# Formalizing and Proving a Typing Result for Security Protocols in Isabelle/HOL

Andreas Viktor Hess and Sebastian Mödersheim
*DTU Compute*
*Technical University of Denmark*
*2800 Kongens Lyngby, Denmark*
*e-mails: {avhe,samo}@dtu.dk*

*Abstract*—**There are several works on the formalization of security protocols and proofs of their security in Isabelle/HOL; there have also been tools for automatically generating such proofs. This is attractive since a proof in Isabelle gives a higher assurance of the correctness than a pen-and-paper proof or the positive output of a verification tool. However several of these works have used a typed model, where the intruder is restricted to "well-typed" attacks. There also have been several works that show that this is actually not a restriction for a large class of protocols, but all these results so far are again pen-and-paper proofs. In this work we present a formalization of such a typing result in Isabelle/HOL. We formalize a constraint-based approach that is used in the proof argument of such typing results, and prove its soundness, completeness and termination. We then formalize and prove the typing result itself in Isabelle. Finally, to illustrate the real-world feasibility, we prove that the standard Transport Layer Security (TLS) handshake satisfies the main condition of the typing result.**

## I. Introduction

Proof assistants like Isabelle [20] allow us to formalize and check proofs with an almost "absolute" precision and reliability: once a theorem is proved, the chance of a mistake or hole in the proof is extremely low. This is very attractive for proofs of security protocols since security protocols are relatively small systems that are critical to our infrastructure and often have very subtle flaws that are easily overlooked. Paulson and Bella have proved security properties of numerous protocols in Isabelle and established a general paradigm of modeling protocols [21], [22], [5], [4].

Despite many automated proof tactics in Isabelle, conducting security proofs is still labor-intensive. There are many automated tools like ProVerif [6] that can verify most of the protocols of Paulson and Bella within minutes. However, an implementation mistake in such a tool can easily lead to a "false negative": no attack is found even though there is one. To get full reliability, one could directly model such a method in Isabelle, using Isabelle as a kind of interpreter. A more efficient way is to have automated tools generate proofs that Isabelle can check as in the works of Brucker and Mödersheim [8] and Meier et al. [16] (and by Goubault-Larrecq [12] for Coq).

Many of the mentioned works [21], [22], [5], [4], [8] rely on a typed protocol model that excludes that the attacker can send any ill-typed messages and thereby rules out any type-flaw attacks. In general, such a restriction to a typed model makes many aspects of the analysis easier. Most notably, in the abstract interpretation method used by [6], [8], protocol security is still undecidable, but under the restriction to a typed model the question becomes decidable.

There are in fact several results that show the *relative soundness* of a typed model if the protocol satisfies certain reasonable sufficient conditions: Heather et al. [13], Cortier and Delaune [9], Mödersheim [19], Arapinis and Duflot [2], and Almousa et al. [1]. Relative soundness means a result of the form: if a protocol (that satisfies the sufficient conditions) has an attack then it has a well-typed attack. So if we can verify that the protocol has no attack in the typed model (with whatever method), then it also has no attack in the untyped model. Closely related are relative soundness results from compositional reasoning, e.g., the mentioned works [9], [1] rely on typing results to obtain parallel composition results.

All these relative soundness results are so far classical pen-and-paper proofs. They contain complex proof arguments that, despite not being formalized out to the last detail, span easily ten pages (including all relevant formal definitions and lemmas with their proofs). It is not unlikely that such a result can have mistakes, from simple holes in a proof to wrong statements. Relying on such results bears some similarity to relying on unverified tools: we may wrongly accept a protocol as secure that actually is not (in the considered model). "Checking" the proof of such a result may be as complex a task as verifying a verification tool. The final and third parallel to verification tools is: there are often subtle differences in the protocol models and in the sufficient conditions that a casual user (who did not study the result in detail) may fail to notice. This bears the risk that in a hand-wavy fashion, one may accidentally apply a typing or compositionality result to protocols for which it does not hold.

This paper is a first step to overcome these problems of relative soundness results, by formalizing them in Isabelle. This allows us to use these results directly in security proofs like any other proved theorem. For instance, we may use any of the previous methods, manual or automatic, to prove the security of a protocol in the typed model, and then use the typing theorem to infer the result for the untyped model as a theorem entirely proved within Isabelle. Also this ensures that the theorem can only be applied if all the sufficient conditions are indeed satisfied (otherwise they will remain as open subgoals to prove). This therefore solves the problem of overlooking incompatible assumptions. The long-term goal is to establish a verification framework in Isabelle where different proof methods, manual and automatic, can be integrated with relative soundness results as far as they are applicable. This is interesting since probably no single verification approach is suited for all kinds of protocols.

In order to prove a typing result, one needs to make arguments of the form "in every step of an attack where the intruder sends something ill-typed, he may send something well-typed instead and the attack would work similarly." To make such arguments in a clear and precise way—avoiding handwavy and roundabout proofs—existing typing results [9], [19], [2], [1] use a popular verification technique that uses symbolic intruder constraints and that we simply refer to as the *lazy intruder*. This idea is originally used to cope with the infinity induced by the Dolev-Yao model in automated verification [17], [23], [3]: the intruder is lazy in the sense that he chooses parts of messages only in a demand-driven way, i.e., if they are necessary for a particular attack. One can use this technique in a different way for the typing results by showing (for protocols that satisfy some requirements) that the lazy intruder never makes ill-typed choices, and all type-flaw attacks are ill-typed choices of message parts that the lazy intruder did not instantiate. This allows one to conclude that, if there is a solution to the constraints, then there is a well-typed one. This is at the core of all typing results and we thus formalize the lazy intruder in Isabelle, including the proof that the reduction procedure for constraints is sound, complete, and terminating, because the typing result relies on this.

The main contribution of the paper is the formalization and proof in Isabelle/HOL of the relative typing result from [1]. Our entire Isabelle/HOL theory is approximately 8000 lines of code and takes about two minutes for Isabelle/HOL to load and verify on a standard machine. While some of the formalization could be streamlined it shows the complexity of formalizing and proving such a typing result when modeled with absolute formal precision. During this formalization effort we discovered a number of errors in [1]. We have fixed these problems by imposing some additional conditions on the class of protocols. We argue that these conditions are reasonable restrictions and still more liberal then those of other typing results. This is discussed in detail in Section VI. To illustrate the feasibility of our requirements, as a real-world case study, we prove in Isabelle that the Transport Layer Security (TLS) protocol satisfies the requirement of the typing result, namely *type-flaw resistance*.

In order to facilitate easier reasoning in Isabelle, we have also made several simplifications to the lazy intruder, in particular "out-sourcing" the analysis to the transition system. We also prove in Isabelle that these simplifications are without loss of generality, i.e., we prove the equivalence to a standard transition system with a full intruder. We use the Isabelle formalization of the lazy intruder in this paper only as a means to prove the main typing result, however it can also be employed directly to conduct lazy intruder-based proofs in Isabelle. More generally, we believe that all tools that use the lazy intruder technique can benefit from the simplification we made to the technique here.

Since this work consists of many definitions and theorems, including several variants of theorems, we here give a shortlist of definitions and the main typing result, i.e., everything that one needs to consider in order to apply our result:

- Given a protocol described as a countable set of closed strands, we define a state transition system with constraints (Definition 9).
- We define the semantics of constraints using a standard Dolev-Yao intruder deduction relation (in the free algebra) (Section IV-B and Definition 1).
- We define the notion of *type-flaw resistance* for protocols (Definition 8).
- We define a requirement on the use of operators in the protocol, called *analysis-invariance*, needed to fix a mistake in [1] (Definition 11).
- The main result is that for any reachable state of a type-flaw resistant, analysis-invariant protocol, there is a solution for the constraints if and only if there is a well-typed solution (Theorem 4).

Throughout the paper we have chosen to use slightly simplified Isabelle notation. The full Isabelle formalization (with all proofs) is available at our website:

http://www2.compute.dtu.dk/~samo/typing-soundness/

## II. Expressing the Preliminaries in Isabelle

In this section we summarize some standard definitions along with a discussion of how we model them in Isabelle/HOL whenever there are some differences. This also gives us a chance to review a few features of Isabelle that are relevant for following this paper in detail. In fact, we simplify the Isabelle notation in several places.

### A. Term Algebra

At the core of all definitions are the protocol messages that we model in a free first-order term algebra as is often done. The standard notions like unification are already part of several Isabelle libraries namely the *Unification* example theory that ships with Isabelle and the IsaFoR/CeTA library [26], and we point out only where our definitions augment them.

Our definitions are parameterized over a set $\mathcal{V}$ of *variables* (typically denoted with letters $x, y, z$) and a set $\Sigma$ of *function symbols* (typically denoted with letters $f, g, h$). We also assume a function $\texttt{arity} : \Sigma \to \mathbb{N}$ that assigns each function symbol its arity. We denote by $\mathcal{C}$ the subset of $\Sigma$ of *constants*, i.e., the function symbols of arity $0$ (typically denoted with letters $a, b, c$). By $\Sigma^n$ we then denote the subset of $\Sigma$ containing all symbols of arity $n$. We further partition $\mathcal{C}$ into the disjoint sets $\mathcal{C}_{pub}$ of public and $\mathcal{C}_{priv}$ of private constants. We later define that the intruder has access to all constants in $\mathcal{C}_{pub}$.

We now define the set of terms over $\Sigma$, $\mathcal{V}$ in Isabelle as an inductive datatype:

$$\textbf{datatype}\ (\Sigma, \mathcal{V})\ \textit{term} = \texttt{Var}\ \mathcal{V} \mid \texttt{Fun}\ \Sigma\ ((\Sigma, \mathcal{V})\ \textit{term list})$$

Here we have slightly changed the Isabelle notation that would have instead of $\Sigma$ and $\mathcal{V}$ two type variables, which we for ease of notation do not distinguish from the universes of those values.[1] For instance the expression $\texttt{Fun}\ f\ [\texttt{Var}\ x,\ \texttt{Fun}\ c\ []]$ represents the term $f(x, c)$ in more conventional notation, and we will use that notation whenever possible. Note that because this definition introduces the data constructors Var and Fun as injective functions, we obtain a free term algebra, i.e. $f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)$ iff $f = g$ and $t_i = s_i$ for $1 \leq i \leq n = m$.

Typically there is only a small finite set of non-constant function symbols representing the cryptographic primitives and the like; therefore this set is actually fixed in several other works on protocol security [21], [4], [16] with one data constructor for each function symbol. Our parameterized version has the advantage that our proofs do not depend on the particular set of operators used, so we do not have to update our proofs when adding a new operator. A slight disadvantage is that we cannot control as part of the data-type definition that a function symbol $f$ with $\texttt{arity}\ f = n$ is only applied to a list of exactly $n$ arguments. We can fix this however by the following notion of *well-formed* terms (where $\sqsubseteq$ is the subterm relation that is defined as expected):

$$\texttt{wf}_{\texttt{trm}}\ t \equiv \forall f\ T.\ \texttt{Fun}\ f\ T \sqsubseteq t \longrightarrow \texttt{length}\ T = \texttt{arity}\ f$$

We deal only with well-formed terms and for simplicity omit writing it as a side-condition on all terms we use in the rest of the paper. We further define the function FV for the *free variables* of a term as standard, and say that a term $t$ is *ground* if $\texttt{FV}\ t = \emptyset$. Additionally, the *set of subterms* of a term $t$ is written $\texttt{subterms}\ t$. Both functions are extended to sets as expected.

### B. Substitutions and Interpretations

We use for substitutions and unifications the definitions and theorems of the IsaFoR/CeTA library where substitutions (typically denoted with letters $\theta, \delta$) are functions from variables $\mathcal{V}$ to terms $(\Sigma, \mathcal{V})$ *term*. They are homomorphically extended to functions on terms as expected, and we simply write $\theta\ t$ for applying substitution $\theta$ to term $t$ (omitting the extensions function). The *composition* $\theta \cdot \delta$ of substitutions $\theta$ and $\delta$ is defined as $(\theta \cdot \delta)\ t = \delta\ (\theta\ t)$. (This is following the convention of IsaFoR/CeTA.) The *substitution domain* $\texttt{dom}_{\texttt{subst}} : (\Sigma, \mathcal{V})\ \textit{subst} \rightarrow \mathcal{V}\ \textit{set}$ of a substitution is the set of variables that are not mapped to themselves:

$$\texttt{dom}_{\texttt{subst}}\ \theta \ \equiv\ \{x \mid\ \theta\ x \neq \texttt{Var}\ x\}$$

For substitutions with finite domain we will use the common notation of value mappings, like $\theta = [x \mapsto s, y \mapsto t]$ for the substitution $\theta$ with substitution domain $\{x, y\}$ sending $x$ to $s$ and $y$ to $t$. Thus, $[]$ denotes the identity substitution.

The *substitution image* $\texttt{img}_{\texttt{subst}} : (\Sigma, \mathcal{V})\ \textit{subst} \rightarrow (\Sigma, \mathcal{V})\ \textit{term set}$ is defined in terms of the domain by applying the substitution to every element of the domain (where $f\ `\ S$ denotes the image of $f$ under the set $S$):

$$\texttt{img}_{\texttt{subst}}\ \theta \ \equiv\ \theta\ `\ \texttt{dom}_{\texttt{subst}}\ \theta$$

Every substitution we use in this paper has either a finite domain or its domain are all variables of $\mathcal{V}$ and it maps them to ground terms. The latter kind we call *interpretations*. We thus divert here from the common convention that substitution and interpretation are two disjoint notions, because they are conceptually so similar (e.g. they can be applied to all term-based data-structures) that having them separated would lead to two similar versions of many definitions and lemmas.

It is cumbersome to work with substitutions where some variable occurs both in the domain and the image like $[x \mapsto f(x)]$ as they are, for instance, not idempotent. Thus, we introduce a notion of well-formedness of substitutions that excludes any variable to occur both in domain and image and that requires a finite domain (because we will not use this notion on interpretations):

$$\begin{aligned}\texttt{wf}_{\texttt{subst}}\ \theta \ \equiv\ & \texttt{dom}_{\texttt{subst}}\ \theta \cap \texttt{FV}\ (\texttt{img}_{\texttt{subst}}\ \theta) = \emptyset\ \wedge \\ & \texttt{finite}\ (\texttt{dom}_{\texttt{subst}}\ \theta)\end{aligned}$$

Intuitively, a well-formed substitution represents a set of solutions (e.g. all solutions to a unification problem). We can prove the following lemma that is useful later when we compose substitutions in constraint reduction:

*Lemma 1 (Well-formedness preservation of substitution composition):* If $\theta_1$ and $\theta_2$ are well-formed substitutions such that $\texttt{dom}_{\texttt{subst}}\ \theta_1 \cap \texttt{dom}_{\texttt{subst}}\ \theta_2 = \emptyset$ and $\texttt{dom}_{\texttt{subst}}\ \theta_1 \cap \texttt{FV}\ (\texttt{img}_{\texttt{subst}}\ \theta_2) = \emptyset$ then the composition $\theta_1 \cdot \theta_2$ is also well-formed.

An interpretation (typically denoted by letter $\mathcal{I}$) is a substitution that represents one single solution, mapping every variable to a ground term:

$$\begin{aligned}\texttt{interpretation}_{\texttt{subst}}\ \mathcal{I} \ \equiv\ & \texttt{dom}_{\texttt{subst}}\ \mathcal{I} = \mathcal{V}\ \wedge \\ & \texttt{ground}\ (\texttt{img}_{\texttt{subst}}\ \mathcal{I})\end{aligned}$$

We define that a substitution $\theta$ *supports* an interpretation $\mathcal{I}$ iff $\mathcal{I}$ is a solution represented by $\theta$:

$$\theta\ \texttt{supports}\ \mathcal{I} \ \equiv\ \forall x.\ \mathcal{I}\ (\theta\ x) = \mathcal{I}\ x$$

### C. Unification

A *most general unifier* (*mgu*) $\theta$ between two terms, $t_1$ and $t_2$, is defined as a substitution satisfying the following standard definition:

$$\texttt{MGU}\ \theta\ t_1\ t_2 \equiv \theta\ t_1 = \theta\ t_2 \wedge (\forall \delta.\ \delta\ t_1 = \delta\ t_2 \longrightarrow (\exists \gamma.\ \delta = \theta \cdot \gamma))$$

In other words, $\theta$ is a unifier which can be used to construct any other unifier $\delta$ of $t_1$ and $t_2$ using composition with a third substitution $\gamma$. *Well-formed mgu*s are furthermore restricted to the variables of the terms being unified. That is:

$$\begin{aligned}\texttt{wf}_{\texttt{MGU}}\ \theta\ s\ t \equiv\ & \texttt{wf}_{\texttt{subst}}\ \theta \wedge \texttt{MGU}\ \theta\ s\ t\ \wedge \\ & \texttt{dom}_{\texttt{subst}}\ \theta \cup \texttt{FV}\ (\texttt{img}_{\texttt{subst}}\ \theta) \subseteq \texttt{FV}\ s \cup \texttt{FV}\ t\end{aligned}$$

The library of IsaFoR/CeTA provides the function

$$\texttt{mgu} :: (\Sigma, \mathcal{V})\ \textit{term} \Rightarrow (\Sigma, \mathcal{V})\ \textit{term} \Rightarrow (\Sigma, \mathcal{V})\ \textit{subst option}$$

[1] In Isabelle, one has to rather write *UNIV* to refer to the set of values that belong to a particular type.

that computes the most general unifier of two terms if one exists. We proved that this unifier is always well-formed.

## III. Dolev-Yao Style Intruder Model

We define a standard symbolic Dolev-Yao style intruder deduction relation $M \vdash t$ to formalize that the intruder can derive term $t$ from the set of terms $M$, his *knowledge*. We define $\vdash$ inductively as the least relation closed under the following rules:

*Definition 1 (The Intruder Model):*

$$\frac{}{M \vdash t}\ \begin{array}{c}(Axiom),\\ t \in M\end{array} \qquad \frac{M \vdash t_1 \quad \cdots \quad M \vdash t_n}{M \vdash f(t_1, \ldots, t_n)}\ \begin{array}{c}(Compose),\\ f \in \Sigma^n,\\ \texttt{public}\ f\end{array}$$

$$\frac{M \vdash t \quad M \vdash k_1 \quad \cdots \quad M \vdash k_n}{M \vdash t_i}\ \begin{array}{c}(Decompose),\\ \texttt{Ana}\ t = (K, T),\ t_i \in T,\\ K = \{k_1, \ldots, k_n\}\end{array}$$

Here, all terms are at first ground terms without variables (as we are not performing any substitutions); later we will use constraints with variables in terms. For later we also define a restricted variant $\vdash_c$ that is the closure only under $(Axiom)$ and $(Compose)$, but omitting $(Decompose)$.

The first rule expresses that the intruder can derive everything in his knowledge. The second rule allows the intruder to *compose* messages by applying *public* function symbols to messages he can already derive. (Note that $f$ might have arity $n = 0$, in which case it is a constant.) To that end, we define the function `public` that yields true for all public constants and any function with arity greater than zero (i.e., all symbols of $\Sigma^n \setminus \mathcal{C}_{priv}$). The third rule allows the intruder to *decompose* (i.e. *analyze*) messages. To avoid that we have to write a special decryption rule for each operator to consider, we define a function `Ana` as an interface. Intuitively, `Ana` $t = (K, T)$ means that the intruder can analyze the term $t$ provided that he knows the "keys" in $K$ and then obtain as the result of analysis the terms of $T$. The advantage of this interface function is that in order to add a new operator to the model, one simply has to specify the `Ana` function for it, but none of the following definitions or theorems require an update. We will require some restrictions on the `Ana` function in Section VI-B.

*Example 1:* Consider the following set of non-constant operators (with their arities): asymmetric encryption $\mathsf{crypt}/2$, symmetric encryption $\mathsf{scrypt}/2$, signatures $\mathsf{sign}/2$, a function $\mathsf{pub}/1$ that yields the public key for a given private key, hash function $\mathsf{hash}/1$, a key derivation function $\mathsf{kdf}/2$ and message structuring formats $\mathsf{f}_i/i$ ($i \in \mathbb{N}$), together with the following `Ana` function:

$$\begin{array}{lcl}
\texttt{Ana}\ \mathsf{scrypt}(k, m) & = & (\{k\}, \{m\}) \\
\texttt{Ana}\ \mathsf{crypt}(\mathsf{pub}(k), m) & = & (\{k\}, \{m\}) \\
\texttt{Ana}\ \mathsf{sign}(k, m) & = & (\emptyset, \{m\}) \\
\texttt{Ana}\ \mathsf{f}_i(t_1, \ldots, t_i) & = & (\emptyset, \{t_1, \ldots, t_i\}) \\
\multicolumn{3}{l}{\quad \text{and in all other cases: } \texttt{Ana}\ t = (\emptyset, \emptyset)}
\end{array}$$

This describes the decryption of symmetric and asymmetric encryptions as expected; the contents of signatures we assume can be obtained without knowing the signing key (i.e., the signature primitive includes the signed text in clear). The functions $\mathsf{pub}$, $\mathsf{hash}$, and $\mathsf{kdf}$ are one-way in the sense that they do not yield any information in analysis. The non-cryptographic message structuring formats $\mathsf{f}_i$ exist only to structure clear-text messages (other than pure concatenation). Like [1] we use these formats instead of the classical "concatenate" operator: this allows for modeling abstractly the actual mechanisms of the implementation to structure messages unambiguously, such as tags, length information, or character encodings; compare for instance the TLS example in Section V-B. The formats are *transparent*, i.e., the analysis function yields all direct subterms without requiring any key. For signatures, we similarly allow the intruder to obtain the signed message without any key. This models a signature scheme where the message being signed is given along with a signature on a hash of that message; thus one does not need any key to *obtain* the message, but only to *verify* the signature. More on signature verification below.

Let $M = \{\mathsf{scrypt}(\mathsf{kdf}(\mathsf{n1}, \mathsf{n2}), \mathsf{secret}), \mathsf{n1}, \mathsf{n2}\}$ then for instance $M \vdash \mathsf{secret}$ since the intruder can first compose the key $\mathsf{kdf}(\mathsf{n1}, \mathsf{n2})$ and then decrypt the encrypted message. □

### A. Free Algebra

Recall that our definition of terms yields a free term algebra, i.e., two terms are equal only if they are syntactically equal. This prevents many interesting properties of operators like the property $g^{xy} = g^{yx}$ that is needed for all Diffie-Hellman-based protocols. Modeling algebraic properties in Isabelle is not trivial: one has to work with a quotient algebra (every term represents the *set* of terms that are algebraically equal) and thus everything becomes more complex, see for instance [24], [14]. Therefore all protocol verification in Isabelle that we know of uses the free algebra. Most typing results also are limited to the free algebra (an exception being [18]).

One may wonder, however, how this free algebra model compares to other protocol models like the Applied-$\pi$ calculus model of ProVerif that supports algebraic properties to some extent. As an example, to describe signature verification, one may specify a *destructor* function verify that takes as arguments a signed message and a public key and yields true if the signature is correct w.r.t. that public key. This is expressed by the rewrite rule:

$$\mathsf{verify}(\mathsf{sign}(privkey, msg), \mathsf{pub}(privkey)) \to true$$

(Similar rewrite rules we have for decryption functions and the like.) This is in fact an algebraic property and it cannot be directly expressed in a free algebra term model. Note that internally, ProVerif works with Horn clauses in the free algebra as well. Therefore a transformation step is taken when translating a given Applied-$\pi$-calculus specification into Horn clauses. In the example of signature verification or similar theories of constructors and destructors, this would amount to pattern matching. Consider for instance an honest agent who receives an arbitrary message $x$ and checks that applying verify with a particular key $\mathsf{pub}(privkey)$ yields $true$; ProVerif's transformation would yield an agent who now receives only messages of *pattern* $\mathsf{sign}(privkey, y)$ where $y$ is a variable to which the content of the signature is bound. This is precisely how free algebra approaches handle constructors—having no explicit destructors anymore.

Furthermore, ProVerif also allows for equations and it similarly applies a completion procedure to the Horn clauses to take into account all algebraic variants of a term. Note this feature may easily lead to non-termination and one must carefully craft the algebraic properties for this, see for instance [15]. In principle one can apply the same transformation also to strands in order to handle some algebraic properties, but we leave this external to our approach.

## IV. Modeling the Lazy Intruder in Isabelle/HOL

A naïve approach to model checking security protocols would be to devise a transition system that contains transitions for honest agents and composition/decomposition steps for the intruder. Since composition is infinite, one would not only bound the number of honest agents and the number of protocol runs they can participate in, but also the complexity of messages that the intruder can compose. (In fact, the typing result shows that for a large class of protocols this bounding of the intruder would be without loss of attacks.) But even under tight bounds, the search space is infeasibly large. Therefore a technique has emerged that replaces this "eager" exploration of what the intruder can do by a symbolic approach with constraints and a demand-driven, "lazy" evaluation of these constraints [17], [23], [3]. We thus like to call this technique the *lazy intruder*. While a successful method in the analysis and verification (for a bounded number of sessions) of security protocols, it has also been used as a proof argument for typing and compositionality results [1], [9], [2] like the one we formalize here in Isabelle. Note that in this way of using the lazy intruder, there is no bound on the number of sessions.

The lazy intruder constraints are of the form $M \vdash t$ where now $M$ and $t$ can contain variables. Intuitively, $M$ is the knowledge that the intruder had at a point where he sent a message of the form $t$ to some honest agent. Here the term $t$ may contain variables, so that $t$ is a *pattern* of what messages the agent would accept and the variables are the places where the agent does not expect a particular value. This is where the lazy intruder is lazy: we do not right away try to determine a value for each variable. Therefore, the next messages this honest agent sends may contain variables from $t$ and this is how variables can end up in the intruder knowledge $M'$ in a successor state.

For a feasible procedure for checking the satisfiability of constraints, one needs to require a well-formedness condition on constraints: they can be ordered as $M_1 \vdash t_1, \ldots, M_n \vdash t_n$ (the order in which the constraints occurred) where

1) $M_i \subseteq M_{i+1}$ (for $1 \leq i < n$): the intruder knowledge grows monotonically; and
2) $\mathtt{FV}\ M_i \subseteq \bigcup_{k=1}^{i-1} \mathtt{FV}\ t_k$: all variables originate from a term sent by the intruder.

A large part of this work is to formalize in Isabelle/HOL these lazy intruder constraints, a reduction procedure for the constraints, and to prove the soundness, completeness, and termination of this procedure. Completeness and termination are quite difficult even as standard pen-and-paper proofs. We therefore had to first seek for any possibilities to make the task and the formalization as easy and light-weight as possible. The main simplifications are a different representation and an "outsourcing" of decomposition steps, as we explain next.

The formalization we present here is a so-called *deep embedding*, i.e. we formalize constraints as objects in Isabelle that we can reason about. This is in contrast to a *shallow embedding* where we simply consider them as HOL formulae that use the $\vdash$ predicate. A shallow embedding would have advantages (both in terms of simplicity and performance) if one would like to directly perform constraint reasoning in Isabelle. A deep embedding is however necessary for our purpose, since we want to reason about *a procedure* that manipulates constraints, and in particular prove that this procedure is complete and terminates (the soundness proof could also be expressed in a shallow embedding).

### A. The Lazy Intruder on the Beach

The first idea for keeping matters simple is to change the representation of the constraints by using *strands*.[2] A strand is a sequence of send and receive operations and strand spaces are a nice formalism to reason about protocol executions [25]. We define an *intruder strand* as a list of received and sent messages:

$$\begin{array}{l}\textbf{datatype}\ (\Sigma,\mathcal{V})\ \textit{strand-step} = \\ \quad \mathtt{Send}\ ((\Sigma,\mathcal{V})\ \textit{term}) \mid \mathtt{Receive}\ ((\Sigma,\mathcal{V})\ \textit{term}) \\ \textbf{type-synonym}\ (\Sigma,\mathcal{V})\ \textit{strand} = (\Sigma,\mathcal{V})\ \textit{strand-step list}\end{array}$$

Thus, the intruder knowledge at each point in the strand are the messages that the intruder has received up to this point, and each sent message must be something he can construct from the knowledge at that point.

*Example 2:* Consider the following constraints in traditional representation:

$$\begin{array}{ll}\{\mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}), \mathsf{secret}), \mathsf{k_i}\} & \vdash \mathsf{crypt}(\mathsf{pub}(x), y) \\ \{\mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}), \mathsf{secret}), \mathsf{k_i}, y\} & \vdash \mathsf{secret}\end{array}$$

In the strand representation we would write this as:

$$\begin{array}{l}\mathtt{Receive}\ \mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}), \mathsf{secret}).\mathtt{Receive}\ \mathsf{k_i}. \\ \mathtt{Send}\ \mathsf{crypt}(\mathsf{pub}(x), y).\mathtt{Receive}\ y.\mathtt{Send}\ \mathsf{secret}.0\end{array}$$

Instead of $[st_1, \ldots, st_n]$ we rather write $st_1. \ \ldots \ .st_n.0$ like in process calculi. □

The advantage of our representation is that we have "built-in" the first condition of the well-formedness: that the intruder knowledge monotonically grows. The second condition—that all variables originate in terms sent by the intruder—is now easy to formulate:

*Definition 2:* An intruder strand $\mathcal{A}$ is *well-formed* iff $\mathtt{wf_{st}}\ \emptyset\ \mathcal{A}$ holds where:

$$\begin{array}{rcl}\mathtt{wf_{st}}\ V\ 0 & \text{iff} & \textit{True} \\ \mathtt{wf_{st}}\ V\ (\mathtt{Receive}\ t.\mathcal{A}) & \text{iff} & \mathtt{FV}\ t \subseteq V \text{ and } \mathtt{wf_{st}}\ V\ \mathcal{A} \\ \mathtt{wf_{st}}\ V\ (\mathtt{Send}\ t.\mathcal{A}) & \text{iff} & \mathtt{wf_{st}}\ (V \cup \mathtt{FV}\ t)\ \mathcal{A}\end{array}$$

[2] Note that the word *strand* in Danish and German means *beach*.

Here the parameter $V$ of $\mathtt{wf}_{\mathtt{st}}\ V\ \mathcal{A}$ is meant to denote the free variables of all sent messages that have occurred in a prefix of the parameter $\mathcal{A}$. The variables occurring in an intruder strand $\mathcal{A}$ are denoted by $\mathtt{vars}_{\mathtt{st}}\ \mathcal{A}$. Moreover, the *intruder knowledge* of an intruder strand $\mathcal{A}$, written $\mathtt{ik}_{\mathtt{st}}\ \mathcal{A}$, is the set of received messages. That is, $t \in \mathtt{ik}_{\mathtt{st}}\ \mathcal{A}$ iff $\mathtt{Receive}\ t$ occurs in $\mathcal{A}$.

### B. Constraint Semantics

We define the semantics of intruder strands based on the Dolev-Yao deduction relation $\vdash$. Recall that an interpretation $\mathcal{I}$ maps all variables to ground terms. We write $[\![\mathcal{A}]\!]\ M\ \mathcal{I}$ to denote that $\mathcal{I}$ is a solution of an intruder strand $\mathcal{A}$ where $M$ is an (initially empty) set of messages available to the intruder at the start, and define this relation as follows:

$$\begin{array}{rll} [\![0]\!]\ M\ \mathcal{I} & \text{iff} & \textit{True} \\ [\![\mathtt{Send}\ t.\mathcal{A}]\!]\ M\ \mathcal{I} & \text{iff} & (\mathcal{I}\ M) \vdash (\mathcal{I}\ t) \text{ and } [\![\mathcal{A}]\!]\ M\ \mathcal{I} \\ [\![\mathtt{Receive}\ t.\mathcal{A}]\!]\ M\ \mathcal{I} & \text{iff} & [\![\mathcal{A}]\!]\ (\{t\} \cup M)\ \mathcal{I} \end{array}$$

Thus, every message he receives is simply added to the parameter $M$ that collects the intruder knowledge in this inductive definition. For every message $t$ that the intruder sends, we require that he can derive it from knowledge $M$ (under the interpretation $\mathcal{I}$).

*Example 3:* Consider the intruder strand from the previous example. Any $\mathcal{I}$ with $\mathcal{I}\ x = \mathsf{k_a}$ and $\mathcal{I}\ y = \mathsf{secret}$ is a solution of the constraint. Consider only the prefix up to (and including) the first $\mathtt{Send}$ step; then also $\mathcal{I}\ x = \mathcal{I}\ y = \mathsf{k_i}$ is a solution, and so is any interpretation that maps $x$ and $y$ to terms that the intruder can generate from his knowledge at that point. □

During constraint reduction below we will consider pairs $(\mathcal{A}, \theta)$ of an intruder strand $\mathcal{A}$ and a well-formed substitution $\theta$ that represents the (partial) solution obtained so far (like the solution for $x$ and $y$ in the example above). At the beginning of the reduction, $\theta$ is simply the identity. Whenever $\theta$ is augmented during reduction, we apply it also to $\mathcal{A}$, i.e., $\mathcal{A}$ contains no variables in the domain of $\theta$ (that are already "solved").[3] Formally:

*Definition 3:* An *intruder constraint* is of the form $(\mathcal{A}, \theta)$, where $\mathcal{A}$ is an intruder strand and $\theta$ is a substitution. An intruder constraint $(\mathcal{A}, \theta)$ is furthermore *well-formed* if 1) $\mathcal{A}$ is a well-formed intruder strand, 2) $\theta$ is a well-formed substitution, and 3) the domain of $\theta$ and the variables of $\mathcal{A}$ are disjoint.

The interpretation $\mathcal{I}$ is said to be a *model* of $(\mathcal{A}, \theta)$ (with initial intruder knowledge $M_0$) written $M_0, \mathcal{I} \models (\mathcal{A}, \theta)$, iff $\theta$ supports $\mathcal{I}$ and $[\![\mathcal{A}]\!]\ M_0\ \mathcal{I}$ holds. For the default $M_0 = \emptyset$ we simply write $\mathcal{I} \models (\mathcal{A}, \theta)$, and $\mathcal{I} \models \mathcal{A}$ if additionally $\theta = []$.

### C. Out-sourcing Analysis

One aspect that makes the lazy intruder complicated, both in terms of an implementation in tools, and in terms of proving completeness and termination of the reduction procedure below, is analysis of terms. For instance, if the intruder learns an encrypted term where the subterm for the key contains a variable, then whether he can decrypt the term may depend on the substitution for that variable. In general, one has to make then a case split: the case that the message can be deciphered and the case that it cannot, since ignoring either case may eliminate solutions. In fact, in the case a message has not been decrypted, after each received message another case split is necessary whether or not the term should now be decrypted. Another complication arises from the fact that a term in the intruder knowledge could directly be a variable that may represent a decryptable term, and one has to carefully argue that the term in question was known to the intruder earlier (and could have been decrypted then), but this argument requires that the earlier constraints have already been simplified.

To avoid these complications, we now consider the following idea: we limit the intruder to composition steps, i.e., the $\vdash_c$ relation, and "out-source" all decomposition steps to the transition system. To that end, we can imagine special honest agents that perform decryption operations for the intruder. We discuss this in detail (and prove correctness) in Section VI.

One may wonder why we even bother with the lazy intruder and do not simply out-source also the composition steps of $\vdash_c$ as well. Recall that the lazy intruder was conceived to counter the problem that the $\vdash_c$ closure is infinite (while closure under analysis is finite). In other words, in a forward exploration of a transition system, composition leads to blind exploration (while analysis is not a problem). A backwards search like lazy intruder constraint solving is a clear demand-driven way to handle composition steps. Exactly this demand-driven, lazy aspect is what we shall exploit in the typing results: while the intruder *can* compose ill-typed messages, this is *never necessary* to mount the attack, and this "never necessary" is captured by the laziness of the intruder. Not having analysis steps as part of that argument does not hurt, because the analysis of terms is not what introduces ill-typed messages. In fact, we believe that even for automated tools that use the lazy intruder technique, the out-sourcing of analysis could be beneficial, since it drastically simplifies the technique, and as far as we can see every provision for efficiency (e.g. eagerly performing analysis steps that require no substitution) can be similarly applied at the transition system level.

We then define a variant $[\![\cdot]\!]_c$ of the semantics $[\![\cdot]\!]$ restricted to composition steps by replacing $\vdash$ with $\vdash_c$ in the definition of $[\![\cdot]\!]$. Similarly, we define $\models_c$ by replacing $[\![\cdot]\!]$ with $[\![\cdot]\!]_c$ in the definition of $\models$.

### D. Constraint Reduction

The goal of the constraint reduction procedure is to determine all solutions of a constraint. There are in general infinitely many solutions, but they can be finitely represented, namely by *simple* constraints:

*Definition 4:* An intruder constraint $(\mathcal{A}, \theta)$ is *simple* if and only if $t \in \mathcal{V}$ for every $\mathtt{Send}\ t$ that occurs in $\mathcal{A}$.

The point is that simple constraints are always satisfiable: the remaining variables are arbitrary, so the intruder can choose any term from his knowledge. A key point of the typing result

[3] In Isabelle, we have to define explicitly an extension of substitution to functions on constraints (defined homomorphically as expected) but we leave this implicit in the notation in this paper, for simplicity.

is: when the variables are annotated with an intended type (and the intruder knows values for each type), then there always also exists a well-typed solution for a simple constraint.

The goal of the reduction procedure is now to transform a given constraint into an equivalent set of simple constraints. That is, we define a reduction relation $\rightsquigarrow$ on constraints[4], and for a given $(\mathcal{A}_0, \theta_0)$, we consider the reachable constraints, i.e., $(\mathcal{A}_0, \theta_0) \rightsquigarrow^* (\mathcal{A}, \theta)$. (The relation $\rightsquigarrow^*$ denotes the reflexive transitive closure of $\rightsquigarrow$ while $\rightsquigarrow^+$ then denotes the transitive closure.) We prove in Isabelle that there are finitely many (termination), and that the union of the models of the reachable simple constraints is exactly the set of models of $(\mathcal{A}_0, \theta_0)$ (soundness and completeness).

*Definition 5:* The lazy intruder is the least relation $\rightsquigarrow$ between constraints closed under the following rules:

$$\begin{aligned}(\mathit{Compose}_{LI}):\quad & (\mathcal{A}.\mathtt{Send}\ f(t_1,\ldots,t_n).\mathcal{A}', \theta)\\ & \rightsquigarrow (\mathcal{A}.\mathtt{Send}\ t_1.\cdots.\mathtt{Send}\ t_n.\mathcal{A}', \theta)\\ & \text{if } \mathtt{simple}\ \mathcal{A}, f \in \Sigma^n, \mathtt{public}\ f\end{aligned}$$

$$\begin{aligned}(\mathit{Unify}_{LI}):\quad & (\mathcal{A}.\mathtt{Send}\ s.\mathcal{A}', \theta) \rightsquigarrow (\delta\ (\mathcal{A}.\mathcal{A}'), \theta \cdot \delta)\\ \text{if}\quad & s, t \notin \mathcal{V}, \mathtt{simple}\ \mathcal{A}, t \in \mathtt{ik}_{\mathtt{st}}\ \mathcal{A},\ \mathtt{Some}\ \delta = \mathtt{mgu}\ s\ t\end{aligned}$$

The $(\mathit{Compose}_{LI})$ rule corresponds to the $(\mathit{Compose})$ rule of the Dolev-Yao model, i.e., if the intruder has to produce $f(t_1,\ldots,t_n)$ for a public symbol $f$, one way to do it is to produce each of the $t_i$ (in whatever way) and apply $f$ to them.

The $(\mathit{Unify}_{LI})$ rule corresponds to the $(\mathit{Axiom})$ rule of the Dolev-Yao model: it states that the intruder can send a message $s$ if he has previously learned a message $t$ that can be unified with $s$. More precisely, the most general unifier $\delta$ of $s$ and $t$ (if it exists; the data constructor Some is from the *option* datatype) represents all interpretations of the constraint, under which $s$ and $t$ are equal, and thus under which the Send $s$ can be removed as this requirement is satisfied. However, we have to integrate $\delta$ by composing it with the existing solution $\theta$ and applying it to the rest of the constraint, so that no variable in the domain of $\delta$ remains in the intruder strand.

Note that the $(\mathit{Unify}_{LI})$ rule is not applicable if the term $s$ to be generated is a variable, because we are lazy: since so far there is no more constraint on what $s$ should be precisely, it is pointless to explore options—the intruder can always generate *something*. This is a key to the typing result later. In following reduction steps, this variable may be replaced with a more concrete term, and then we explore how that term can be generated. Finally, the rule also forbids that the term $t$ that we unify with $s$ is a variable (and this is again crucial in the typing result), but that this restriction is without loss of generality is a tricky part of the completeness proof.

Note that in contrast to many other lazy intruder methods, these rules are only applicable to the first term of the form Send $t$ where $t$ is not a variable, i.e., all prior Send steps must be simple already. This restriction is without loss of generality again as our proofs show; since this removes some non-determinism, it also makes some arguments later easier, because we can rely on the prefix to be simple.

*Example 4:* Let us reduce the constraint from Example 2 (with [] as initial substitution). With one $\rightsquigarrow$ step (addressing the first Send) we can get (using $(\mathit{Compose}_{LI})$) to: $\ldots\mathtt{Send}\ \mathsf{pub}(x).\mathtt{Send}\ y.\mathtt{Receive}\ y.\mathtt{Send}\ \mathsf{secret}.0$.

Since we cannot unify the $\mathsf{pub}(x)$ with anything, we then are forced to make another compose, leading to: $\ldots\mathtt{Send}\ x.\mathtt{Send}\ y.\mathtt{Receive}\ y.\mathtt{Send}\ \mathsf{secret}.0$ The remaining non-simple Send secret cannot be solved (if $\mathsf{secret} \in \mathcal{C}_{priv}$), i.e. it has no successor under $\rightsquigarrow$. Since it is not simple, it does not have any solution by completeness of the lazy intruder (the actual completeness theorem will be introduced later).

From the original constraint we can however also reach another constraint when using $(\mathit{Unify}_{LI})$ between the received encrypted message and the one to be sent, giving the unifier $\delta = [x \mapsto \mathsf{k_a}, y \mapsto \mathsf{secret}]$: $\ldots\mathtt{Receive}\ \mathsf{secret}.\mathtt{Send}\ \mathsf{secret}.0$ which can trivially be solved with another unify step. □

Our formalization can be extended with equality constraints $s_i \doteq t_i$ as well. To solve such constraints we can simply compute the mgus $\theta_i$ of all equality constraints $s_1 \doteq t_1, \cdots, s_n \doteq t_n$ associated with an intruder strand $\mathcal{A}$ and then solve the constraint $(\theta\ \mathcal{A}, \theta)$ where $\theta = \theta_1 \cdot \ldots \cdot \theta_n$. The resulting constraint is well-formed as well since $\theta$ is a well-formed substitution by Lemma 1 and hence the set of variables of $\theta\ \mathcal{A}$ and the domain of $\theta$ would be disjoint. Negative equality constraints $\neg\exists\bar{x}.\ s_i \doteq t_i$ can be handled separately from the lazy intruder, similar to how they are handled in other works like [1].

### E. Proving Soundness & Completeness

A great part of the contribution of this paper lies in the Isabelle proof of the soundness and completeness of lazy intruder reduction. This is following basically the proofs in [1] and we do not repeat here any proof sketches. In fact, with analysis (that we out-source to the transition system) we found several mistakes in [1]; these are discussed on the transition system level in Section VI-B, along with a correction.

We first prove that all reductions preserve well-formedness of the constraints:

*Lemma 2 (Well-formedness preservation):* If $(\mathcal{A}_1, \theta_1)$ is well-formed and $(\mathcal{A}_1, \theta_1) \rightsquigarrow^* (\mathcal{A}_2, \theta_2)$ then $(\mathcal{A}_2, \theta_2)$ is well-formed.

From the well-formedness we can quite easily derive soundness, i.e., that no reduction step introduces new solutions:

*Theorem 1 (Soundness):* If $(\mathcal{A}, \theta)$ is well-formed, $(\mathcal{A}, \theta) \rightsquigarrow^* (\mathcal{A}', \theta')$, and $\mathcal{I} \models_c (\mathcal{A}', \theta')$, then $\mathcal{I} \models_c (\mathcal{A}, \theta)$.

The proof of completeness relies on the termination which we thus prove first.

*Lemma 3 (Termination):* For a constraint $(\mathcal{A}, \theta)$, the set of reachable constraints $\{(\mathcal{A}', \theta') \mid (\mathcal{A}, \theta) \rightsquigarrow^* (\mathcal{A}', \theta')\}$ is finite.

The next step is that all simple constraints are satisfiable:

*Lemma 4 (Simple constraints are satisfiable):* If $(\mathcal{A}, \theta)$ is well-formed and $\mathcal{A}$ is simple, then there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models_c (\mathcal{A}, \theta)$.

[4] This relation $\phi \rightsquigarrow \psi$ is sometimes written $\frac{\psi}{\phi}$, i.e, in the form of a proof calculus for satisfiability. One can read each such rule top down: every solution of $\psi$ is also a solution of $\phi$. The procedure, however, works by backwards exploring the rules: the solutions of $\phi$ include all solutions of $\psi$.

The most difficult lemma to prove is that given a non-simple but satisfiable constraint, then for every solution $\mathcal{I}$ of that constraint exists a reduction $\rightsquigarrow$ that preserves $\mathcal{I}$.

*Lemma 5 (Completeness, single step):* If $(\mathcal{A}, \theta)$ is well-formed, $\mathcal{I} \models_c (\mathcal{A}, \theta)$, and $\mathcal{A}$ is not simple, then there exists $(\mathcal{A}', \theta')$ such that $(\mathcal{A}, \theta) \rightsquigarrow (\mathcal{A}', \theta')$ and $\mathcal{I} \models_c (\mathcal{A}', \theta')$.

From this, we obtain the completeness: a well-formed constraint is either simple (and thus satisfiable), or we can make further reductions (and no solution gets lost), or else we are stuck at an irreducible constraint (that is thus unsatisfiable). Together with termination we thus have:

*Theorem 2 (Completeness):* If $(\mathcal{A}, \theta)$ is well-formed and $\mathcal{I} \models_c (\mathcal{A}, \theta)$ then there exists a $(\mathcal{A}', \theta')$ such that $\mathcal{A}'$ is simple, $(\mathcal{A}, \theta) \rightsquigarrow^* (\mathcal{A}', \theta')$, and $\mathcal{I} \models_c (\mathcal{A}', \theta')$.

## V. Typed Model

So far our model of the intruder is untyped. We now define a simple type system and consider the restriction of the model where the intruder is limited to well-typed messages. The main result of this section is the formalization of a typing result on intruder constraints for a large class of protocols: if a constraint $(\mathcal{A}, \theta)$ has a solution $\mathcal{I}$, then it also has a well-typed solution $\mathcal{I}'$. Thus, if we can verify a protocol in a typed model (all constraints that arise only have well-typed solutions) then we can infer that it is also secure in the untyped model. In this section we first develop the typing result on the level of constraints (without analysis) and then extend it to entire protocols and transition systems in Section VI.

### *A. The Type System*

Recall that our notion of terms is parameterized over a set $\Sigma$ of function symbols, so one can easily introduce new operators without updating all the proofs. Similarly, for the type system we introduce another set over which our result is parameterized: a finite set $\mathfrak{T}_a$ of *atomic* types. An example is $\mathfrak{T}_a = \{\mathsf{Agent}, \mathsf{Nonce}, \mathsf{SymmetricKey}, \mathsf{PrivateKey}\}$. (Note that public keys do not have an atomic type, because we build them using operator pub from private keys.) Next, we introduce composed types as built like terms from $\mathfrak{T}_a$ and the operators of $\Sigma$, for instance $\mathsf{crypt}(\mathsf{pub}(\mathsf{PrivateKey}), \mathsf{Nonce})$ could be a composed type. As types are thus very similar to normal terms, we re-use the definition for terms:

$$\textbf{type-synonym } (\Sigma, \mathfrak{T}_a) \textit{ term-type} \equiv (\Sigma, \mathfrak{T}_a) \textit{ term}$$

Thus, we put atomic types in every place where normal terms would have variables. (Note that our type system has no type variables, atomic types are like constants.) To avoid confusion and make definitions nicer to read, we introduce two synonyms for the constructors `Var` and `Fun` of terms, namely `TAtom` and `TComp`, and consistently use them when talking about types. We inherit all previous notions from terms, e.g., well-formedness for types (all operators are used with correct arity). However, we additionally require that no constants occur in types. Further, our result is parameterized over a *typing function* $\Gamma : (\Sigma, \mathcal{V})$ *term* $\Rightarrow (\Sigma, \mathfrak{T}_a)$ *term-type* that maps each term to a type and that must satisfy the following properties:

1) $\Gamma(c) \in \mathfrak{T}_a$ for every $c \in \mathcal{C}$.
2) $\Gamma(f(t_1, \ldots, t_n)) = f(\Gamma(t_1), \ldots, \Gamma(t_n))$ for every $f \in \Sigma \setminus \mathcal{C}$.
3) $\Gamma(v)$ must be a well-formed type for every $v \in \mathcal{V}$.

In fact, it is thus sufficient to specify $\Gamma$ for all constants (as atomic types) and all variables (as arbitrary well-formed types), and then homomorphically extend $\Gamma$ to arbitrary terms. Thus, every well-formed term $t$ has a well-formed type $\Gamma(t)$.

Finally, we want to give the intruder an unbounded supply of terms of every type, thus we require that $\mathcal{C}_{pub}$ contains infinitely many constants of every atomic type: $\bigwedge a.\ \texttt{infinite}\ \{c.\ \Gamma\ (\texttt{Fun}\ c\ [])= \texttt{TAtom}\ a \wedge \texttt{public}\ c\}$

An important point why this type system is of foundational interest is that it limits the size of terms that can be substituted for a variable, e.g., when the protocol requires a value to be of type nonce, it cannot be a composed term in the typed model anymore. Abstract interpretation approaches like the one used in ProVerif (where $\Sigma$ is finite) become decidable under this restriction, and several Isabelle proof methodologies are based on a typed model [21], [22], [5], [4], [8]. This restriction on substitutions—that they preserve typing—is captured by the following definition:

*Definition 6 (Well-typed substitutions):* A substitution $\delta$ is *well-typed* iff $\Gamma(\delta\ x) = \Gamma(x)$ for all $x \in \mathcal{V}$.

The requirement that we need for our typing result is that the messages and sub-messages of a protocol must have a different shape whenever they have different types. For that reason we specify the set of *sub-message patterns* given the set of messages $M$. In the next section we will use as $M$ the set of all messages of the protocol description (containing variables, hence *message patterns*).

*Definition 7 (Sub-message patterns):* The *sub-message patterns* $\mathit{SMP}(M)$ for a set of messages $M$ is defined as the least set satisfying the following rules:

1) $M \subseteq \mathit{SMP}(M)$.
2) If $t \in \mathit{SMP}(M)$ and $t' \sqsubset t$ then $t' \in \mathit{SMP}(M)$.
3) If $t \in \mathit{SMP}(M)$ and $\delta$ is a well-typed substitution then $\delta\ t \in \mathit{SMP}(M)$.

The intuition behind this definition is that during constraint reduction we can get to subterms of the initially given terms and apply substitutions. We will show that for the considered class of protocols these substitutions will always be well-typed, so we never fall out of $\mathit{SMP}(M)$.

We can now define the main requirement for our typing result, as a property of the set $\mathit{SMP}(M)$:

*Definition 8 (Type-flaw resistance):* We say a set $M$ of messages is *type-flaw resistant* iff $\forall s, t \in \mathit{SMP}(M) \setminus \mathcal{V}.\ (\exists \delta.\ \delta\ s = \delta\ t) \longrightarrow \Gamma(s) = \Gamma(t)$. We may also apply the notion of type-flaw resistance directly to an intruder strand $\mathcal{A}$ to mean that the set of all $t$ for which `Send` $t$ or `Receive` $t$ occurs in $\mathcal{A}$ is type-flaw resistant.

The notion of type-flaw resistance requires that we cannot unify any subterms (except variables) that have different types, i.e., terms that have different meaning must be clearly distinguishable. This is a bit more general than results that are based on adding *tags* to messages to make them distinguishable, like [13], [7] since we do not impose a particular mechanism to disambiguate messages, such as tags, but rather have a very general definition: to prove type-flaw resistance you just have to ensure that terms of different types are not unifiable (hence distinguishable). We illustrate this with a real-world example, also formalized in Isabelle, by proving type-flaw resistance of TLS.

### B. TLS Example

As a real-world example, let us consider the messages of the TLS Handshake protocol [10]. TLS defines several concrete message structuring formats, e.g., the first message of the TLS handshake is called clientHello, and contains essentially five distinct pieces of information (such as a time stamp and a random number); the concrete message format includes also length information and a tag (to distinguish the clientHello from other messages). We represent in our term algebra by an abstract function of five arguments $\mathtt{clientHello}(T, R, S, C, K)$ and define it as a transparent function in Ana, i.e., the intruder can extract all fields from a known message of this format (without knowing any keys). All other formats of TLS are modeled the same way. The entire TLS handshake protocol can then be represented by the following set of message patterns $M$:

$$\begin{array}{l}\mathtt{clientHello}(T_1, R_A, S, \mathit{Cipher}, \mathit{Comp}),\\ \mathtt{serverHello}(T_2, R_B, S, \mathit{Cipher}, \mathit{Comp}),\\ \mathtt{serverCert}(\mathtt{sign}(Pr_{ca}, \mathtt{x509}(B, P_B))),\\ \mathtt{clientKeyExchange}(\mathtt{crypt}(P_B, \mathtt{pmsForm}(\mathit{PMS}))),\\ \mathtt{finished}(\mathtt{prf}(\mathtt{clientFinished}(\\ \quad \mathtt{prf}(\mathtt{master}(\mathit{PMS}, R_A, R_B)), R_A, R_B, \mathtt{hash}(\mathit{HSMsgs}))))\end{array}$$

Here crypt is again asymmetric encryption, sign is signature, and master, prf and hash are one-way functions for hashing, key derivation, and MAC'ing; all other functions are formats. Most variables are of atomic type except for $P_B$ being of type $\mathsf{pub}(\mathsf{PrivateKey})$ and $\mathit{HSMsgs}$ which represents the concatenation of all handshake messages, i.e., its type is $\mathtt{concat}(\mathtt{clientHello}(\ldots), \ldots, \mathtt{finished}(\ldots))$ for yet another format concat.

One may wonder at this point how this finite set $M$ is sufficient to represent the protocol with an unbounded number of sessions. In fact, we will define below a protocol by an unbounded number of strands for the honest agents (essentially the initial state of a transition system). In fact, the sent and received messages of these strands shall be *well-typed* instances of $M$: we rename variables so that strands use pairwise disjoint sets of variables, but this renaming is well-typed, and we may instantiate some variables with ground terms, e.g., in all client strands the variable $\mathit{PMS}$ shall be instantiated with a unique constant of the according type. Collecting all messages from these strands we thus obtain an infinite set $M'$, however, $\mathit{SMP}(M) \supseteq \mathit{SMP}(M')$ since $M'$ contains only well-typed instances of $M$, and thus if $M$ is type-flaw resistant, so is $M'$. More generally, for checking that a protocol is type-flaw resistant, it is sufficient to consider any set $M$ that subsumes all messages of the protocols' honest agent strands as well-typed instances.

It is not too difficult to show that $M$ for TLS is type-flaw resistant: every operator except prf is applied to arguments of the same type throughout $\mathit{SMP}(M)$; for prf the argument is either of the form $\mathtt{clientFinished}(\cdot)$ or $\mathtt{master}(\cdot)$ (but never a variable, because prf is always applied to non-variable arguments in $M$ and it does not occur in the type of any term in $M$). Due to the free algebra, it follows almost immediately that two unifiable elements of $\mathit{SMP}(M) \setminus \mathcal{V}$ have the same type. While we have conducted the proof manually in Isabelle, we believe it is possible to automate such proofs as a general proof strategy.

### C. Constraint-level Typing Result & Formalization in Isabelle

For our typing result on the constraint-level we first prove that well-typedness and type-flaw resistance are invariants of the constraint reduction:

*Lemma 6 (Invariants):* If $(\mathcal{A}, \theta)$ is well-formed, $\mathcal{A}$ is type-flaw resistant, $\theta$ is well-typed, and $(\mathcal{A}, \theta) \rightsquigarrow^* (\mathcal{A}', \theta')$, then $\mathcal{A}'$ is type-flaw resistant and $\theta'$ is well-typed.

Recall that by Lemma 4, every simple constraint has an interpretation; we now show that it even has a well-typed interpretation. This is because the intruder can generate terms of any type (as he knows constants of any type and can compose with public functions).

*Lemma 7 (Simple intruder strands are well-typed satisfiable):* If $(\mathcal{A}, \theta)$ is well-formed, $\mathcal{A}$ is simple, and $\theta$ is well-typed, then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models_c (\mathcal{A}, \theta)$.
In fact, the proof is constructive, using this interpretation:

$$\mathcal{I}_{\mathit{simple}} \equiv \lambda v. \varepsilon t. \Gamma(v) = \Gamma(t) \wedge \emptyset \vdash_c t$$

where $\varepsilon$ is the Hilbert operator, i.e. $\varepsilon t.\ \phi$ yields a value $t$ such that $\phi$ holds, and $\emptyset \vdash_c t$ means that the intruder can generate $t$ without any prior knowledge except for the public constants. Since all intruder deduction constraints are on the form $M \vdash_c x$ all variables of the same type can safely be interpreted as the same public, ground term.[5]

From this we get the typing result on the constraint level:

*Theorem 3 (Existence of well-typed attacks, on the constraint-level):* If $(\mathcal{A}, \theta)$ is well-formed and type-flaw resistant, $\theta$ is well-typed, and $\mathcal{I} \models_c (\mathcal{A}, \theta)$, then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models_c (\mathcal{A}, \theta)$.
The proof idea is that all terms in the constraints reduction are elements of $\mathit{SMP}(\mathcal{A})$ and thus any unifier between non-variable terms must be well-typed.

## VI. Protocol Transition Systems

The previous sections have established the typing result on the level of constraints and we now lift it to transition systems. Since we had out-sourced the entire question of analysis, we also have to take care of it now.

[5] It is possible to extend this result to include inequalities like $x \neq y$ as in Almousa et al. [1], by simply ensuring that $\mathcal{I}_{\mathit{simple}}$ chooses different terms for each variable; this is possible since the intruder has an infinite supply of every atomic type.

### A. *Definitions*

We now represent also the honest agents by strands (reusing the definition of intruder strands), and we define a protocol to be a countably infinite set of such honest agent strands:

$$\textbf{type-synonym}\ (\Sigma, \mathcal{V})\ \textit{protocol} \equiv (\Sigma, \mathcal{V})\ \textit{strand set}$$

As is usual we allow the intruder full control of all communication happening in the protocol: whenever an honest agent receives a message the intruder must have sent it, and whenever an honest agent sends a message the intruder intercepts it. Hence, for protocol execution, we define a symbolic transition system in which honest agents can send and receive messages (that might contain variables, hence *symbolic*) and where we record the steps taken during these transitions. A *state* $(\mathcal{S}; \mathcal{A})$ then consists of a protocol $\mathcal{S}$ and an intruder strand $\mathcal{A}$ which represents the steps taken from the intruder's point of view and which we will build up during execution of $\mathcal{S}$. Since the goal of this section is to lift the typing result to the transition system, where we use the full semantics $\models$, we interpret the constraints in states under $\models$ and *not* $\models_c$ as we did in previous sections. For the *initial state* the intruder strand is empty, that is $(\mathcal{S}_0; 0)$ where $\mathcal{S}_0$ denotes the initial protocol and where the empty intruder strand 0 will be filled during transitions.

*Definition 9 (Protocol transition system):*

$$\begin{array}{ll} \textit{TS}_1: & (\mathcal{S};\mathcal{A}) \Rightarrow^{\bullet} (\mathcal{S} \setminus \{0\}; \mathcal{A}) \text{ if } 0 \in \mathcal{S} \\ \textit{TS}_2: & (\mathcal{S};\mathcal{A}) \Rightarrow^{\bullet} (\{S\} \cup (\mathcal{S} \setminus \{\texttt{Send}\ t.S\}); \mathcal{A}.\texttt{Receive}\ t) \\ & \quad \text{if } \texttt{Send}\ t.S \in \mathcal{S} \\ \textit{TS}_3: & (\mathcal{S};\mathcal{A}) \Rightarrow^{\bullet} (\{S\} \cup (\mathcal{S} \setminus \{\texttt{Receive}\ t.S\}); \mathcal{A}.\texttt{Send}\ t) \\ & \quad \text{if } \texttt{Receive}\ t.S \in \mathcal{S} \end{array}$$

The first rule simply removes empty strands, i.e., honest agents that have finished execution. The second rule allows honest agents to send messages, in which case the intruder intercepts and receives this message. Hence we extend the intruder knowledge (by adding a Receive step to the intruder strand) at that point with the message that is sent. The third and final rule allows an honest agent to receive a message, and in this case we require that the intruder must generate this message. Thus we extend the intruder strand with an additional derivation requirement by adding a Send step. As usual we write $\Rightarrow^{\bullet *}$ for the reflexive transitive closure.

Note that we require intruder strands to be well-formed, including those emerging from an execution of a protocol. For this reason, we impose a requirement on the variables in all honest-agent strands of the protocols we consider that is dual to the requirement for intruder strands: while in the intruder strands all variables must originate in a Send step, we require that in an honest agent strand they are all originating in a Receive step. Formally, we define the *dual* of a strand $S$ as "swapping" the direction of the steps of $S$:

$$\begin{array}{rcl} \texttt{dual}_{\texttt{st}}\ 0 & = & 0 \\ \texttt{dual}_{\texttt{st}}\ (\texttt{Send}\ t.S) & = & \texttt{Receive}\ t.(\texttt{dual}_{\texttt{st}}\ S) \\ \texttt{dual}_{\texttt{st}}\ (\texttt{Receive}\ t.S) & = & \texttt{Send}\ t.(\texttt{dual}_{\texttt{st}}\ S) \end{array}$$

Then we define protocol well-formedness using Definition 2:

*Definition 10:* A protocol $\mathcal{S}$ is *well-formed* iff $\texttt{wf}_{\texttt{sts}}\ \mathcal{S}$ where

$$\texttt{wf}_{\texttt{sts}}\ \mathcal{S} \equiv \forall S \in \mathcal{S}.\ \texttt{wf}_{\texttt{st}}\ \emptyset\ (\texttt{dual}_{\texttt{st}}\ S)$$

It is now immediate that all intruder strands of reachable states are well-formed if the initial protocol is well-formed.

### B. *Problems of the Original Paper*

Recall that in our Isabelle formalization of the lazy intruder, we have decided to "out-source" the analysis step from the intruder to the transition system. Therefore, we need to now show that the transition system from the previous section (that assumes the full intruder in its semantics) is equivalent to a transition system where the intruder is restricted to composition steps (i.e., $\vdash_c$) and that has special transition steps for analysis—and make that work with the typing result. Upon trying to prove these results in Isabelle we discovered several problems in the result of Almousa et al. [1]. In fact, that paper handles analysis as part of the lazy intruder, but the problems appear in similar form. In fact, discovering and provably fixing all such mistakes is indeed the main goal of the Isabelle formalization. We discuss first the errors and ways to fix them, and then how other typing results are doing on these issues.

The lazy intruder analysis rule of [1] would in our notation look like this:

$$\begin{array}{lr} (\textit{Decompose}_{LI}) & (\mathcal{A}.\mathcal{A}', \theta) \rightsquigarrow (\mathcal{A}.\texttt{Send}\ K.\texttt{Receive}\ T.\mathcal{A}', \theta) \\ & \text{if } s \in \texttt{ik}_{\texttt{st}}\ \mathcal{A}, \texttt{Ana}\ s = (K, T), T \nsubseteq \texttt{ik}_{\texttt{st}}\ \mathcal{A} \end{array}$$

Here we use Send $K$ and Receive $T$ for sets $K$ and $T$ of messages as obvious abbreviation for sequences of send and receive steps. The rule means, at any point in an intruder strand, the intruder can attempt the analysis of a term $s$ that he learned before that point, and this attempt would mean that he has to generate ("Send") the key terms $K$ and would obtain ("Receive") the resulting messages $T$. (In fact, our handling of analysis as part of the transition system adds analysis steps that similarly produce such sending and receiving steps in the intruder strand.)

Like [1], we make the following requirement on the Ana function that all key and result terms are subterms of the term being analyzed:

$$\texttt{Ana}_1: \texttt{Ana}\ t = (K, T) \Longrightarrow K \cup T \subset \texttt{subterms}\ t$$

This is necessary for termination, since without such a restriction to subterms one could encode undecidable problems into analysis. This is however not enough as our first counterexample to correctness of [1] shows:

*Example 5:* Suppose for two public unary operators $f$ and $g$ we define: $\texttt{Ana}\ f(g(x)) = (\emptyset, \{x\})$. Then the constraint $\texttt{Receive}\ g(c).\texttt{Send}\ c$ has a solution since $\{g(c)\} \vdash c$. This solution would however be missed by $(\textit{Decompose}_{LI})$, thus the lazy intruder of [1] is incomplete. □

The same problem does not occur in other typing results, or works with lazy intruder constraints [2], [9] because they consider a fixed set of operators where none has a destructor-like behavior upon analysis. A property of analysis that all these approaches use is that the intruder does not learn anything new from analyzing terms that he composed himself, e.g., encrypting a term and then decrypting it will not reveal new information, and without loss of generality we thus can exclude intruder-composed terms from analysis. Also [1] uses this argument, but as example 5 shows, this is not true for all intruder theories they allow. We thus make an additional restriction on $\mathtt{Ana}$, namely that analysis can only yield *direct* subterms:

$$\mathtt{Ana}_2\colon \mathtt{Ana}\ f(t_1,\ldots,t_n) = (K,T) \Longrightarrow T \subseteq \{t_1,\ldots,t_n\}$$

*Example 6:* Let now $f$ be a binary operator with the following $\mathtt{Ana}$ rule:

$$\mathtt{Ana}\ \mathsf{f}(s,t) = (\emptyset,\{t\}) \text{ if } s \in \mathcal{V}$$

This is hardly a reasonable analysis rule since it gives results only for symbolic terms, but not for ground terms. The constraint $\mathtt{Send}\ x.\mathtt{Receive}\ f(x,c).\mathtt{Send}\ c$ has no solution since there is no interpretation $\mathcal{I}$ with $\mathcal{I}\ \{f(x,c)\} \vdash c$. However, constraint reduction with rule $(\mathit{Decompose}_{LI})$ yields a simple (and thus satisfiable constraint), and we thus also have a counter-example for soundness of [1]. □

To correct this, we add the following requirement:

$$\mathtt{Ana}_3 : \mathtt{Ana}\ t = (K,T) \neq (\emptyset,\emptyset) \Longrightarrow \mathtt{Ana}\ (\delta\ t) = (\delta\ K, \delta\ T) \quad \text{for any substitution } \delta$$

Thus, when applying $\mathtt{Ana}$ on any analyzable term $t$, then any instance $\delta\ t$ must allow for the same analysis under $\delta$.

*Example 7:* Consider again our standard $\mathtt{Ana}$ (which satisfies all three requirements). For the full intruder model $\vdash$ (that is not restricted to composition only) the lazy intruder with $(\mathit{Decompose}_{LI})$ is not complete: The constraint $\mathtt{Send}\ x.\mathtt{Receive}\ \mathsf{crypt}(x,\mathsf{c}).\mathtt{Send}\ \mathsf{c}$ has the solution $\mathcal{I} = [x \mapsto \mathsf{pub}(\mathsf{c}')]$ for some constant $\mathsf{c}'$. However, this solution is not found by the lazy intruder (with the above analysis rule) because $\mathtt{Ana}\ \mathsf{crypt}(x,\mathsf{c}) = (\emptyset,\emptyset)$. The problem is that the case $\mathtt{Ana}\ \mathsf{crypt}(\mathsf{pub}(k),m)$ does not *match* the term we need to analyze, since it has the variable $x$ in the key position. One may wonder if the authors of [1] actually meant to apply this rule under *unification* with a term in the intruder knowledge, however that would require to apply the unifier (in the example $[x \mapsto \mathsf{pub}(x')]$ for a new $x'$) to the rest of the constraint—while all other rules of [1] explicitly denote such unifiers; moreover this reading of the rule would lead to non-termination. □

In the other typing results [2], [9], this problem does not occur because they fix the public-key infrastructure, i.e., they cannot model that an honest agent receives an arbitrary public key $x$ in a message and use it for encrypting a message, i.e., $\mathsf{crypt}(x,m)$. When fixing the public key infrastructure, all keys used for public key encryption are of the form $\mathsf{pub}(\cdot)$ (in our notation) and then the mentioned problem does not occur. However, we do not want to impose this strong restriction to a fixed public key infrastructure and rather allow for protocols that can also exchange public keys. A milder restriction is that all terms used as a first argument of $\mathsf{crypt}$ must have the form $\mathsf{pub}(\cdot)$, for instance the strand of an honest agent could be: $\mathtt{Receive}\ \mathsf{pub}(x).\mathtt{Send}\ \mathsf{crypt}(\mathsf{pub}(x),\mathsf{c})$. The restriction here is that this agent only accepts a public key as input, i.e., restricting this bit to a typed model by assumption. There are several ways to justify this restriction, e.g., it is common in protocols where a new public key $t$ can be introduced that the creator has to sign any message with the corresponding private key, proving that $t = \mathsf{pub}(s)$ for some private key $s$ (and without the recipient learning $s$). Also, when receiving a public key as part of a certificate from a trusted authority, one may rely that the authority has required this kind of proof from the owner of the public key, and thus it is justifiable to model the certified key to have the form $\mathsf{pub}(\cdot)$.[6]

While the pub-requirement solves the problem for the concrete example crypt, we need a general requirement for arbitrary operators. The example shows that this cannot be a property of $\mathtt{Ana}$ alone, but relates to the use of the operators in the protocol:

*Definition 11 (Analysis-invariance):* A protocol $\mathcal{S}_0$ is *analysis-invariant* iff

$$\begin{aligned}&\forall t \in (\mathtt{subterms}\ M) \setminus \mathcal{V}.\ \forall K,T,\delta.\\ &\mathtt{Ana}\ t = (K,T) \longrightarrow \mathtt{Ana}\ (\delta\ t) = (\delta\ K, \delta\ T)\end{aligned}$$

where $M$ is the set of *sent* messages occurring in $\mathcal{S}_0$.

Thus we require that any subterm $t$ of the protocol, except variables, can be analyzed if some instance $\delta\ t$ can be analyzed. This excludes a term like $\mathsf{crypt}(x,t)$ since it cannot be analyzed while the instance $\mathsf{crypt}(\mathsf{pub}(c),t)$ can. In general, this restriction affects only those operators $f$ where the analysis rule has the form $\mathtt{Ana}\ f(t_1,\ldots,t_n)$ where some $t_i$ is not a pattern variable; then the protocol cannot use a variable for that argument.

These restrictions are sufficient to conclude the typing result on the transition system level, as described next, and they still support strictly more protocols than the previous typing results (except the flawed [1]).

### C. Handling Analysis

As an intermediate step towards the result, we now define a second transition system $\Rightarrow^\bullet_c$ similar to $\Rightarrow^\bullet$, but where the intruder does not handle analysis himself (interpreting constraints under $\models_c$ instead of the full $\models$ as in $\Rightarrow^\bullet$) and where we have special transitions for analysis. An easy way to handle this would be to simply define a set of honest agents that behave like the analysis functionality, e.g., $\mathtt{Receive}\ \mathsf{crypt}(\mathsf{pub}(x),y).\mathtt{Receive}\ x.\mathtt{Send}\ y$. In fact, this works fine (and does not even need the requirement of analysis invariance we introduced before) as far as the equivalence to the standard transition system $\Rightarrow^\bullet$ is concerned. However this does not directly work with the typing result: the notion of type-flaw resistance would have to be satisfied on the set of all honest agent strands, including the ones for analysis. This

[6] Many approaches have other models of the relation between public and private keys, e.g., mappings on constants that are not part of $\Sigma$, or a private function from public to private keys. All these seem to have trouble with either the lazy intruder or the typing result.

would be violated for many reasonable protocols (that have no type-flaw problems). Luckily, there is a (more complicated) solution that requires no further restriction on protocols.

The idea is that the intruder is allowed to attempt analysis for every non-variable subterm of a term in his knowledge. (Note that includes subterms he may be unable to derive, but as part of the analysis step he has to prove he can produce them, so this is sound.) Thus, the $\Rightarrow_c^\bullet$ is defined like $\Rightarrow^\bullet$ plus the following additional rule:

$$\begin{aligned} \mathit{TS}_4^c: \quad & (\mathcal{S};\mathcal{A}) \Rightarrow_c^\bullet (\mathcal{S};\mathcal{A}.\mathtt{Send}\ t.\mathtt{Send}\ k_1.\cdots.\mathtt{Send}\ k_m.\\ & \qquad\qquad\qquad \mathtt{Receive}\ s_1.\cdots.\mathtt{Receive}\ s_n)\\ & \text{where } t \in (\mathtt{subterms}\ (\mathtt{ik_{st}}\ \mathcal{A})) \setminus \mathcal{V}\\ & \text{and } \mathtt{Ana}\ t = (\{k_1,\dots,k_m\},\{s_1,\dots,s_n\}) \end{aligned}$$

*Example 8:* Consider the protocol $\mathcal{S}_0 = \{S_1, S_2, S_3\}$ where

$$\begin{aligned} S_1 &= \mathtt{Send}\ \mathsf{k_a}.\mathtt{Send}\ \mathsf{scrypt}(\mathsf{k_b}, \mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}), \mathsf{secret}))\\ S_2 &= \mathtt{Receive}\ \mathsf{scrypt}(\mathsf{k_b}, x).\mathtt{Send}\ x\\ S_3 &= \mathtt{Receive}\ \mathsf{secret} \end{aligned}$$

Here, the strand $S_3$ represents a strand to check a secrecy goal, i.e., we want to check that we cannot reach a state where $S_3$ has executed and the intruder constraint is satisfiable. Consider the execution of the steps of $S_1$ and $S_2$, i.e., $(\mathcal{S}_0; 0) \Rightarrow_c^{\bullet *} (\{S_3\}; \mathcal{A})$ where

$$\begin{aligned} \mathcal{A} = {} & \mathtt{Receive}\ \mathsf{k_a}.\mathtt{Receive}\ \mathsf{scrypt}(\mathsf{k_b}, \mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}), \mathsf{secret}))\\ & \mathtt{Send}\ \mathsf{scrypt}(\mathsf{k_b}, x).\mathtt{Receive}\ x \end{aligned}$$

For the intruder to obtain the secret, we can now make an analysis step with rule $(\mathit{TS}_4^c)$ for the term $t = \mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}), \mathsf{secret})$, yielding the state $(\{S_3\}; \mathcal{A}.\mathcal{D})$ with

$$\mathcal{D} = \mathtt{Send}\ \mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}), \mathsf{secret}).\mathtt{Send}\ \mathsf{k_a}.\mathtt{Receive}\ \mathsf{secret}$$

and then execute $S_3$, yielding state $(\emptyset; \mathcal{A}.\mathcal{D}.\mathcal{A}')$ with $\mathcal{A}' = \mathtt{Send}\ \mathsf{secret}$. This constraint $\mathcal{A}.\mathcal{D}.\mathcal{A}'$ is satisfiable in $\models_c$.

In the standard transition system the corresponding state would just omit the analysis step, i.e., $(\mathcal{S}_0; 0) \Rightarrow^{\bullet *} (\emptyset; \mathcal{A}.\mathcal{A}')$. This constraint $\mathcal{A}.\mathcal{A}'$ is satisfiable for the full intruder $\models$. □
More generally, we prove in Isabelle that the two transition systems are equivalent. In particular, for every reachable state $(\mathcal{S}; \mathcal{A})$ of $\Rightarrow^\bullet$ and every solution $\mathcal{I} \models \mathcal{A}$, an equivalent state $(\mathcal{S}; \mathcal{A}')$ of $\Rightarrow_c^\bullet$ is reachable where $\mathcal{A}'$ is like $\mathcal{A}$ augmented with analysis steps, and $\mathcal{I} \models_c \mathcal{A}'$. From the initial state $(\mathcal{S}_0; 0)$ this can be stated as follows:

*Lemma 8 (Equivalence of transition systems, part 1):* If protocol $\mathcal{S}_0$ is well-formed and analysis-invariant, $(\mathcal{S}_0; 0) \Rightarrow^{\bullet *} (\mathcal{S}; \mathcal{A}_1.\cdots.\mathcal{A}_n)$, and $\mathcal{I} \models \mathcal{A}_1.\cdots.\mathcal{A}_n$ where each $\mathcal{A}_i$ emerged from exactly one application of $(\mathit{TS}_1)$, $(\mathit{TS}_2)$, or $(\mathit{TS}_3)$, then there exists $\mathcal{D}_1, \dots, \mathcal{D}_{n-1}$ such that $(\mathcal{S}_0; 0) \Rightarrow_c^{\bullet *} (\mathcal{S}; \mathcal{A}_1.\mathcal{D}_1.\cdots.\mathcal{A}_{n-1}.\mathcal{D}_{n-1}.\mathcal{A}_n)$ and $\mathcal{I} \models_c \mathcal{A}_1.\mathcal{D}_1.\cdots.\mathcal{A}_{n-1}.\mathcal{D}_{n-1}\mathcal{A}_n$ and where each $\mathcal{D}_i$ emerged from zero or more applications of $(\mathit{TS}_4^c)$.
The most complicated aspect of the proof is to show that, if $\mathcal{I}\ (\mathtt{ik_{st}}\ \mathcal{A}) \vdash \mathcal{I}\ t$ for some intruder strand $\mathcal{A}$ with model $\mathcal{I}$ and some term $t$, then there exists some sequence of $(\mathit{TS}_4^c)$ steps $\mathcal{D}$ such that $\mathcal{I}\ (\mathtt{ik_{st}}\ (\mathcal{A}.\mathcal{D})) \vdash_c \mathcal{I}\ t$ and where $\mathcal{I}$ is also a model of $\mathcal{A}.\mathcal{D}$. This proof proceeds by an induction on the derivation of $\mathcal{I}\ t$. Not surprisingly, this case bears many similarities to completeness proofs of lazy intruder constraint reduction systems like [1], [9] where the intruder can analyze terms. The most complicated case—both in our proof and the proofs of completeness—is where the last step of the derivation is an application of the (*Decompose*) rule, i.e., where $\mathcal{I}\ t$ is derived by analyzing another ground term $t'$. In the completeness proofs we would in this case have to inspect the derivation tree for $t'$, eliminate redundant parts (namely, analysis of intruder-composed terms), and, in the case where the last step in the derivation is yet another application of (*Decompose*), regress to a point in the derivation tree for $t'$ where no (*Decompose*) has occurred yet. In our setting, because we have a clear separation between term analysis and composition (because of the out-sourcing analysis and by considering the sub-relation $\vdash_c$ of $\vdash$), we immediately get from the induction hypothesis that there exists some $\mathcal{D}'$ (where $\mathcal{I}$ is still a model of $\mathcal{A}.\mathcal{D}'$) such that $\mathcal{I}\ (\mathtt{ik_{st}}\ (\mathcal{A}.\mathcal{D}')) \vdash_c t'$. Hence we essentially perform the regression by simply applying the induction hypothesis instead of inspecting and transforming derivation trees, making the proof slightly easier.

The other direction of the equivalence is more straightforward and does not require any assumptions on the protocol. It follows easily by an induction on reachability:

*Lemma 9 (Equivalence of transition systems, part 2):* If $(\mathcal{S}_0; 0) \Rightarrow_c^{\bullet *} (\mathcal{S}; \mathcal{A}_1.\mathcal{D}_1.\cdots.\mathcal{A}_n.\mathcal{D}_n)$ and $\mathcal{I} \models_c \mathcal{A}_1.\mathcal{D}_1.\cdots.\mathcal{A}_n.\mathcal{D}_n$ where each $\mathcal{A}_i$ emerged from exactly one application of $(\mathit{TS}_1)$, $(\mathit{TS}_2)$, or $(\mathit{TS}_3)$, and where each $\mathcal{D}_i$ emerged from zero or more applications of $(\mathit{TS}_4^c)$, then $(\mathcal{S}_0; 0) \Rightarrow^{\bullet *} (\mathcal{S}; \mathcal{A}_1.\cdots.\mathcal{A}_n)$ where $\mathcal{I} \models \mathcal{A}_1.\cdots.\mathcal{A}_n$.

### *D. Lifting the Typing Result*

With this equivalence between the transition systems proven, we can now lift the typing result of Theorem 3 to constraints reachable in $\Rightarrow^\bullet$ where these constraints are interpreted under the full intruder $\models$ instead of $\models_c$. First we define that an entire protocol $\mathcal{S}_0$ (a set of strands for honest agents) is type-flaw resistant if the set $M$ of all sent and received messages of $\mathcal{S}_0$ is type-flaw resistant. It is now immediate that all intruder strands reachable from $(\mathcal{S}_0; 0)$ in both transition systems we defined (including analysis steps) are also type-flaw resistant, because the set of sub-message patterns are closed under subterms.

We can now first apply Lemma 8 to any satisfiable reachable state $(\mathcal{S}; \mathcal{A}_1.\cdots.\mathcal{A}_n)$ in $\Rightarrow^\bullet$ to obtain an equivalent state $(\mathcal{S}; \mathcal{A}_1.\mathcal{D}_1.\cdots.\mathcal{A}_n.\mathcal{D}_n)$ with the same solution reachable in our intermediate transition system $\Rightarrow_c^\bullet$. Then we can lift the typing result from the constraint level to $\Rightarrow_c^\bullet$, since here constraints are interpreted in $\models_c$, i.e., solving the constraints does not require analysis steps and thus our constraint-level typing result Theorem 3 applies. Then, by the equivalence to $\Rightarrow^\bullet$ with the full intruder model $\models$, i.e. Lemma 9, we obtain our main result that every reachable state of a type-flaw resistant and analysis-invariant protocol has a solution iff it has a well-typed one:

*Theorem 4 (Existence of well-typed attack, on transition system level):* If protocol $\mathcal{S}_0$ is well-formed, type-flaw resistant and analysis-invariant, $(\mathcal{S}_0; 0) \Rightarrow^{\bullet *} (\mathcal{S}; \mathcal{A})$, and $\mathcal{I} \models \mathcal{A}$, then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models \mathcal{A}$.

## VII. Conclusion

We have established a typing result in Isabelle: Given an Isabelle proof of security of a protocol where the intruder is limited to well-typed messages (e.g., like proofs in the works of [21] and [4]), then the typing result allows us to lift this proof to an intruder model without the restriction to well-typed messages. As an example, we have proved that the type-flaw resistance requirement of our result is indeed satisfied by the TLS protocol.

The particular value of this is the high reliability of proofs checked with Isabelle, in contrast to pen-and-paper proofs in partially natural language. This is illustrated by several errors we discovered in the pen-and-paper proofs of Almousa et al. [1]. Strictly speaking, their result does not hold without further restrictions on the supported operators and protocols. The complexity of such results (as well as verification tools) makes such mistakes likely and this bears the risk of accepting false security proofs. The Isabelle proof of the typing result under some additional restrictions is thus also a step towards "cleaning up".

In other typing results, namely Cortier and Delaune [9] and Arapinis and Duflot [2], the problems of [1] do not arise, since they have fixed public key infrastructures (and fixed sets of supported operators). Our restrictions in contrast do allow also protocols where public keys are exchanged, though one must ensure or assume that the received terms are indeed public keys, but this is often in our opinion a realistic restriction.

At the core of our result, is the formalization of the lazy intruder and its correctness. This, as well as the proving of the typing result, gives insights for modeling and proving protocols in general: Since Isabelle forces one to be precise about every single detail, one is compelled to abstract, generalize, and simplify as far as possible, to reduce the formalization to the absolute essence. We have simplified the constraint representation and shown how to "out-source" the analysis steps of the intruder to steps in the protocol transition system. We believe that such insights are helpful beyond the result itself.

The typing results can also used as a stepping stone for compositional reasoning, e.g., [9], [1] prove that two protocols that are secure in isolation can also run securely on the same communication medium in parallel, if their messages do not interfere with each other, a requirement closely related to the typing result. We plan to formalize such a result in Isabelle as future work.

*Acknowledgments:* This work was supported by the Sapere-Aude project "Composec: Secure Composition of Distributed Systems", grant 4184-00334B of the Danish Council for Independent Research. We thank Luca Viganò, Achim Brucker, and Anders Schlichtkrull for helpful comments and discussions.

## References

[1] O. Almousa, S. Mödersheim, P. Modesti, and L. Viganò. Typing and compositionality for security protocols: A generalization to the geometric fragment. In *ESORICS 2015*, pages 209–229, 2015. Extended version available at http://www.imm.dtu.dk/~samo/.

[2] M. Arapinis and M. Duflot. Bounding messages for free in security protocols - extension to various security properties. *Inf. Comput.*, 239:182–215, 2014.

[3] D. A. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005.

[4] G. Bella. *Formal Correctness of Security Protocols - With 62 Figures and 4 Tables*. Information Security and Cryptography. Springer, 2007.

[5] G. Bella, F. Massacci, and L. C. Paulson. Verifying the SET purchase protocols. *J. Autom. Reasoning*, 36(1-2):5–37, 2006.

[6] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW 2001*, pages 82–96, 2001.

[7] B. Blanchet and A. Podelski. Verification of cryptographic protocols: tagging enforces termination. *Theor. Comput. Sci.*, 333(1-2):67–90, 2005.

[8] A. D. Brucker and S. Mödersheim. Integrating automated and interactive protocol verification. In *FAST 2009*, pages 248–262, 2009.

[9] V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, 2009.

[10] T. Dierks and E. Rescorla. RFC 5246: The Transport Layer Security (TLS) Protocol, Version 1.2, 2008. Available: http://tools.ietf.org/rfc/rfc5246.txt.

[11] G. Gonthier and M. Norrish, editors. *Certified Programs and Proofs - Third International Conference, CPP 2013, Melbourne, VIC, Australia, December 11-13, 2013, Proceedings*, volume 8307 of *Lecture Notes in Computer Science*. Springer, 2013.

[12] J. Goubault-Larrecq. Towards producing formally checkable security proofs, automatically. In *Computer Security Foundations Symposium*, 2008.

[13] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[14] B. Huffman and O. Kuncar. Lifting and transfer: A modular design for quotients in Isabelle/HOL. In Gonthier and Norrish [11], pages 131–146.

[15] R. Küsters and T. Truderung. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *CSF*, pages 157–171. IEEE, 2009.

[16] S. Meier, C. Cremers, and D. A. Basin. Efficient construction of machine-checked symbolic protocol security proofs. *Journal of Computer Security*, 21(1):41–87, 2013.

[17] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 166–175, 2001.

[18] S. Mödersheim. Diffie-Hellman without difficulty. In *FAST*, pages 214–229, 2011.

[19] S. Mödersheim. Deciding security for a fragment of ASLan. In *ESORICS*, pages 127–144. Springer, 2012.

[20] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[21] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[22] L. C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.

[23] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *Theor. Comput. Sci.*, 299, 2003.

[24] A. Schropp and A. Popescu. Nonfree datatypes in Isabelle/HOL - animating a many-sorted metatheory. In Gonthier and Norrish [11], pages 114–130.

[25] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1):191–230, 1999.

[26] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009.