigma(P)$.

We assume that $\Sigma(P)$ contains a countably infinite set of function symbols of arity 0, which we call *names* and denote $\mathrm{names}(P)$; they represent data created during an execution of a protocol. We additionally assume a countably infinite set of variables $\mathcal{V}(P)$, for which $\Sigma(P) \cap \mathcal{V}(P) = \emptyset$. The set $\mathcal{T}(\Sigma(P), \mathcal{V}(P))$ contains all terms over signature $\Sigma(P)$ and variables $\mathcal{V}(P)$; $t \in \mathcal{T}(\Sigma(P), \mathcal{V}(P))$ if it is generated by the following grammar:

$t :=$	TERMS
$\mid x$	$x \in \mathcal{V}(X)$
$\mid f(t_1, \ldots, t_n)$	$f \in \Sigma(P),\ \mathrm{ar}(f) = n$

The root symbol of a term t is denoted $\mathrm{root}(t)$; t is called a *ground term* if it does not contain any variable. If $t = f(t_1, \ldots, t_n)$, $t_1, \ldots, t_n$ are *subterms* of t; recursively, subterms of $t_1, \ldots, t_n$ are also subterms of t. We will use terms to model messages sent over the network during a protocol execution.

A *substitution* $\sigma := \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ describes a replacement of variables with terms; the domain $\mathrm{dom}(\sigma) := \{x_1, \ldots, x_n\}$ contains all variables for which σ contains a mapping. For a term t, $t\sigma$ describes the term where all variables in $\mathrm{dom}(\sigma)$ have been replaced in t with the corresponding terms. σ is called a *ground substitution* if it maps every variable in $\mathrm{dom}(\sigma)$ to a ground term. To rename variables $x_1, \ldots, x_n$ in a term t, we perform an α-*renaming*, written $t\{^{x_1}/_{x'_1}, \ldots, ^{x_n}/_{x'_n}\}$.

An *equational theory* $\mathrm{E}(P)$ models the properties of a cryptographic primitive from $\Sigma(P)$. It is a finite set of equations $s = t$ of terms $s, t \in \mathcal{T}(\Sigma(P), \mathcal{V}(P))$. We denote with $=_{\mathrm{E}(P)}$ the closure of $\mathrm{E}(P)$ under equivalence, substitution of variables with terms and application of function symbols according to $\mathrm{E}(P)$. If we are concerned with the union $\mathrm{E}(P) \cup \mathrm{E}(Q)$ of two equational theories, we often write $=_{\mathrm{E}}$ instead of $=_{\mathrm{E}(P)\cup\mathrm{E}(Q)}$. The equivalence class of all terms equal to t according to an equational theory E is denoted $[t]_{=_{\mathrm{E}}}$. Note that in this work, we will only consider non-trivial equational theories, i.e., those that do not equate all terms. If for some substitution σ and some term $t \in \mathcal{T}(\Sigma(P), \mathcal{V}(P))$ there is another term $r \in \mathcal{T}(\Sigma(P) \setminus \mathrm{names}(P), \mathrm{dom}(\sigma))$ such that $r\sigma =_{\mathrm{E}} t$, we say that t can be *deduced* from σ and write $\sigma \vdash t$.

Example 3.3. Consider the modular exponentiation used in a Diffie-Hellman key exchange. We need one function of arity 2, so we have $\Sigma(DH) := \{mexp\langle 2\rangle\}$. If we assume that g is a primitive root modulo some prime number p, we can express that the modular exponentiation is commutative with $\mathrm{E}(DH) := \{\mathtt{mexp}(\mathtt{mexp}(g,a),b) = \mathtt{mexp}(\mathtt{mexp}(g,b),a)\}$.

3.3.2 Syntax

The process syntax is defined by the grammar given in Figure 1, where $x \in \mathcal{V}(P)$ is a variable, c is a name, and t, M and N are terms built over $\Sigma(P)$.

The meaning of the respective operators is the usual; we will formalize it in Section 3.3.3. Informally speaking, the nil

$P,Q :\equiv$	PROCESSES
$\mid 0$	Nil process
$\mid \nu x.P$	Name restriction
$\mid \mathbf{in}(c,x).P$	Input
$\mid \mathbf{out}(c,t).P$	Output
$\mid \mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q$	Case distinction
$\mid P \parallel Q$	Parallel composition
$\mid\ !P$	Replication

Figure 1: Syntax of Applied π-Calculus Processes

process does nothing; the name restriction $\nu x.P$ binds x to a name in P. An input $\mathbf{in}(c,x).P$ reads a value from channel c and stores it in variable x; an output $\mathbf{out}(c,t).P$ on the other hand sends the term t on channel c. Both then continue with P. After a case distinction $\mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q$ the process continues with P if M equals N in a given equational theory, otherwise with Q. As a more concise notation, we also write $[M = N]\ P : Q$ for the case distinction. A parallel composition $P \parallel Q$ interleaves the execution of P and Q in an arbitrary order, unless they can synchronize on an input and output on the same channel. Finally, a replication $!P$ is an parallel composition of P with itself, repeated arbitrarily often, possibly infinitely.

Operator precedence in ascending order is replication !, parallel composition $\parallel$, and finally sequential composition .; if in doubt, this will be made clear by the appropriate usage of round brackets $(\ldots)$. On a structural level, we consider processes equal modulo α-renaming and the equivalences $0.P \equiv P.0,\ P.0 \equiv P,\ !0 \equiv 0,\ P \parallel 0 \equiv P,\ P \parallel Q \equiv Q \parallel P$, and $(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$ for processes P, Q and R.

Note that the above definition of a process is minimal. Many convenient features like assignments or complex case distinctions are not explicitly included but can be replicated with the given operations; see the Full Version for a list of possible constructions. We will use them for modeling, but we will assume for all proofs that processes consist only of basic operations.

Formalizing the disjointness requirement from Section 3.2, we say that two protocols P and Q built over $\Sigma(P)$ and $\Sigma(Q)$, respectively, use *disjoint primitives* if $\Sigma(P) \cap \Sigma(Q) = \emptyset$.

For a process P, $\mathrm{vars}(P)$ denotes the set of all variables occurring in P, $\mathrm{fv}(P)$ and $\mathrm{bv}(P)$ are the sets of all free and bound variables in P, respectively, and $\mathrm{names}(P)$ is the set of all names occurring in P. We also extend α-renaming to a process P by applying it to all terms occurring in P.

Example 3.4. We model a Diffie-Hellman key exchange as a process. One party binds a fresh name to a variable a, which represents the generation of a random number. It then sends out the modular exponentiation of some constant term g and a over a channel c, while waiting for the other party to send its half-key B on channel d. Once B has arrived, it stores the result of a modular exponentiation of B and a in a variable k_A.

$$DH_A :\equiv \nu a.(\mathbf{out}(c,\mathtt{mexp}(g,a)) \parallel \mathbf{in}(d,B)).k_A := \mathtt{mexp}(B,a)$$

Analogously, the other party generates some b to use as the exponent, sends its half-key on d and listens on c.

$$DH_B :\equiv \nu b.(\mathbf{in}(d,A) \parallel \mathbf{out}(d,\mathtt{mexp}(g,b))).k_B := \mathtt{mexp}(A,b)$$

Composing DH_A and DH_B in parallel yields the full model.

$$DH :\equiv DH_A \parallel DH_B$$

3.3.3 Semantics

We are not only interested in the process execution itself but also in whether an adversary might be able to deduce information from the communication and whether they are able to interfere with the process in unwanted ways. Therefore, we *extend* the processes with additional structures that keep track of variable assignments and adversary knowledge.

An *extended process* consists of a set $\mathcal{E}$ of *restricted names* that are unknown to the adversary, the process P to execute, a substitution φ that represents the *adversary knowledge*, and a substitution σ that represents the *variable assignments*.

Definition 3.5 (Extended Processes)**.** *Let $\mathcal{E}$ be a set of names, P be a process such that $\mathrm{fv}(P) = \emptyset$, and let φ and σ be ground substitutions. Then an* extended process *is a 4-tuple $(\mathcal{E},P,\varphi,\sigma)$. Instead of $(\emptyset,P,\emptyset,\emptyset)$ we will simply write P.*

The semantics of an extended process are given as a labeled transition relation $\xrightarrow{l}$, where l can be either an *external action*, i.e., an output to or an input from a channel, or an *internal action* τ if there is no interaction with the environment.

Definition 3.6 (Semantics of Extended Processes)**.** *Let $\mathcal{P}$ and $\mathcal{P}'$ be two extended processes. $\mathcal{P}'$ is* reachable from $\mathcal{P}$ *if $(\mathcal{P},\mathcal{P}') \in \xrightarrow{tr}$, where $\xrightarrow{tr}$ is the reflexive transitive closure of the relation described by the rules in Figure 2.*

For a more intuitive description of the transition rules see the Full Version.

We write $\mathcal{P} \xrightarrow{tr} \mathcal{P}'$ instead of $(\mathcal{P},\mathcal{P}') \in \xrightarrow{tr}$, and we write $\mathcal{P} \Rightarrow \mathcal{P}'$ if $\mathcal{P}'$ can be reached from $\mathcal{P}$ via some unspecified series of actions. The *length* $|tr|$ of a transition is the number of actions needed to perform this transition.

Example 3.7. The semantics allow both the desired behavior of DH, and an adversary to block its execution by intercepting all output and not providing any input; in doing so, they learn both the half-keys:

$$\begin{aligned} DH \xrightarrow{tr} (\{c_A,c_B,n_a,n_b\}, &\mathbf{in}(d,B).k_A := \mathtt{mexp}(B,a) \\ &\parallel \mathbf{in}(c,A).k_B := \mathtt{mexp}(A,b), \{w_1 \mapsto \mathtt{mexp}(g,a), \\ &w_2 \mapsto \mathtt{mexp}(g,b)\}, \{a \mapsto n_a, b \mapsto n_b\}) \end{aligned}$$

$$\frac{M\sigma =_{\mathrm{E}} N\sigma}{(\mathcal{E}, \mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q \parallel R, \varphi, \sigma) \xrightarrow{\tau} (\mathcal{E}, P \parallel R, \varphi, \sigma)}\ \text{IF} \qquad \frac{M\sigma \neq_{\mathrm{E}} N\sigma}{(\mathcal{E}, \mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q \parallel R, \varphi, \sigma) \xrightarrow{\tau} (\mathcal{E}, Q \parallel R, \varphi, \sigma)}\ \text{ELSE}$$

$$\frac{\mathrm{vars}(M) \subseteq \mathrm{dom}(\varphi) \quad \mathrm{names}(M) \cap \mathcal{E} = \emptyset \quad c \notin \mathcal{E} \quad x \notin \mathrm{dom}(\sigma) \quad M\varphi = t}{(\mathcal{E}, \mathbf{in}(c,x).P \parallel R, \varphi, \sigma) \xrightarrow{\mathbf{in}(c,M)} (\mathcal{E}, P \parallel R, \varphi, \sigma \cup \{x \mapsto t\})}\ \text{IN}$$

$$\frac{c \notin \mathcal{E} \quad w_n \text{ variable} \quad n = |\varphi| + 1}{(\mathcal{E}, \mathbf{out}(c,t).P \parallel R, \varphi, \sigma) \xrightarrow{\nu w_n.\mathbf{out}(c,w_n)} (\mathcal{E}, P \parallel R, \varphi \cup \{w_n \mapsto t\sigma\}, \sigma)}\ \text{OUT-T}$$

$$\frac{c, h \notin \mathcal{E}}{(\mathcal{E}, \mathbf{out}(c,h).P \parallel R, \varphi, \sigma) \xrightarrow{\mathbf{out}(c,h)} (\mathcal{E}, P \parallel R, \varphi, \sigma)}\ \text{OUT-CH} \qquad \frac{c \notin \mathcal{E} \quad h \in \mathcal{E} \quad a \text{ fresh name}}{(\mathcal{E}, \mathbf{out}(c,h).P \parallel R, \varphi, \sigma) \xrightarrow{\nu a.\mathbf{out}(c,a)} (\mathcal{E}, (P \parallel R)\{^h/_a\}, \varphi, \sigma\{^h/_a\})}\ \text{OPEN-CH}$$

$$\frac{x \notin \mathrm{dom}(\sigma)}{(\mathcal{E}, \mathbf{out}(c,t).P \parallel \mathbf{in}(c,x).Q \parallel R, \varphi, \sigma) \xrightarrow{\tau} (\mathcal{E}, P \parallel Q \parallel R, \varphi, \sigma \cup \{x \mapsto t\sigma\})}\ \text{SYNC} \qquad \frac{x \notin \mathrm{dom}(\sigma) \quad n \text{ fresh name}}{(\mathcal{E}, \nu x.P \parallel R, \varphi, \sigma) \xrightarrow{\tau} (\mathcal{E} \cup \{n\}, P \parallel R, \varphi, \sigma \cup \{x \mapsto n\})}\ \text{NEW}$$

$$\frac{x' \text{ fresh variable}}{(\mathcal{E}, !P \parallel R, \varphi, \sigma) \xrightarrow{\tau} (\mathcal{E}, P\{^x/_{x'}\}_{x \in \mathrm{bv}(P)} \parallel (!P) \parallel R, \varphi, \sigma \cup \{x' \mapsto x\sigma\}_{x \in \mathrm{bv}(P)})}\ \text{REP}$$

Figure 2: Transition Rules Describing the Semantics of an Extended Process

3.3.4 Traces and Trace Properties

The properties of an extended process are defined relative to the labels it produces and the knowledge an adversary acquires along the way. To capture the latter, we use *frames*. A frame combines the adversary knowledge and the restricted names at a given point of an execution – essentially, all the information which the adversary has and they may not have.

Definition 3.8 (Frame)**.** *Let* φ *be a substitution, and let* $\mathcal{E}$ *be a finite set of names. Then a* frame *is defined as* $\phi := \nu\mathcal{E}.\varphi$.

Two frames might contain the same information modulo an equational theory, even if they are not structurally equal. For example, if an adversary learns $\mathtt{mexp}(\mathtt{mexp}(g,a),b)$ from one execution of a process and $\mathtt{mexp}(\mathtt{mexp}(g,b),a)$ from another, they have actually learnt the same thing. Frames like these are called *statically equivalent*.

Definition 3.9 (Static Equivalence of Frames)**.** *Let* $\phi_1 = \nu\mathcal{E}.\varphi_1$ *and* $\phi_2 = \nu\mathcal{E}.\varphi_2$ *be two frames such that* $\mathrm{dom}(\varphi_1) = \mathrm{dom}(\varphi_2)$. *If for every pair of terms* M, N *such that* $(\mathrm{names}(M) \cup \mathrm{names}(N)) \cap \mathcal{E} = \emptyset$ *it holds that* $M\varphi_1 =_{\mathrm{E}} N\varphi_1 \Leftrightarrow M\varphi_2 =_{\mathrm{E}} N\varphi_2$ *modulo* α*-renaming, then* ϕ_1 *and* ϕ_2 *are* statically equivalent*, or* $\phi_1 \sim \phi_2$.

When an extended process is reduced to any form, the semantics induce a series of labels. Paired with the frame resulting from the execution, this constitutes a *trace*.

Definition 3.10 (Trace)**.** *Let* $\mathcal{P} = (\mathcal{E}, P, \varphi, \sigma)$ *be an extended process. The following set contains its* traces*:*

$$\mathrm{tr}(\mathcal{P}) := \{(tr, \nu\mathcal{E}'.\varphi') \mid \exists(\mathcal{E}', P', \varphi', \sigma') \colon \mathcal{P} \xrightarrow{tr} (\mathcal{E}', P', \varphi', \sigma')\}$$

If we want to indicate that a trace $\lambda = (tr, \nu\mathcal{E}'.\varphi')$ is the result of a transition from the extended process $\mathcal{P}$ to $\mathcal{P}'$, instead of $\mathcal{P} \xrightarrow{tr} \mathcal{P}'$, we will write $\mathcal{P} \xrightarrow{\lambda} \mathcal{P}'$. We denote with $\mathrm{tr}(\mathcal{P} \Rightarrow \mathcal{P}')$ and $\mathrm{tr}(\mathcal{P} \xrightarrow{tr} \mathcal{P}')$ the trace resulting from a transition from $\mathcal{P}$ to $\mathcal{P}'$; we will also apply this to multiple consecutive transitions, e.g., $\mathrm{tr}(\mathcal{P} \Rightarrow \mathcal{P}' \xrightarrow{tr'} \mathcal{P}'')$. Finally, we say that tr is a *trace of* P *over* $\Sigma(Q)$, or $\lambda \in \mathrm{tr}(P, \Sigma(Q))$, if λ is a trace of P when P would be defined over $\Sigma(Q)$, where $\Sigma(P) \subseteq \Sigma(Q)$.

To state the properties of a trace, we extend the notion of static equivalence to traces: Two traces are statically equivalent if they have the same sequence of observable actions and if their frames are statically equivalent.

Definition 3.11 (Static Equivalence of Traces)**.** *Let* $\lambda_1 = \{tr_1, \phi_1\}$ *and* $\lambda_2 = \{tr_2, \phi_2\}$ *be two traces. Then,* λ_1 *and* λ_2 *are* statically equivalent *if* $tr_1 = tr_2$ *modulo* α*-renaming and* $\phi_1 \sim \phi_2$.

A *trace property* is a set of statically equivalent traces.

Definition 3.12 (Trace Property)**.** *Let* P *be a process, and let* $\alpha \subseteq \mathrm{tr}(P)$ *be a set of traces of* P*. If* $\alpha = \{\lambda \mid \lambda \in \alpha \vee \exists \lambda' \in \alpha \colon \lambda \sim \lambda'\}$*, then* α *is called a* trace property *defined over* P*.*

A trace $\lambda \in \mathrm{tr}(P)$ satisfies *a trace property* α *defined over* P*, or* $\lambda \vdash \alpha$*, if and only if* $\lambda \in \alpha$*:*

$$\lambda \vdash \alpha :\Leftrightarrow \lambda \in \alpha$$

An extended process $\mathcal{P}$ satisfies *a trace property* α*, also denoted* $\mathcal{P} \vDash \alpha$*, if and only if all traces of* $\mathcal{P}$ *satisfy* α*:*

$$\mathcal{P} \vDash \alpha :\Leftrightarrow \forall \lambda \in \mathrm{tr}(\mathcal{P}) \colon \lambda \vdash \alpha$$

Example 3.13. We can define a trace property over all the components of a trace. One of the key properties of $\mathcal{DH}$ is that the adversary does not learn the generated shared key. A trace property could capture this as follows:

$$\begin{aligned}&\mathrm{KeySecrecy}(\mathcal{DH}) :=\\ &\quad \{(tr, \nu\mathcal{E}.\varphi) \in \mathrm{tr}(\mathcal{DH}) \mid \neg(\varphi \vdash \mathtt{mexp}(\mathtt{mexp}(g,a),b))\}\end{aligned}$$

4 Corruption, Composition and Shared Data

In order to formulate our composition result, we now introduce our model of dynamic corruption. Afterwards, we describe how processes can be composed and how they can share data.

4.1 Corruption Model

Since the semantics do not include a way for the adversary to retrieve variable values at will, we have to encode this into the process we want to analyze. This encoding needs to achieve three things: It should be able to leak any given variable, it should be able to do so at any point in time during the execution of the process, and it should not alter the behavior of the process any further than enabling leaks.

To achieve this, we fix a set ω of *corruptible variables*. Then, whenever we bind a corruptible variable x with a νx or an $\mathbf{in}(c,x)$, we compose an $\mathbf{out}(lc,x)$ in parallel to the next step in the process. lc has to be a channel that the adversary can listen to (i.e., not a restricted name); for simplicity, we will require that it does not occur in the base process at all. Since parallel composition is a nondeterministic operator, the $\mathbf{out}(lc,x)$ may be activated before any step in the process execution following the binding of x. Activating the $\mathbf{out}(lc, x)$ leads to the corruption of x: The adversary learns its value. Overall, we formalize this with the following transformation.

Definition 4.1 (Dynamically Corrupted Process)**.** *Let* P *be a process,* $\omega \subseteq \mathrm{vars}(P)$ *be a subset of variables of* P*, and let* $lc \notin \mathrm{names}(P)$ *be a name that is not used in* P*. Then the* dynamically corrupted process $\varsigma(P,\omega)$ leaking on channel lc *is given as* $\varsigma(P,\omega) :\equiv$

$$\begin{cases} 0 & \text{if } P \equiv 0 \\ \nu x.(\varsigma(Q,\omega) \parallel \mathbf{out}(lc,x)) & \text{if } P \equiv \nu x.Q \wedge x \in \omega \\ \nu x.\varsigma(Q,\omega) & \text{if } P \equiv \nu x.Q \wedge x \notin \omega \\ \mathbf{in}(d,x).(\varsigma(Q,\omega) \parallel \mathbf{out}(lc,x)) & \text{if } P \equiv \mathbf{in}(d,x).Q \wedge x \in \omega \\ \mathbf{in}(d,x).\varsigma(Q,\omega) & \text{if } P \equiv \mathbf{in}(d,x).Q \wedge x \notin \omega \\ \mathbf{out}(d,t).\varsigma(Q,\omega) & \text{if } P \equiv \mathbf{out}(d,t).Q \\ [M=N]\ \varsigma(Q_1,\omega) : \varsigma(Q_2,\omega) & \text{if } P \equiv [M=N]\ Q_1 : Q_2 \\ \varsigma(Q_1,\omega) \parallel \varsigma(Q_2,\omega) & \text{if } P \equiv Q_1 \parallel Q_2 \\ !\varsigma(Q,\omega) & \text{if } P \equiv\ !Q \end{cases}$$

Note that unbound variables are not leaked under ς. However, unbound variables do not carry any information either way, so they can safely be ignored.

Example 4.2. Recall *DH*, the model of a Diffie-Hellman key exchange we developed in Section 3.3. If we want to analyze the properties it has when the resulting keys are leaked, we can turn *DH* into a corrupted process and leak k_A and k_B:

$$\varsigma(DH, \{k_A, k_B\}) :\equiv (DH_A.\mathbf{out}(lc,k_A) \parallel DH_B.\mathbf{out}(lc,k_B))$$

Of course, this is especially interesting if *DH* and its keys are actually used by another protocol. We will see in the next section how *DH* can be composed with other protocols.

4.2 Composed Processes and Shared Data

Next, we describe how two protocols can be composed and how they share data. We are interested in two kinds of shared data: *pre-shared data*, which is assumed to be distributed amongst all participants before protocol execution, and *transferred data*, which is sent from one protocol to another.

4.2.1 Pre-shared Data and Composition Contexts

If some value is already fixed before the start of a protocol execution, we call it *pre-shared data*. To model it, we can leave the corresponding variables unbound and instantiate them in a *composition context* instead. As in [4, 5], a composition context is essentially a process restricted to νx bindings and replication !, with exactly one hole that the composed process will fill. As such, it also describes the way in which processes are composed, e.g., by putting a process under replication if it is supposed to repeatedly generate a session key.

Definition 4.3 (Composition Context)**.** *A* composition context C *is defined by the following grammar for a variable* x*:*

$$C :\equiv \ \_ \mid \nu x.C \mid\ !C$$

For a process P*,* $C[P]$ *is the* filled composition context.

$P[], Q[] :\equiv$	PARTIAL PROCESS
$\mid 0$	Nil process
$\mid \_.P[]$	Hole
$\mid \nu x.P[]$	Name restriction
$\mid \mathbf{in}(c,x).P[]$	Input
$\mid \mathbf{out}(c,t).P[]$	Output
$\mid [M=N]\ P[] : Q[]$	Case distinction
$\mid P[] \parallel Q[]$	Parallel composition
$\mid\ !P[]$	Replication

Figure 3: Syntax of Applied π-Calculus Partial Processes

The shorthands defined for processes in Section 3.3.2 carry over to composition contexts by ignoring the hole; e.g., $\text{vars}(\nu x.\_) = \{x\}$. To make explicit that $\text{vars}(C)$ represents pre-shared data, we will also write $\text{psh}(C)$ and define $\text{psh}(C[P]) := \text{psh}(C)$ for a process P.

Example 4.4. In *DH*, the exact value of the generator g is of no great interest. We can thus model g as an unbound variable that is instantiated by a simple composition context $C :\equiv \nu g.\_$. This context can then be filled by a *DH* key exchange.

$$C[DH] \equiv \nu g.(DH_A \parallel DH_B)$$

4.2.2 Transferred Data and Executables

Often, not all data shared by two protocols is fixed from the very beginning, but it is generated by one protocol and transferred to the other. This *transferred data* can also be modeled by unbound variables. If we add a placeholder to the process, we can later replace it with another process that binds the transferred data.

Definition 4.5 (Partial Process)**.** *Let $x \in \mathcal{V}()$ be a variable, c a name, and t, M and N terms built over $\Sigma(P)$. Then a* partial process *is defined by the grammar given in Figure 3.*

At its core, a partial process is a process with holes that can be filled by other processes. For simplicity, we will assume that every partial process has exactly one hole. The unbound variables of a partial process $P[]$ and a composition context C represent transferred data and will be denoted by $\text{trf}(C[P[]]) := \text{fv}(C[P[]])$. When we combine C and $P[]$ (and Q), we call the union of pre-shared and transferred data *shared data* and write $\text{sh}(C[P[Q]]) := \text{psh}(C) \cup \text{trf}(C[P[]])$.

Example 4.6. We model a basic data exchange protocol. Consider the equational theory $\text{E}(XP) := \{\texttt{sdec}(\texttt{senc}(x,k),k) = x\}$ and the following processes.

$$XP_A :\equiv \nu data.\mathbf{out}(xc, \texttt{senc}(data, k_A^{\text{sess}}))$$
$$XP_B :\equiv \mathbf{in}(xc,t).x := \texttt{sdec}(t, k_B^{\text{sess}})$$

A generates a data item, encrypts it with a session key, and sends it on channel xc. B listens to this channel, retrieves the encrypted message, and stores its decryption in variable x.

The session key can be provided by any key exchange protocol. We thus combine XP_A and XP_B to a partial process that could be completed by *DH*.

$$XP :\equiv !\_.(XP_A \parallel XP_B)$$

Many protocols require a certain output from other protocols they use; e.g., for *XP* to work as intended, it needs symmetric encryption keys. *DH* thus has to guarantee that $k_A =_{\text{E}} k_B$. We call this type of guarantee on the relations of variables provided by a protocol a *set-up guarantee*.

Definition 4.7 (Set-up Guarantee)**.** *Let $\varsigma(Q,\omega)$ be a corrupted process, and let $x_i, y_j \in \text{vars}(Q)$ be variables. A* set-up guarantee *μ is the transitive closure of a set $\{x_i =_{\text{E}(Q)} y_j \mid 1 \le i,j \le n\}$ of n equations for $n \in \mathbb{N}$. Its* domain $\text{dom}(\mu) := \{x \mid \exists y \in \text{vars}(Q)\colon x = y \in \mu\}$ *is the set of all variables occurring in μ. We define that $\varsigma(Q,\omega)$ satisfies μ, or $\varsigma(Q,\omega) \Vdash \mu$, as follows:*

$$\begin{aligned}\varsigma(Q,\omega) \Vdash \mu :\Leftrightarrow\ & \forall \lambda \in \text{tr}(\varsigma(Q,\omega))\colon (\varsigma(Q,\omega) \xrightarrow{\lambda} (\mathcal{E},Q',\varphi,\sigma)\\ & \Rightarrow ((\forall w \in \omega\colon \varphi \not\vdash w\sigma)\\ & \Rightarrow (((x \in \text{dom}(\sigma) \wedge y \in \text{dom}(\sigma) \wedge x =_{\text{E}(Q)} y \in \mu)\\ & \Rightarrow x\sigma =_{\text{E}(Q)} y\sigma)))\\ & \wedge \exists \lambda'\colon \varsigma(Q,\omega) \xrightarrow{\lambda} \xrightarrow{\lambda'} (\mathcal{E}',Q'',\varphi',\sigma')\\ & \wedge \text{dom}(\mu) \subseteq \text{dom}(\sigma'))\end{aligned}$$

This statement requires that for every trace λ of a process Q, two things hold: First, if two variables have been assigned a value at the end of λ, and these variables are required to be equal by the set-up guarantee μ, then the two values have to have a value equal in the equational theory $\text{E}(Q)$ of Q. Second, the extended process to which λ leads has to enable that, if executed further, all variables appearing in μ have been assigned a value. However, this only has to hold if none of the corruptible variables has been leaked yet – in this case, we cannot get a guarantee on the variable values anyways.

If a protocol is modeled as a partial process, it will often still be useful to analyze it without having to specify a concrete protocol to complete it; a generic process that just sets up variables to satisfy a set-up guarantee suffices for that. Such a process only needs the ability to bind variables with a name restriction and to assign terms to variables.

Definition 4.8 (Executable)**.** *Let μ be a set-up guarantee, let x and y be variables and let t be a term. Let $\langle\rangle$ be a process generated by the following grammar:*

$$\langle\rangle :\equiv\ 0 \mid \nu x.\langle\rangle \mid y := t \parallel P$$
$$P :\equiv\ 0 \mid y := t \parallel P$$

Then, if $\langle\rangle \vDash \mu$, $\langle\rangle$ is called an executable*; we write $\langle\mu\rangle\,.\,\texttt{exe}$.*

Note that although we use the assignment operator in this definition, we will assume in all following proofs that it has been resolved to synchronizing inputs and outputs; similarly, we assume that executables are unrolled such that they do not end in a parallel composition when placed in a partial process. Both transformations are described in the Full Version.

Example 4.9. *XP* needs an executable that guarantees $k_A =_{\mathrm{E}} k_B$ to work properly. This is easily implementable as $\langle k_A =_{\mathrm{E}} k_B \rangle .\,\mathtt{exe} :\equiv \nu k.k_A := k.k_B := k$. We can then use this executable to investigate $XP[\langle k_A =_{\mathrm{E}} k_B \rangle .\,\mathtt{exe}]$, which allows us to analyze the data exchange protocol for its desired properties.

5 Composition Results

We are now able to state and prove our composition result; we do so in two steps. First, we sketch a theorem that serves as the most important lemma in our later proof – and also generalizes a previous result from the literature. Afterwards, we state our main result on composition under dynamic corruption, discuss the requirements of the theorem and outline its proof. We omit the full proofs and some minor definitions due to space constraints; they can be found in the Full Version of this paper, available in [38].

5.1 Disjoint Composition

We first give a brief summary of our *disjoint composition* result: We prove that for two protocols that have no common equational theory and share neither names nor variables, every execution of their parallel composition consists of two executions of the individual protocols. We will need this in the proof of our main result.

Although the following theorem essentially serves as a lemma in the proof of our main result, we mention it here since it is a generalization of [33]. While only a limited set of primitives is allowed there (namely concatenation, symmetric and asymmetric encryption, signatures and hash functions), we prove disjoint composition for arbitrary primitives. The Full Version of this work contains some additional definitions.

Theorem 5.1. *Let P and Q be two processes built over $\Sigma(P)$ and $\Sigma(Q)$, respectively, such that both $\Sigma(P) \cap \Sigma(Q) = \emptyset$ and $\mathrm{vars}(P) \cap \mathrm{vars}(Q) = \emptyset$. Let α be a trace property defined over P or Q. Then, $P \vDash \alpha \wedge Q \vDash \alpha \implies P \,\|\, Q \vDash \alpha$.*

Proof. We provide a proof sketch here and the full proof in the Full Version. Since P and Q have nothing in common – they share neither primitives nor data nor communication channels – we can cleanly separate every execution of $P \,\|\, Q$ into one execution of P and one execution of Q. This is formalized in the Full Version. Thus, any *attack trace* λ of $P \,\|\, Q$ that does not satisfy α can be split into traces of P and Q, λ_P and λ_Q. As α must be defined over one of P or Q, we now have an attack trace on one of the two. □

5.2 Composition under Dynamic Corruption

We now move on to our main result. After providing a high-level overview of what the theorem achieves and what it requires, we will state it formally and give a proof sketch; the full proof can be found in the Full Version.

Consider two protocols P and Q. Intuitively, our composition theorem states that a property of P or Q is also a property of the composed protocol $P[Q]$, if some conditions hold.

Formally, we consider P to be a partial process by a process Q and put in some composition context C, where Q satisfies some set-up guarantee μ on the variables transferred from Q to P. We assume two sets of corruptible variables $\omega_P \subseteq \mathrm{vars}(C[P])$ and $\omega_Q \subseteq \mathrm{vars}(C[Q])$ and some trace property α that is defined over one of the dynamically corrupted processes $\varsigma(C[P[\langle \mu \rangle .\,\mathtt{exe}]], \omega_P)$ or $\varsigma(C[Q], \omega_Q)$. Our composition theorem states that, given some preconditions, it suffices to verify that α holds for $\varsigma(C[P[\langle \mu \rangle .\,\mathtt{exe}]], \omega_P)$ and $\varsigma(C[Q], \omega_Q)$ alone to show that α holds for $\varsigma(C[P[Q]], \omega_P \cup \omega_Q)$.

A typical instantiation of this would be a data exchange or messaging protocol P that uses some key exchange protocol Q to establish long-term keys. μ would state that the keys provided to the participating parties by Q are equal; ω_P and ω_Q could contain the long-term keys, and α could encode Forward Secrecy in case of leaked long-term keys. We investigate this case closer in 6.2.

We will now go through the theorem requirements, what they imply in practice, and how we can efficiently verify them.

1. Disjoint Primitives Most importantly, we require that P and Q use disjoint signatures, i.e., $\Sigma(P) \cap \Sigma(Q) = \emptyset$. This means that P and Q may not use the same cryptographic primitives. With this limitation, it is difficult for an adversary to trick a protocol into using messages sent by the other protocol to misuse its behavior, as neither of the two protocols has the tools – i.e., primitives – to process information sent out by the other protocol. To achieve this, practitioners should ensure that their protocols are properly domain-separated, e.g., by tagging the primitives they use; this is a best practice in security protocol design anyways. We discussed the implications of disjoint primitives more broadly in Section 3.2 already. As of now, this requirement needs to be manually verified, but since it is a static condition, this can be easily done.

2. Syntactic Requirements Additionally, there are more technical preconditions: The free variables of Q have to be a subset of the variables of C, and the free variables of P that are not also variables of C – i.e., the transferred variables – have to be a subset of the bound variables of Q. This is essentially a sanity check that P, Q and C can be properly composed, i.e., $C[P[Q]]$ does not contain any free variables anymore and the protocols only share variables that are properly initialized. This does not put any limitations on a protocol model, but requires more care when naming variables.

Also, if P and Q leak shared variables, both have to leak the same shared variables; otherwise the adversary could get

access to new information and, with that, new attack vectors through composition. Since in practice, two protocols that are meant to be composed should assume the same threat model, and thus, the same set of corruptible variables, anyways, this is a natural assumption.

As these requirements are syntactic conditions, they are trivial to verify manually.

3. Non-shared Channels Next, we require that P and Q do not use their shared variables to exchange any channel names. If they communicated over a channel received from the respective other protocol, they could potentially synchronize in unintended ways, which could lead to unintended behavior that is not present in P or Q alone and could thus enable new attacks. With the following formalization, this can be checked using a protocol verifier.

Definition 5.2. *Let P be a process built over a signature $\Sigma(P)$ with equational theory $\mathrm{E}(P)$, and let $x \in \mathrm{vars}(P)$. Then, x appears in place of a name in P if there is a subprocess $\mathbf{in}(v_c, y) \preceq P$ or $\mathbf{out}(v_c, t) \preceq P$ such that $v_c =_{\mathrm{E}(P)} x$. The set of all variables appearing in place of a name in P is denoted* $\mathrm{nv}(P) := \{x \in \mathrm{vars}(P) \mid \exists \mathbf{in}(v_c, y) \preceq P\colon v_c =_{\mathrm{E}(P)} x \vee \exists \mathbf{out}(v_c, t) \preceq P\colon v_c =_{\mathrm{E}(P)} x\}$.

4. Secrecy of Shared Variables We also require that P and Q do not reveal any shared variable on their own; this is defined as follows.

Definition 5.3 (Revealing a Variable)**.** *Let P be a process and let $x \in \mathrm{vars}(P)$ be a variable of P. Then P* reveals *x if there is a trace λ such that $P \xrightarrow{\lambda} (\mathcal{E}, P', \varphi, \sigma)$ and $\varphi \vdash x\sigma$.*

Why is this necessary? Intuitively, if some shared variable is revealed, even though it might not be corruptible, the adversary could use that value to construct new terms to send to the parties, changing their intended behavior. More technically, on one hand, we need this restriction to prove that all terms that are sent out from P or Q have their root in the respective signature; with this we can then prove that all synchronizations that take place in the composed protocol also happen in one of P or Q on their own. On the other hand, this restriction is necessary to make sure that shared variables can always be instantiated with fresh names if necessary. It can be checked automatically, using any protocol verifier that can reason about secrecy properties.

5. Freshness of Shared Variables Finally, shared variables also need to be *fresh*. A variable x is fresh relative to variables $y_1, \ldots, y_n$ if none of them have the same value as x. This is necessary to properly keep track of shared data, which we need to make general statements about information that has been leaked to an adversary. In practice, this requires that no copies are made of shared variables. Whenever we need a shared value, we should read it from the original variable and only store it in a new one if we change it sufficiently. This can also be checked with an automated protocol verifier.

Formally, freshness means that a process has to ensure that none of the variables relative to which x is supposed to be fresh is assigned a value equal to that of x.

Definition 5.4 (Fresh Variables)**.** *Let P be a process built over $\Sigma(P)$ with equational theory $\mathrm{E}(P)$. Let $x, y_1, \ldots y_n \in \mathrm{vars}(P)$ be variables. Then x is* fresh *relative to $y_1, \ldots, y_n$ if for every $P \Rightarrow (\mathcal{E}, P', \varphi, \sigma)$ and for every $i \in \{1, \ldots, n\}$, $x\sigma \neq_{\mathrm{E}(P)} y_i\sigma$.*

In summary, requirements 1 and 2 can easily be checked by hand, while a protocol verifier can check requirements 3 through 5. However, as we will see in Sections 6.2 and 6.3, all five requirements can be verified manually with little effort even for protocols the size of TLS 1.3 ECH.

With all requirements and definitions in place, we are now able to formally state our main theorem.

Theorem 5.5. *Let P be a partial process and Q be a process built over $\Sigma(P)$ and $\Sigma(Q)$, respectively, such that $\Sigma(P) \cap \Sigma(Q) = \emptyset$ and $\mathrm{names}(P) \cap \mathrm{names}(Q) = \emptyset$, with equational theories $\mathrm{E}(P)$ and $\mathrm{E}(Q)$. Let C be a composition context such that $\mathrm{fv}(P) \subseteq \mathrm{vars}(C)$, $\mathrm{fv}(Q) \subseteq \mathrm{vars}(C)$ and $\mathrm{trf}(C[P[Q]]) \subseteq \mathrm{bv}(Q)$. Let $\omega_P \subseteq \mathrm{vars}(C[P])$ and $\omega_Q \subseteq \mathrm{vars}(C[Q])$ be a subset of variables of P and Q, respectively, such that $\omega_P \cap \mathrm{sh}(C[P[Q]]) = \omega_Q \cap \mathrm{sh}(C[P[Q]])$. Let μ be a set-up guarantee such that $\varsigma(C[Q], \omega_Q) \Vdash \mu$ and $\mathrm{dom}(\mu) \subseteq \mathrm{trf}(C[P[Q]])$. Finally, let α be a trace property defined over either P or Q. Then, if neither $C[P[\langle\mu\rangle\,.\,\mathtt{exe}]]$ nor $C[Q]$ reveal any variable in $\mathrm{sh}(C[P[Q]])$, every $x \in \mathrm{sh}(C[P[Q]])$ is fresh relative to every $y \in \mathrm{vars}(C[P[Q]]) \setminus \mathrm{sh}(C[P[Q]])$ in $\varsigma(C[P[Q]], \omega_P \cup \omega_Q)$, and $\mathrm{nv}(C[P[Q]]) \cap \mathrm{sh}(C[P[Q]]) = \emptyset$:*

$$\varsigma(C[P[\langle\mu\rangle\,.\,\mathtt{exe}]], \omega_P) \vDash \alpha \wedge \varsigma(C[Q], \omega_Q) \vDash \alpha \implies \varsigma(C[P[Q]], \omega_P \cup \omega_Q) \vDash \alpha$$

Proof. Once again, we provide the full proof in the Full Version and give an outline here. The main goal is to show that every attack on the composed protocol already is an attack on one of the base protocols alone. For this, we assume some *attack trace* λ that violates α. In a first step, we prove that whenever a shared variable is leaked, it can instantly be leaked again without removing the attack. This procedure yields a *re-leaking trace* $\mathrm{rl}(\lambda)$.

We then show that $\mathrm{rl}(\lambda)$ is in static equivalence with a trace of the parallel composition of the base protocols, where variables have been renamed such that the protocols do not share any variables anymore. More formally, we prove that there is a trace λ' of the process $D :\equiv \varsigma(C[P[\langle\mu\rangle\,.\,\mathtt{exe}]], \omega_P)\{^x/_{x_P}\}_{x \in \mathrm{vars}(C[P[Q]])} \parallel \varsigma(C[Q], \omega_Q)\{^x/_{x_Q}\}_{x \in \mathrm{vars}(C[P[Q]])}$ such that $\lambda' \sim \mathrm{rl}(\lambda)$.

This is achieved via induction over the length of $\mathrm{rl}(\lambda)$. Building on an idea from [31], we construct a trace of D by assigning the now renamed variables in D the values of their original counterparts, but we replace all terms introduced by the respective other protocol with names. Since equal terms are replaced by the same name, this preserves equalities.

Now $\mathrm{rl}(\lambda)$ and λ' are in static equivalence, so we have an attack trace on D. However, D is constructed in a way that allows us to apply Theorem 5.1, concluding the proof. □

6 Applications

When analyzing security protocols, automated protocol verifiers may run out of memory or fail to produce a result within a reasonable timeframe. This is not a problem that can be universally solved by increasing the available resources, since security properties are generally undecidable [55].

However, by reducing the analysis of large protocols to smaller subprotocols, compositional analysis can often circumvent this problem. We will now demonstrate how our result enables compositional analysis, and, with that, the analysis of larger and more complex protocols.

First, we discuss the compatibility of our result with existing protocol verifiers, before illustrating the usage of our composition theorem with a Diffie-Hellman key exchange and a data exchange protocol. We then perform a compositional analysis of Forward Secrecy in TLS 1.3 within the scope of RFC 8446 and the ECH extension. Finally, we provide an example of how our result can be used to enable compositional reasoning in other scenarios.

6.1 Tool Compatibility

Although there are to the best of our knowledge no dedicated tools for compositional reasoning, traditional protocol verifiers can still be used with a composition theorem: Prove properties of individual protocols with tool support, and derive properties of the composed protocol by applying the theorem. Our result is directly applicable to two state-of-the-art verification tools: ProVerif [14, 19] and SAPIC+ [30]. Both model protocols in a dialect of the Applied π-Calculus. Even though the ProVerif dialect is richer than the core Applied π-Calculus we use here, they are actually equivalent. In [16] a translation from the former to the latter is introduced and proven to produce equivalent traces (cf. Chapter 4).

Similarly, it is shown in [30] that a SAPIC+ dialect process and its translation to ProVerif dialect produce the same traces. Since the latter is equivalent to core Applied π-Calculus, by transitivity, core Applied π-Calculus is equivalent to the SAPIC+ dialect.

This leaves the properties: Are all properties expressible in the tools covered by our result? We did not fix any specification language but consider trace properties to be sets of traces closed under static equivalence. Trace sets are encoded by *queries* in ProVerif and by *trace formulas* in SAPIC+; if a query or trace formula is defined such that it is closed under static equivalence, it is therefore covered by Theorem 5.5.

In conclusion, our result can be applied to protocols and trace properties analyzed with both ProVerif and SAPIC+; however, since Theorem 5.5 only covers trace properties, we cannot use it for a compositional analysis of equivalence properties. We make use of this observation in our case studies we present in the next sections.

6.2 Diffie-Hellman and a Data Exchange

Recall the data exchange protocol XP we modeled over the course of Section 4.2. It models the repeated transfer of a data item from one party to another.

$$\begin{aligned} XP :\equiv !\_.(\nu data.\mathbf{out}(xc, \mathtt{senc}(data, k_A^{\mathrm{sess}})) \\ \| \mathbf{in}(xc,t).x := \mathtt{sdec}(t, k_B^{\mathrm{sess}})) \end{aligned}$$

As you can see, XP contains a hole; this is since we do not care how the two session keys k_A^{sess} and k_B^{sess} are created. The only requirement is that they are equal, i.e., they are covered by the set-up guarantee $k_A^{\mathrm{sess}} =_{\mathrm{E}} k_B^{\mathrm{sess}}$.

We also modeled a Diffie-Hellman key exchange DH that satisfies this set-up guarantee, which we slightly extend to an authenticated version: The two parties sign their respective half keys, and only process the other party's half key if its signature is valid. For this, we use the symbols $\mathtt{sign}\langle 2\rangle$ for signing a message with a private key, $\mathtt{check}\langle 2\rangle$ for verifying the signature of a message with a public key, and $\mathtt{get}\langle 1\rangle$ for retrieving the message from a signature. The corresponding equational theory consists of the equations $\mathtt{check}(\mathtt{sign}(m,k),\mathtt{pk}(k)) = m$ and $\mathtt{get}(\mathtt{sign}(m,k)) = m$.

$$\begin{aligned} DH :\equiv (\nu a.(\mathbf{out}(c, \mathtt{sign}(\mathtt{mexp}(g,a), sk_A)) \,\|\, \mathbf{in}(d,B). \\ \mathbf{if}\ \mathtt{check}(B, \mathtt{pk}(sk_B)) = \mathtt{get}(B) \\ \mathbf{then}\ k_A^{\mathrm{sess}} := \mathtt{mexp}(\mathtt{get}(B), a)\ \mathbf{else}\ 0)) \\ \|(\nu b.(\mathbf{out}(d, \mathtt{sign}(\mathtt{mexp}(g,b), sk_B)) \,\|\, \mathbf{in}(c,A). \\ \mathbf{if}\ \mathtt{check}(A, \mathtt{pk}(sk_A)) = \mathtt{get}(A) \\ \mathbf{then}\ k_B^{\mathrm{sess}} := \mathtt{mexp}(\mathtt{get}(A), b)\ \mathbf{else}\ 0)) \end{aligned}$$

For DH to work properly, we need the two private keys sk_A and sk_B, from which we can derive the respective public keys, and a generator g. We put these values into the composition context as they are pre-shared values agreed on beforehand.

$$C :\equiv \nu sk_A.\nu sk_B.\nu g.\_$$

Finally, recall the equational theories for XP and DH, which specify how symmetric encryption and modular exponentiation work. To the latter, we add the two equations modeling signatures. The key derivation function in our model does not need an equational theory; we assume it to essentially work like a hash function.

$$\begin{aligned} \mathrm{E}(XP) &:= \{\mathtt{sdec}(\mathtt{senc}(x,k),k) = x\} \\ \mathrm{E}(DH) &:= \{\mathtt{mexp}(\mathtt{mexp}(g,a),b) = \mathtt{mexp}(\mathtt{mexp}(g,b),a), \\ &\qquad \mathtt{check}(\mathtt{sign}(m,k),\mathtt{pk}(k)) = m, \\ &\qquad \mathtt{get}(\mathtt{sign}(m,k)) = m\} \end{aligned}$$

In combination, we now have $C[XP[DH]]$ and the associated equational theories as a monolithic model of the data exchange protocol. We want to know if it provides Forward Secrecy. Theorem 5.5 allows us to ignore the specific key exchange, so instead of analyzing $C[XP[DH]]$, we can define Forward Secrecy over the corrupted version of $C[DH]$.

$$\begin{aligned} FS := \{ &(tr, \nu\mathcal{E}.\varphi) \in \mathrm{tr}(\varsigma(C[DH], \{sk_A, sk_B\})) \mid \\ &\exists i\colon w_i \in \mathrm{dom}(\varphi) \wedge w_i\varphi = \mathtt{sign}(\mathtt{mexp}(g,b), sk_B) \\ &\wedge \varphi \vdash k_A^{\mathrm{sess}} \\ \Rightarrow &\exists j\colon w_j \in \mathrm{dom}(\varphi) \wedge (w_j\varphi = sk_A \vee w_j\varphi = sk_B) \wedge j < i\} \end{aligned}$$

This states Forward Secrecy of k_A^{sess}: If the message containing B's half key has been sent from B to A (i.e., A should be able to calculate k_A^{sess}), and if the adversary is able to deduce k_A^{sess}, then at least one of the secret long-term keys sk_A and sk_B must have been leaked at an earlier point in time. Forward Secrecy of k_B^{sess} is defined analogously.

Before we apply Theorem 5.5 to XP and DH, we have to ensure that it is applicable. First, FS is closed under static equivalence: Informally, every leak is a visible action on the trace and reflected in the adversary knowledge – which both have to be equal for two traces to be statically equivalent. Next, we argue that the requirements of the theorem hold.

1. Disjoint Primitives The signatures of XP, $\Sigma(XP) = \{\mathtt{senc}\langle 2\rangle, \mathtt{sdec}\langle 2\rangle, \mathtt{kdf}\langle 1\rangle\}$, and DH, $\Sigma(DH) = \{\mathtt{mexp}\langle 2\rangle, \mathtt{sign}\langle 2\rangle, \mathtt{check}\langle 2\rangle, \mathtt{get}\langle 1\rangle\}$, are clearly disjoint.

2. Syntactic Requirements We have to check whether $\mathrm{fv}(DH) = \{sk_A, sk_B, g\} \subseteq \mathrm{vars}(C)$ and $\mathrm{fv}(XP) \setminus \mathrm{vars}(C) = \{k_A^{\mathrm{sess}}, k_B^{\mathrm{sess}}\} \subseteq \{A, B, k_A^{\mathrm{sess}}, k_B^{\mathrm{sess}}\} = \mathrm{bv}(DH)$, both of which holds. We also analyze both DH and XP under the assumption that sk_A and sk_B can be leaked.

3. Non-shared Channels Neither XP nor DH have variables in place of a channel name.

4. Secrecy of Shared Variables Since sk_A, sk_B and g only occur in positions where they are not retrievable through the equational theory, they are not revealed in $C[XP[\langle k_A^{\mathrm{sess}} =_{\mathrm{E}} k_B^{\mathrm{sess}}\rangle.\mathtt{exe}]]$ and $C[DH]$. k_A^{sess} and k_B^{sess} remaining unknown to the adversary is an essential property of DH, and in XP they, too, are not retrievable through the equational theory. None of the shared values are thus revealed.

5. Freshness of Shared Variables The shared variables are sk_A, sk_B, g, k_A^{sess} and k_B^{sess}. The former three are used only in DH, and only to create new terms unequal to the respective variables. The latter two are used in the same way in XP, while the values they get assigned in DH are by construction different to those of the non-shared variables A and B.

Thus, all requirements of Theorem 5.5 are satisfied, and we can use it to conclude that $\varsigma(C[XP[DH]], \{sk_A, sk_B\}) \vDash FS$ – if we can show that $\varsigma(C[DH], \{sk_A, sk_B\}) \Vdash \{k_A^{\mathrm{sess}} =_{\mathrm{E}} k_B^{\mathrm{sess}}\}$ and $\varsigma(C[DH], \{sk_A, sk_B\}) \vDash FS$ hold. We do so in a ProVerif model, which we provide in [38][1]. As a sanity check, we also perform a monolithic analysis of $\varsigma(C[XP[DH]], \{sk_A, sk_B\})$ and show that it does in fact provide FS.

[1] `/dh/dh_auth.pv` in the `secure_composition_under_dynamic_corruption_case_studies.zip` archive

6.3 A Compositional Analysis of Forward Secrecy in TLS 1.3 ECH

TLS is one of the most widely used protocols and crucial for protecting internet communication. It is also one of the most thoroughly examined protocols, with automated verification playing a major rule in the standardization of TLS 1.3. It solved several privacy issues as compared to TLS 1.2, but important metadata, including the intended communication partner of the client, is still visible in plaintext. To mitigate these issues, the Encrypted Client Hello (ECH) extension is currently under standardization [59].

We will now do a compositional analysis of Forward Secrecy of the Master Secret in TLS 1.3 ECH to illustrate how Theorem 5.5 can be used to improve automated analysis. For this, we split TLS in two subprotocols and a composition context such that our result is applicable.

A TLS 1.3 ECH handshake can be roughly divided into three phases: The *Key Exchange*, the *Authentication Phase* and the *Data Exchange*. The Key Exchange performs a Diffie-Hellman key exchange and derives several keys, including the *master secret* as the basis for all upcoming session keys. During the Authentication Phase, the server authenticates itself to the client and vice versa. The Data Exchange finally transfers application data, encrypted with session keys derived from the keys established earlier. Additionally, both client and server have some pre-established keys.

Put in the terminology of Theorem 5.5, we can now model the Key Exchange up to the derivation of the master secret as a process KEx, while the Authentication Phase and Data Exchange form a partial process $AD :\equiv \_.Auth.Data$. Pre-established keys are set up in a composition context C.

Then, TLS is modeled by $C[AD[KEx]]$, and we can analyze whether Forward Secrecy holds for the master secret on KEx alone if the private ECH key dk_F of the server is corruptible. We write $FS(ms)$.

While we could split TLS a few steps into AD, we would gain less from a compositional analysis since we would have to analyze a larger part of the protocol.

We can now use Theorem 5.5 to perform this compositional analysis: As already argued in Section 6.2, Forward Secrecy is closed under static equivalence; we also observe that KEx can be abstracted by an executable satisfying the set-up guarantee $\mu := \{ms = k_{h,c}, ms = k_{h,s}, ms = k_{m,c}, ms = k_{m,s}\}$, which equates all the keys derived at the end of KEx. Finally, we will go through the theorem requirements.

1. Disjoint Primitives KEx uses HPKE, modular exponentiation, and three different KDFs $\mathtt{kdf}_0$, $\mathtt{kdf}_{hs}$ and $\mathtt{kdf}_{ms}$; AD uses symmetric encryption, signatures, MACs, and other KDFs $\mathtt{kdf}_k$ and $\mathtt{kdf}_{psk}$. Note that all of these KDFs are configured to produce different keys even for the same input, so

they can safely be modeled as different functions.

2. Syntactic Requirements The free variables of KEx, i.e., the public-private key pairs, are bound by C; the remaining free variables of AD (all derived keys, e.g., ms) are bound by KEx. Finally, dk_F is corruptible in both parts of TLS.

3. Non-shared Channels No shared variable is used in the place of a channel name, and since shared variables are fresh with respect to non-shared variables, none of their values is used as a channel name.

4. Secrecy of Shared Variables Pre-shared variables, namely sk_C, pk_C, $psk_{C,S}$, sk_S, pk_S, sk'_S, pk'_S, dk_F, and ek_F, are only used to encrypt or decrypt terms and thus not deducible; this is also true for transferred variables in AD, i.e., the keys ms, $k_{h,c}$, $k_{h,s}$, $k_{m,c}$, and $k_{m,s}$. In KEx, transferred variables cannot be deduced since all keys have the same value as ms and Forward Secrecy holds for ms.

5. Freshness of Shared Variables All pre-shared variables are freshly generated through a name restriction, and the transferred variables are constructed such that they do not share a value with a non-shared variable. Additionally, no value of a shared variable is copied into another variable. Shared variables are thus fresh.

An extensive analysis of TLS 1.3 ECH using ProVerif has been performed in [13]. The authors analyze 14 security properties in 480 different scenarios of different protocol functionalities and behaviors, agents involved, and key compromise scenarios. However, they can only prove 63% of the properties due to ProVerif running out of time or memory.

We base our analysis on the most recent version of the ProVerif model used for this analysis[2]; it covers the scope of RFC 8446 and the ECH extension, i.e., the entirety of TLS itself, but not attacks on the wider TLS ecosystem or different versions of TLS. We extend the model with a forward secrecy query on the master secret and implement a composition mode, which allows us to measure the improvements of a compositional verification approach. The composition mode corresponds to an analysis of $\varsigma(C[KEx], \{dk_F\})$; the standard mode analyzes the full protocol monolithically, corresponding to $\varsigma(C[AD[KEx]], \{dk_F\})$.

In this updated model, we evaluate both queries in a representative subset of 32 relevant scenarios and compare the runtime of a compositional and monolithic analysis with the same timeout of 48 hours as in [13]. Our server features an Intel® Xeon® CPU E5-4650L with 2.60 GHz and 1 TB of RAM. We summarize the results here; the complete data and a more extensive discussion can be found in Appendix A.

Within 48 hours, only 28 monolithic analyses terminate, but all 32 compositional ones. In the scenarios in which both analyses terminate, the compositional is 71 % faster on average, with runtime decreases of up to 97 %, and needs 86 % less memory, with a range of 57 % to 99 %. We also observe that these improvements are fairly consistent and scale linearly, with the shortest analysis reduced by 45 % from 11 min to 6 min, and the longest by 88 % from 29 h 43 min to 03 h 31 min. Therefore, we expect these improvements to also scale to more extensive analyses of both TLS 1.3 ECH and other protocols. The model with our changes, the benchmarking script, and the raw data of our tests is available in [38][3].

[2] Last updated September 1st, 2022: https://gitlab.cs.ox.ac.uk/vinval/ech-tls/-/tree/9fd6b2a0523087d41ba480c6180b9da92833ca50

6.4 Iterative Composition

Theorem 5.5 can not only be used for compositional analysis, but it can also serve as the basis for other composition results. In this section, we give an example for such a corollary and prove that we can securely compose protocols with a chain-like structure arbitrarily often with themselves, given that they are properly *tagged*.

We consider a process tagged if every instantiation of it uses primitives that add a unique *tag* to its outputs. Then the messages of a specific process can only be used by that specific process. In its most basic form, tagging means that instead of using a function f, we use a function f_i. As discussed in Section 3.2, the decisive factor here is that the functions produce different outputs from the same input.

Our corollary now assumes a protocol that works by repeatedly calling upon itself and that is tagged differently for every iteration. Then it suffices to prove that a trace property holds for one iteration to conclude that the property holds for an arbitrary number of iterations.

How do we prove this? The central observation is that a protocol running n iterations is nothing more than a protocol running $n-1$ iterations composed with a protocol running a single iteration. Since the protocol is tagged, every iteration satisfies the disjointness criteria to inductively apply Theorem 5.5. Then we only need to verify a single iteration to prove a property for an arbitrary number of runs.

This approach could be used in the verification of many practically relevant protocols. One-Time Password protocols often rely on a hash chain, and the Double Ratchet used in the Signal protocol essentially calls itself repeatedly. The approach can also be applied to TLS 1.3: The session keys are derived from, amongst others, transcripts that strictly increase in size for every iteration. Given the collision-resistance of the hash and key derivation functions, we can consider the transcript length as a tag for the top-level key derivation functions and can thus consider them as disjoint KDF functions.

We only state the corollary and give all definitions and the proof in the Full Version of this work.

Corollary 6.1. *Let P be an iterable partial process built over $\Sigma(P)$ with an equational theory $\mathrm{E}(P)$. Let Q be an executable. Let $\omega \subseteq \mathrm{vars}(P[Q])$ be a subset of variables of $P[Q]$. Let μ be a set-up guarantee such that $\varsigma(Q,\omega) \Vdash \mu$, $\varsigma(P[Q],\omega) \Vdash \mu$*

[3] /tls_ech/ in the secure_composition_under_dynamic_corruption_case_studies.zip archive

and $\mathrm{dom}(\mu) \subseteq \mathrm{trf}(P[Q])$. *Finally, let* α *be a trace property defined over* $P[Q]$. *Then, if* $P[Q]$ *does not reveal any variable in* $\mathrm{sh}(P[Q])$, *every* $x \in \mathrm{sh}(P[Q])$ *is fresh relative to every* $y \in \mathrm{vars}(P[Q]) \setminus \mathrm{sh}(P[Q])$ *in* $\varsigma(P[Q], \omega_P \cup \omega_Q)$, *and* $\mathrm{nv}(P[Q]) \cap \mathrm{sh}(P[Q]) = \emptyset$:

$$\varsigma(P[Q], \omega) \vDash \alpha \Rightarrow \varsigma(P^n[Q_0], \omega^n) \vDash \alpha^n$$

7 Conclusion and Future Work

We have proven that protocols can be securely composed in the presence of an adversary capable of dynamic corruption and retain their trace properties, as long as they use a disjoint set of cryptographic primitives. This result enabled us to perform a compositional analysis of Forward Secrecy in TLS 1.3 ECH. Notably, we were able to do so with a generic composition theorem that has not been tailored to any specific protocol but fits the framework of two major protocol verifiers, ProVerif and SAPIC+.

In the future, we intend to advance our work in three directions. First, we plan to further demonstrate the usefulness of our result through a large-scale case study. An obvious choice is to analyze TLS more extensively, but other large-scale protocols like SPDM are also viable candidates.

Second, we believe that our composition result can be useful in protocol analysis beyond merely speeding up automated verification. One example would be to use the contraposition of our main theorem for locating vulnerabilities in insecure protocols: Given an attack trace on a protocol and a separation of the protocol into two subprotocols that satisfy the requirements of our theorem, we can conclude that the cause for the vulnerability is in one of the sub-protocols alone. We plan to investigate this and identify other non-standard applications of composability.

Finally, while we have seen that our core composition result is already applicable to real-world protocols, its requirement of disjoint signatures will not be met by some deployed protocols. We plan to investigate alternative notions of disjointness to further increase the applicability of our theorem.

Ethical Considerations

In this work, we propose a methodology for compositional symbolic analysis, applicable to real-world cryptographic protocols with complex security properties. Specifically, we develop a composition result that holds even when considering adversary models that enable dynamic corruption of protocols.

Our methodology consists of introspection and mathematical proof, and is intended for improving the efficiency of symbolic analysis. However, our result is purely constructive: it can be used to prove that security properties hold, but does not make it easier or faster to discover attacks. Therefore, we believe that there are no inherent ethical conflicts related to the publication of this research, despite it having a direct and impactful benefit to secure protocol development and verification.

Nonetheless, we identify three groups of stakeholders that could potentially be, directly or indirectly, affected by our research. In increasing size, these are: (1) the researchers involved in this work, (2) people, groups, and organizations interacting with protocol designs (e.g., industry, researchers, governments, or criminal organizations), and (3) members of the general public.

With respect to the researchers involved in this work, we see no ethical considerations. Our work builds upon previous methodologies, improving the robustness and efficiency of complex proofs. Since we do not introduce a new research methodology, we do not see unfavorable effects on researchers adapting the same methodology in the future.

People, groups and organizations can interact in two roles with security protocol designs: Either by (1) creating new, robust designs that provide certain security guarantees, or (2) attempting to discover vulnerabilities in existing designs. In the first scenario, the benefits of our work are purely beneficial and can only improve the efficiency of the development process. In the second scenario, our approach does not introduce new risks since it cannot be used to uncover attacks, as previously explained.

The above considerations propagate to members of general society. If they are targeted by malicious actors, they will benefit from more robust protocol designs where these are protecting their rights and interests. Our research results can contribute to this protection. We therefore believe that the benefits of our result heavily outweigh its potential risks.

Open Science

We publicly provide all artifacts of this work in a public Figshare repository [38]. This includes a long version of this paper, which contains all proofs omitted here due to space constraints, as well as the ProVerif models of both our case studies. The evaluation data of our TLS 1.3 ECH case study can be found in Appendix A of this work.

References

[1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The Applied Pi Calculus: Mobile values, new names, and secure communication, 2016.

[2] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL01, pages 104–115, New York, NY, USA, January 2001. ACM.

[3] Suzana Andova, Cas Cremers, Kristian Gjøsteen, Sjouke Mauw, Stig F. Mjølsnes, and Saša Radomirović. A framework for compositional verification of security protocols. *Information and Computation*, 206(2–4):425–459, February 2008.

[4] Myrto Arapinis, Vincent Cheval, and Stephanie Delaune. Verifying privacy-type properties in a modular way. In *2012 IEEE 25th Computer Security Foundations Symposium*, pages 95–109, Cambridge, MA, USA, June 2012. IEEE.

[5] Myrto Arapinis, Vincent Cheval, and Stéphanie Delaune. *Composing Security Protocols: From Confidentiality to Privacy*, pages 324–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[6] Various Authors. Thread on banking industry concerns about TLS 1.3. IETF TLS Mailing List (tls@ietf.org), September 2016. https://mailarchive.ietf.org/arch/browse/tls/?gbt=1&index=CzjJB1g0uFypY8UDdr6P9SCQBqA.

[7] Michael Backes, Markus Dürmuth, Dennis Hofheinz, and Ralf Küsters. *Conditional Reactive Simulatability*, pages 424–443. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[8] Michael Backes, Birgit Pfitzmann, and Michael Waidner. *A General Composition Theorem for Secure Reactive Systems*, pages 336–354. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[9] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, December 2007.

[10] Karthikeyan Bhargavan, Abhishek Bichhawat, Quoc Huy Do, Pedram Hosseyni, Ralf Küsters, Guido Schmitz, and Tim Würtele. DY*: A modular symbolic verification framework for executable cryptographic protocol code. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 523–542, Vienna, Austria, September 2021. IEEE.

[11] Karthikeyan Bhargavan, Abhishek Bichhawat, Pedram Hosseyni, Ralf Küsters, Klaas Pruiksma, Guido Schmitz, Clara Waldmann, and Tim Würtele. *Layered Symbolic Security Analysis in DY**, pages 3–21. Springer Nature Switzerland, Cham, 2024.

[12] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 483–502, San Jose, CA, USA, May 2017. IEEE.

[13] Karthikeyan Bhargavan, Vincent Cheval, and Christopher Wood. A symbolic analysis of privacy for TLS 1.3 with encrypted client hello. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 365–379, New York, NY, USA, November 2022. ACM.

[14] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[15] Bruno Blanchet. A computationally sound mechanized prover for security protocols. Cryptology ePrint Archive, Paper 2005/401, 2005.

[16] Bruno Blanchet. Modeling and verifying security protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends® in Privacy and Security*, 1(1–2):1–135, 2016.

[17] Bruno Blanchet. Symbolic and computational mechanized verification of the ARINC823 avionic protocols. In *30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 68–82, Santa Barbara, CA, USA, August 2017. IEEE.

[18] Bruno Blanchet. Composition theorems for CryptoVerif and application to TLS 1.3. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 16–30, Oxford, UK, July 2018. IEEE.

[19] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. ProVerif 2.05: Automatic cryptographic protocol verifier, user manual and tutorial, 2023.

[20] Florian Böhl and Dominique Unruh. Symbolic universal composability. *Journal of Computer Security*, 24(1):1–38, December 2014.

[21] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. *Universal Composition with Responsive Environments*, pages 807–840. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[22] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. *iUC: Flexible Universal Composability Made Simple*, pages 191–221. Springer International Publishing, Cham, 2019.

[23] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, Newport Beach, CA, USA, 2001. IEEE, IEEE.

[24] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. *Universally Composable Security with Global Setup*, pages 61–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[25] Ran Canetti and Tal Rabin. *Universal Composition with Joint State*, pages 265–281. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[26] Sofía Celi, Jonathan Hoyland, Douglas Stebila, and Thom Wiggers. *A Tale of Two Models: Formal Verification of KEMTLS via Tamarin*, pages 63–83. Springer Nature Switzerland, Cham, 2022.

[27] Vincent Cheval, Véronique Cortier, and Alexandre Debant. Election verifiability with ProVerif. In *Proceedings of the 36th IEEE Computer Security Foundations Symposium (CSF'23)*, pages 43–58, Dubrovnik, Croatia, July 2023. IEEE.

[28] Vincent Cheval, Veronique Cortier, and Bogdan Warinschi. Secure composition of PKIs with public key protocols. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 144–158, Santa Barbara, CA, USA, August 2017. IEEE.

[29] Vincent Cheval, Véronique Cortier, and Eric le Morvan. Secure refinements of communication channels. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*, pages 575–589, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[30] Vincent Cheval, Charlie Jacomme, Steve Kremer, and Robert Künnemann. SAPIC+: protocol verifiers of the world, unite! In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3935–3952, Boston, MA, August 2022. USENIX Association.

[31] Stefan Ciobâca and Véronique Cortier. Protocol composition for arbitrary primitives. In *2010 23rd IEEE Computer Security Foundations Symposium*, pages 322–336, Edinburgh, UK, July 2010. IEEE.

[32] Hubert Comon, Charlie Jacomme, and Guillaume Scerri. Oracle simulation: A technique for protocol composition with long term shared secrets. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, pages 1427–1444, New York, NY, USA, October 2020. ACM.

[33] Véronique Cortier and Stéphanie Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, October 2008.

[34] Cas Cremers, Alexander Dax, and Aurora Naska. Formal analysis of SPDM: Security protocol and data model version 1.2. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 6611–6628, Anaheim, CA, USA, 2023. USENIX Association.

[35] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1773–1788, New York, NY, USA, 2017. Association for Computing Machinery.

[36] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 470–485, San Jose, CA, USA, May 2016. IEEE.

[37] Cas Cremers, Benjamin Kiesl, and Niklas Medinger. A formal analysis of IEEE 802.11's WPA2: Countering the kracks caused by cracking the counters. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1–17, Berkeley, CA, United States, August 2020. USENIX Association.

[38] Cas Cremers, Erik Pallas, and Aleksi Peltonen. Secure protocol composition under dynamic corruption: Models and proofs. Figshare repository, December 2025. ProVerif Models for the DHKE and TLS 1.3 ECH Case Studies, and Full Version of the Paper with all Proofs. https://doi.org/10.6084/m9.figshare.29958290.

[39] Stéphanie Delaune, Steve Kremer, and Olivier Pereira. Simulation based security in the Applied Pi Calculus. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 169–180, Dagstuhl, Germany, 2009. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[40] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Composition of password-based protocols. In *2008 21st IEEE Computer Security Foundations Symposium*, pages 239–251, Pittsburgh, PA, USA, 2008. IEEE.

[41] Joshua Gancher, Sydney Gibson, Pratap Singh, Samvid Dharanikota, and Bryan Parno. OWL: Compositional verification of security protocols via an information-flow type system. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1130–1147, San Francisco, CA, USA, May 2023. IEEE.

[42] Sébastien Gondron and Sebastian Mödersheim. Vertical composition and sound payload abstraction for stateful protocols. In *2021 IEEE 34th Computer Security*

Foundations Symposium (CSF), pages 1–16, Dubrovnik, Croatia, June 2021. IEEE.

[43] Mike Graf, Ralf Küsters, and Daniel Rausch. AUC: Accountable universal composability. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1148–1167, San Francisco, CA, USA, May 2023. IEEE.

[44] Thomas Groß and Sebastian Mödersheim. Vertical protocol composition. In *2011 IEEE 24th Computer Security Foundations Symposium*, pages 235–250, Cernay-la-Ville, France, June 2011. IEEE.

[45] J.D. Guttman and F.J.F. Thayer. Protocol independence through disjoint encryption. In *Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13*, CSFW-00, pages 24–34, Cambridge, UK, 2000. IEEE Comput. Soc.

[46] Joshua D. Guttman. *Cryptographic Protocol Composition via the Authentication Tests*, pages 303–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[47] Christoph G. Günther. *An Identity-Based Key-Exchange Protocol*, pages 29–37. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989.

[48] Dennis Hofheinz and Victor Shoup. GNUC: A new universal composability framework. *Journal of Cryptology*, 28(3):423–508, October 2013.

[49] Ralf Küsters, Max Tuengerthal, and Daniel Rausch. The IITM model: A simple and expressive model for universal composability. *Journal of Cryptology*, 33(4):1461–1584, June 2020.

[50] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, November 1995.

[51] Gavin Lowe. *Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR*, pages 147–166. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[52] Moxie Marlinspike and Trevor Perrin. The X3DH key agreement protocol. Technical report, Signal Technology Foundation, November 2016. Last accessed March 28th, 2025.

[53] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, RISP-94, pages 79–93, Oakland, CA, USA, 1994. IEEE Comput. Soc. Press.

[54] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1996.

[55] J. Mitchell, A. Scedrov, N.A. Durgin, and P.D. Lincoln. Undecidability of bounded security protocols. In *Workshop on formal methods and security protocols*. Citeseer, 1999.

[56] Sebastian Mödersheim and Luca Viganò. *Secure Pseudonymous Channels*, pages 337–354. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[57] Sebastian Mödersheim and Luca Viganò. Sufficient conditions for vertical composition of security protocols. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, ASIA CCS '14, pages 435–446, New York, NY, USA, June 2014. ACM.

[58] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[59] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-25, Internet Engineering Task Force, June 2025. Work in Progress.

A Case Study Results

In [13], the authors develop a ProVerif model of TLS 1.3 (with the ECH extension) and analyze 14 security and privacy properties of the protocol, including secrecy, anonymity, and authentication. For each property, they generate a total of 480 variations of the model, which implement different functionalities, attacker models, and behaviors.

We extend the model with a forward secrecy query on the master secret and implement a composition mode, which allows us to measure the improvements of a compositional verification approach. For our benchmark, we use a representative subset of 32 scenarios, including all the different functionalities provided in the original model. We exclude scenarios that are not relevant for the comparison, as they, e.g., do not allow ECH clients in the execution. The results of our benchmark are shown in Table 1.

We observe substantial speed-ups in all but four cases and a reduced memory usage of at least 57 % in all cases, sometimes going as far as 99 %. We especially see high improvements in scenarios that allow for complex interactions after the initial handshake is finished. This is since our composition result allows us to cut out post-handshake interactions in a compositional analysis.

We also observe that several scenarios need similar or equal resources for a compositional analysis. The differences in the respective scenarios are only relevant after the initial handshake is concluded, and they therefore do not affect

Table 1: Performance Comparison

F refers to different optional functionalities (e.g., early data, HRR); **S** determines the compromised keys, cipher suits, and groups; **A** specifies the allowed agent behaviors (e.g., PSK, client certificates); **B** specifies the instantiated agents types (ECH or non-ECH). A detailed description of all scenarios is available in `/tls_ech/README.md` of [38]. If an analysis exceeds a runtime of 48 h, we note this by *timeout* and give the memory used at this point in time.

				monolithic		**compositional**		**improvement**	
F	**S**	**B**	**A**	time (hh:mm)	memory (GB)	time (hh:mm)	memory (GB)	time	memory
2	2	2	2	00:11	02.64	0:06	0.66	45 %	75 %
2	2	2	4	00:16	03.49	0:07	0.76	56 %	78 %
2	2	3	2	00:48	08.08	0:06	0.79	88 %	90 %
2	2	3	4	01:22	12.25	0:08	0.79	90 %	94 %
3	2	2	2	00:12	02.64	0:06	0.66	50 %	75 %
3	2	2	4	00:17	03.49	0:07	0.76	59 %	78 %
3	2	3	2	00:49	08.08	0:06	0.76	88 %	91 %
3	2	3	4	01:25	12.25	0:07	0.87	92 %	93 %
4	2	2	2	01:18	12.25	0:06	0.66	92 %	95 %
4	2	2	4	01:41	18.66	0:07	0.76	95 %	96 %
4	2	3	2	03:43	37.51	0:06	0.66	97 %	98 %
4	2	3	4	05:03	65.67	0:08	0.76	97 %	99 %
5	2	2	2	01:19	12.25	0:06	0.66	92 %	95 %
5	2	2	4	01:41	18.66	0:07	0.76	93 %	96 %
5	2	3	2	03:45	37.51	0:06	0.66	97 %	98 %
5	2	3	4	05:00	65.67	0:08	0.76	97 %	99 %
7	2	2	2	02:55	10.66	2:47	4.62	05 %	57 %
7	2	2	4	03:22	12.25	3:22	4.62	00 %	62 %
7	2	3	2	14:10	32.65	2:50	4.62	80 %	86 %
7	2	3	4	16:29	43.10	3:32	5.31	79 %	88 %
8	2	2	2	02:51	10.66	2:50	4.62	01 %	57 %
8	2	2	4	03:22	12.25	3:30	4.62	−04 %	62 %
8	2	3	2	14:15	32.65	2:54	4.62	76 %	86 %
8	2	3	4	16:59	43.10	3:33	5.31	79 %	88 %
9	2	2	2	25:01	49.53	2:51	4.62	89 %	91 %
9	2	2	4	28:51	65.67	3:30	4.62	88 %	93 %
9	2	3	2	timeout	132.06	2:54	4.62	N/A	N/A
9	2	3	4	timeout	151.72	3:41	5.31	N/A	N/A
10	2	2	2	25:53	56.92	2:52	4.62	89 %	92 %
10	2	2	4	29:43	65.67	3:31	4.62	88 %	93 %
10	2	3	2	timeout	132.06	3:02	4.62	N/A	N/A
10	2	3	4	timeout	151.71	3:39	5.31	N/A	N/A
							∅	71 %	86 %

the compositional analysis. Consider for example the scenarios 2222 and 3222, which need exactly the same amount of time and memory for a compositional analysis. This is since the only difference between the **F** scenarios 2 and 3 is that 3 allows sending post-handshake data, while 2 does not. Post-handshake data, however, is not considered by the compositional analysis and therefore does not affect its performance.

The same mechanism explains the four scenarios 7222, 7224, 8222, and 8224. Here, compositional analysis is barely faster, exactly as fast or even slightly slower than a monolithic analysis. Since the scenarios consider only a very limited amount of post-handshake interaction, a compositional analysis has to consider very similar parts of TLS 1.3 as a monolithic analysis and therefore also needs a very similar amount of resources.

In summary, compositional analysis saves us 71 % of time and 86 % of memory on average, with most numbers consistently ranging from 60 % to 95 %. We therefore expect these results to scale to more extensive investigations of TLS 1.3 ECH as well as other complex protocols.