Computationally Sound Secrecy Proofs by Mechanized Flow Analysis

Michael Backes
Department of Mathematics
and Computer Science
Saarland University
backes@cs.uni-sb.de

Peeter Laud
Department of Mathematics
and Computer Science
Tartu University
peeter.laud@ut.ee

August 10, 2006

Abstract

We present a novel approach for proving secrecy properties of security protocols by mechanized flow analysis. In contrast to existing tools for proving secrecy by abstract interpretation, our tool enjoys cryptographic soundness in the strong sense of blackbox reactive simulatability/UC which entails that secrecy properties proven by our tool are automatically guaranteed to hold for secure cryptographic implementations of the analyzed protocol, with respect to the more fine-grained cryptographic secrecy definitions and adversary models.

Our tool is capable of reasoning about a comprehensive language for expressing protocols, in particular handling symmetric encryption and asymmetric encryption, and it produces proofs for an unbounded number of sessions in the presence of an active adversary. We have implemented the tool and applied it to a number of common protocols from the literature.

1 Introduction

Security proofs of cryptographic protocols are known to be difficult and the automation of such proofs has been studied soon after the first protocols were developed. From the start, the actual cryptographic operations in such proofs were idealized into so-called Dolev-Yao models, following [21, 22, 39], e.g., see [30, 47, 1, 37, 44, 12]. This idealization simplifies proof construction by freeing proofs from cryptographic details such as computational restrictions, probabilistic behavior, and error probabilities. Conducting secrecy proofs by typing based on these abstractions has shown to be a particularly salient technique as it allowed for elegant and fully automated proofs, often even for an unbounded number of sessions.

A type system was recently presented in [34] that combines the conciseness of language-based reasoning in Dolev-Yao models with strong computational soundness guarantees, i.e., if an abstract protocol typechecks then its cryptographic realization provably keeps the quantities handed to it by the protocol users secret in the computational sense. Such computational soundness guarantees of abstract proofs have recently been identified as central for gaining trustworthy guarantees of security protocols: the computational model strives for stronger, more fine-grained security notions and furthermore considers a more realistic adversary that is allowed to perform arbitrary bitstring manipulations as long as they can be performed in probabilistic polynomial-time. However, despite being the first type system that allows for abstract, computationally sound reasoning under active attacks, the major drawback of [34] was that type inference was not considered. As a consequence, this work did not entail an automated procedure for analyzing secrecy aspects of cryptographic protocols with cryptographic soundness guarantees, which arguably is the central goal of unifying the advantages of both approaches.

We remedy this shortcoming by presenting a mechanized approach for soundly proving secrecy properties of cryptographic protocols by analysing the possible flows of data during the execution of the protocol. Our approach is capable of reasoning about a comprehensive language for expressing protocols, in particular handling symmetric encryption and asymmetric encryption, allows for more precise analyses compared with the type system of [34], is fully automated, and produces proofs for an unbounded number of sessions in the presence of an active adversary.

Our results (and the one of [34] as well) rely on a variant of the Dolev-Yao model of Backes, Pfitzmann, and Waidner, henceforth called the BPW model, which has been shown to be computationally sound in the strong sense of of blackbox reactive simulatability (BRSIM). The security notion of BRSIM means that one system (here, the cryptographic realization) can be plugged into arbitrary protocols instead of another system (here, the BPW model) [45, 17]; it is also called UC for its universal composition properties. While first security proofs of several common protocols have been hand-crafted using the BPW model [7, 6], recent work has shown that the BPW model is accessible to theorem proving techniques as well [48]. Our work shows that soundly proving secrecy properties in a fully automated manner is possible using the BPW model, and it identifies cryptographically sound secrecy by typing as a promising direction for future work in general. In particular, our line complements the large number of existing works that aims at establishing computational soundness of Dolev-Yao models without considering secrecy by typing, cf. the section on related work for more details.

The analysis presented in this paper builds on the spi-calculus-style language, its deterministic semantics and the corresponding type system from [34] and is inspired by methods from control flow analysis. It works by collecting for each defined variable at each protocol point the possible shapes of terms that this variable may point to, including the possible creation points of the atomic subterms. The same information is also collected for channels between participants for encryption keys, thus yielding information which terms may be communicated over which channel, and which terms may be encrypted with which keys by honest participants, respectively. Finally, the same abstraction is also collected for terms that the adversary may learn during the run of the protocol.

There are a couple of noteworthy points. First, all inputs from the adversary are modeled using a single abstract value, thus freeing the analyser from the necessity to model every new term that the adversary may construct. Instead, we consider explicit rules for decomposing this abstract value, i.e., the adversary's input, which allows us to keep the description of the adversary's knowledge finite. Secondly, parts of the protocol statically following a public-key decryption are analysed twice — once assuming that the ciphertext was created by an honest participant and once assuming that it was created by the adversary. The distinction of these two cases (which was already present in [3] and also in [34]) is important for the precision of the analyser. Thirdly, we collect not only the possible values of variables but also relationships between them. Whenever certain operations restrict the set of possible values of some variables, we exploit these recorded relationships in order to restrict the set of values of related variables as well. This collection of relationships is reminiscent of shape analysis [50], although our task is considerably simpler here than a full shape analysis because we do not have destructive updates. We record the relationships between variables by collecting a set of constraints that their abstractions must satisfy.

Our prover (consisting of constraint generator and solver) has been implemented in O'Caml and can be downloaded at `http://www.ut.ee/~peeter_l/research/brsiman`.

Related Work Early work on linking Dolev-Yao-style symbolic models and cryptography [5, 27, 31, 28] only considered passive attacks, and therefore cannot make general statements about protocols.

The security notion of BRSIM was first defined generally in [45], based on simulatability definitions for secure (one-step) function evaluation. It was extended in [46, 17], the latter with somewhat different details and called UC (universal composability), and has been widely applied to prove individual cryptographic systems secure and to derive general theoretical results. In particular, BRSIM/UC allows for plugging one system into arbitrary protocols instead of another system while retaining essentially arbitrary security properties [45, 17, 9].

A cryptographic justification of a Dolev-Yao model in the sense of BRSIM/UC was first given in [10] with extensions in [11, 8]. Later papers [40, 33, 18] considered to what extent restrictions to weaker security properties or less general protocol classes allow simplifications compared with [10]: Laud [33] has presented cryptographic foundations for a Dolev-Yao model of symmetric encryption but specific to certain confidentiality properties where the surrounding protocols are restricted to straight-line programs. Warinschi et al. [40, 19] have presented cryptographic underpinnings for a Dolev-Yao model of public-key encryption, yet for a restricted class of protocols and protocol properties that can be analyzed using this primitive. Baudet, Cortier, and Kremer [13] have established the soundness of specific classes of equational theories in a Dolev-Yao model under passive attacks. Canetti and Herzog [18] have shown that a Dolev-Yao-style symbolic analysis can be conducted using the framework of universal composability for a restricted class of protocols, namely mutual authentication and key exchange protocols with the additional constraint that the protocols must be expressible as loop-free programs using public-key encryption as their only cryptographic operation. We stress that none of these works build on type inference for proving secrecy properties of security protocols.

The work that comes closest to our work is the work of Laud [34] who designed a type system for proving secrecy aspects of security protocols based on the BPW model. He shows that if an abstract protocol typechecks in his system, then its cryptographic realization provably keeps the quantities handed to it by the protocol users secret in the computational sense. The proof of this fact exploits the BRSIM/UC soundness result of [10, 8, 9] for carrying over symbolic proofs of secrecy in the BPW model to the actual cryptographic realization, similar to the present paper. However, type inference has not been implemented yet in this paper so that the paper did not entail a mechanized procedure for soundly proving secrecy aspects of security protocols.

Efforts are also under way to formulate syntactic calculi with a probabilistic, polynomial-time semantics, including approaches based on process algebra [41, 36], security logics [29, 20] and cryptographic games [14]. In particular, Datta et al. [20] have proposed a promising logical deduction system to prove computational security properties. We are not aware of any implementations of these frameworks, except for Blanchet's [14], who has recently presented an automated tool for proving secrecy properties of security protocols based on transforming cryptographic games. This line of work is orthogonal to the work of justifying Dolev-Yao models, which offer a higher level of abstractions and thus much simpler proofs where applicable, so that proofs of larger systems can be automated.

Let us also mention some of the work in the area of type systems for cryptographic protocol analysis. The first type system of this kind was proposed by Abadi [2], it could be used for verifying the secrecy of payloads or nonces in the protocols using only symmetric encryption. This type system, as well as all the remaining ones that we describe work in the Dolev-Yao model. The type system was extended to cope with asymmetric encryption by Abadi and Blanchet [3]. Abadi and Blanchet [4] further generalized this type system to handle *generic* cryptographic primitives. The type system of Abadi has also been extended by Gordon and Jeffrey [24, 25, 26] to check for integrity properties. Finally, a static program analysis [32] and a type system [35] exist that work directly in the computational model, handling programs containing symmetric encryption, both for passive adversaries only.

Abstract interpretation, which is in most cases automatable using data flow analysis, has also been considered for the analysis of cryptographic protocols within the Dolev-Yao model. See for example [15] and the references contained therein.

Structure of the paper We start by describing our (machine-based) execution model and the language used to program these machines for expressing security protocols in Sec. 2. We continue in Sec. 3 with the description of the analysis. In particular, we give the correctness theorem stating under which conditions the results of the abstract analysis entail computational security of a cryptographic protocol. Sec. 4 describes the implementation of our tool and its applicability to common security protocols from the literature. In Sec. 5 we give the main technical lemma, similar to subject reduction, used to prove the previously given correctness theorem.

2 Execution Model

We use the same setup of a system as in [34]. In short, the BPW model (sometimes also called abstract cryptographic library in the following corresponding to the original title of [10]) for n honest users is implemented by a machine $\mathcal{TH}_n$ which has input ports $\mathsf{in}_{\mathsf{u}_i}?$ to receive commands from the i-th user, output ports $\mathsf{out}_{\mathsf{u}_i}!$ to return the results of commands and (handles of) received messages, ports $\mathsf{in}_{\mathsf{a}}?$ and $\mathsf{out}_{\mathsf{a}}!$ for the communication with the adversary, and a database of terms. The database records the structure of messages and the knowledge of messages by the parties (n users and the adversary). The users and the adversary access messages through *handles*, the transmission of messages involves the translation of handles. The possible commands are the construction, taking apart, and sending of messages. The protocol logic for the i-th user is implemented by a machine P_i that connects to the ports $\mathsf{in}_{\mathsf{u}_i}?$ and $\mathsf{out}_{\mathsf{u}_i}!$ and offers the ports $\mathsf{pin}_{\mathsf{u}_i}?$ and $\mathsf{pout}_{\mathsf{u}_i}!$ to the user through which it may send and receive data. An execution step of a machine P_i consists of receiving a message (either from $\mathcal{TH}_n$ or the user), performing some computations on the terms, and optionally sending a message. The machines P_i are programmed in a language resembling the spi-calculus, defined below.

$$
\begin{array}{rcllllll}
e & ::= & n & \mid \underline{\mathsf{keypair}}^{\ell} & \mid \mathsf{store}(x) & \mid \mathsf{retrieve}(x) & \mid \mathsf{list}(x_1,\ldots,x_k) \\
& \mid & x & \mid \underline{\mathsf{pubkey}}(x) & \mid \underline{\mathsf{pubenc}}^{\ell}(x_{\mathrm{k}},x_{\mathrm{t}}) & \mid \underline{\mathsf{privenc}}^{\ell}(x_{\mathrm{k}},x_{\mathrm{t}}) & \mid \underline{\pi_i^j}(x) \\
& & & \mid \mathsf{gen\_symenc\_key}(i)^{\ell} & \mid \underline{\mathsf{pubdec}}(x_{\mathrm{k}},x_{\mathrm{t}}) & \mid \underline{\mathsf{privdec}}(x_{\mathrm{k}},x_{\mathrm{t}}) & \mid \mathsf{gen\_nonce}^{\ell}
\end{array}
$$

$$
\begin{array}{rcll}
\mathit{SIP} & ::= & \mathbf{receive}_c^{\ell}[x_{\mathrm{p}}](x) & \\
\mathit{IP} & ::= & \mathit{SIP} & \mid\ !\mathit{SIP} \\
I & ::= & \mathit{IP}.P & \\
I^* & ::= & \mathbf{0} & \mid\ I \mid I^*
\end{array}
\qquad
\begin{array}{rcl}
P & ::= & I^* \\
& \mid & \mathcal{JJ} \\
& \mid & \mathbf{send}_c[x_{\mathrm{p}}](x).I^* \\
& \mid & \mathbf{let}^{\ell}\ x := e\ \mathbf{in}\ P\ \mathbf{else}\ P' \\
& \mid & \mathbf{if}^{\ell}\ x = x'\ \mathbf{then}\ P\ \mathbf{else}\ P'
\end{array}
$$

Here x-s are variables, e-s are expressions, I-s are input processes, P-s are output processes, and ℓ-s are labels for program points and expressions of interest. No label may occur twice in the protocol text, nor can a variable be defined twice or used before being defined. The language contains public-key and symmetric-key encryption as the cryptographic primitives (as well as nonces). A public and secret key pair is created by the expression $\underline{\mathsf{keypair}}$, the public key is extracted by $\underline{\mathsf{pubkey}}$. A *level* i is associated with each symmetric key to prevent encryption cycles (and make the proof relating $\mathcal{TH}_n$ and its concrete implementation go through); a symmetric key may only encrypt keys of lower level. The store- and $\mathsf{retrieve}$-expressions are used to convert payloads (data that can be communicated with the user) to handles and back. The expression $\underline{\pi_i^j}(x)$ extracts the i-th component from the list of length j pointed to by x. In $\mathbf{receive}_c[x_\mathrm{p}](x)$ and $\overline{\mathbf{send}}_c[x_\mathrm{p}](x)$, the variable x is the message and x_p is the identity of the other party. The channel for the message is given by the constant *abstract channel* c. An abstract channel is used to group messages sent between protocol participants, as well as between the protocol user and participant (although the abstract channel does not alone determine the sender and the receiver of a message). Furthermore, the set of abstract channels $\mathbf{Chan}$ is partitioned into four parts, denoted $\mathbf{Chan}_\mathrm{x}$, where $\mathrm{x} \in \{\mathsf{s}, \mathsf{a}, \mathsf{i}, \mathsf{u}\}$. If a message is sent on an abstract channel from $\mathbf{Chan}_\mathsf{s}$ [resp. $\mathbf{Chan}_\mathsf{a}$, $\mathbf{Chan}_\mathsf{i}$] then it means that the message travels between protocol participants over a secure (resp. authentic, insecure) channel. If a message is sent on an abstract channel from $\mathbf{Chan}_\mathsf{u}$ then it travels between the protocol user and the protocol participant (i.e. over one of the concrete channels $\mathsf{pin}_{\mathsf{u}_i}$ or $\mathsf{pout}_{\mathsf{u}_i}$). The variables x and x_p are bound in a **receive**-statement. The variable x is also bound in the default-branch of a **let**-statement, but not in the else-branch, which is taken upon a failure of evaluating e.

The internal state of an inactive (i.e. not currently running) P_i consists of a list of input processes together with their execution environments, giving values to already defined variables. The "program" (or initial state) of each P_i is a list of input processes. An active P_i additionally contains the received message (together with the apparent sender and the name of the channel it was received on) and the currently running (output) process (together with its environment). When P_i receives the message, it is handed over to the first input process $(!)\mathbf{receive}_c[x_\mathrm{p}](x).P$ with matching channel name c in its list of processes. The variables x and x_p are bound to the message and the apparent sender and the process executes until it has become $\mathcal{JJ}$ or a list of input processes I^*. The value $\mathcal{JJ}$ means rejecting the message — the list of input processes of P_i is not changed, the currently executing process and its environment are discarded (thereby forgetting all references to any new terms that may have been created since receiving that message) and the message is handed over to the next input process with the matching channel name in the list of input processes of P_i. When a process accepts the message, it executes until it has become a list of input processes I^*. All processes in this list I^*, together with the environment of the output process, are put to the list of input processes of P_i instead of or in addition of (depending on the presence of replication) the original process. When no process accepts the message, it is simply lost.

Security

The security property we are interested in is the *secrecy of payloads* [9]. The same property was also considered in [34] and our treatment here does not differ from theirs. In short, we want the system implementing the protocol (consisting of the machines $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and $\mathcal{TH}_n$) to retain the secrecy of any payloads handed to it by the users over the ports $\mathsf{pin}_{\mathsf{u}_i}?$. The secrecy of payloads means that the user and the adversary together cannot figure out whether the system implementing the protocol is really computing with the values received from the user or with some other values;

the scrambling is done on the channels $\mathsf{pin}_{\mathsf{u}_i}$ and descrambling on the channels $\mathsf{pout}_{\mathsf{u}_i}$. A precise definition can be found in [9] and a concise description in [34]. In [34] the following five properties were stated to be sufficient for the secrecy of payloads and for the simulatability of the machine $\mathcal{TH}_n$:

(I) the bit-strings that the machines P_i receive from the ports $\mathsf{pin}_{\mathsf{u}_i}?$ do not affect the control flow of P_i, i.e. this data is not used in the **if**-statements;

(II) the machines P_i may pass the bit-strings received from the user to the cryptographic library only in store-commands;

(III) the terms resulting from these store-commands will not become available to the adversary, i.e. the adversary does not get handles for these terms.

(IV) symmetric keys of order i only encrypt terms of order less than i (note that symmetric keys created by the adversary have no order and are thereby not restricted by this condition);

(V) if a symmetric key unknown to the adversary (i.e. the adversary does not have a handle to it) is used for encryption then this key will never become known to the adversary.

The analysis presented in this paper verifies that these five properties hold.

3 Analysis

3.1 Abstract Domain

The possible values of protocol variables are abstracted by sets of the following abstract values AV:

$$AV ::= AV_I \mid AV_H \mid \mathsf{seckey}(\ell, b) \qquad AV_I = \mathsf{X_P} \mid \mathsf{X_S}$$

$$\begin{array}{rclclclcl} AV_H & ::= & \mathsf{store}(AV_I) & \mid & \mathsf{nonce}(\ell, b) & \mid & \mathsf{symkey}(i, \ell, b) & \mid & \mathsf{symkeyname}(\ell, b) \\ & \mid & \mathsf{AnyPubVal} & \mid & \mathsf{pubkey}(\ell, b) & \mid & (AV_H, \ldots, AV_H) & \mid & \mathsf{pubenc}(AV_H, AV_H, \ell, b) \\ & & & & & & & \mid & \mathsf{symenc}(AV_H, AV_H, \ell, b) \end{array} \qquad (1)$$

Here AV_I contains the possible abstractions of payloads — they may be either public ($\mathsf{X_P}$) or secret ($\mathsf{X_S}$). The addresses of the communication partners (variable x_p in **send**- and **receive**-commands) are public. Data received from the protocol users are secret. The terms AV_H are the possible abstractions of terms in the database of $\mathcal{TH}_n$. They should be mostly self-descriptive. The arguments ℓ refer to program points (labels at expressions) where these values have been created. The arguments b also resemble program points — we have mentioned before that we analyse the parts of the protocol following a public-key decryption twice — once assuming that the ciphertext was generated by a protocol participant and once assuming that it was generated by the adversary. Hence, if n public-key decryptions occur before the program point ℓ then this point really counts as 2^n different program points for the analysis. If ℓ is a program point following n decryptions then b is a bit-string of length n where i-th bit records the assumed creator of the i-th decrypted ciphertext (1 — some honest participant; 0 — the adversary). We call b the *decryption context*.

The argument i in $\mathsf{symkey}(i, \ell, b)$ records the level of the symmetric key. The abstract value $\mathsf{symkeyname}(\ell, b)$ corresponds to the identities of the symmetric keys created at the program point ℓ (with the decryption context b). According to $\mathcal{TH}_n$, the adversary is able to find the identities of symmetric keys from the ciphertexts created with them. The abstract value AnyPubVal denotes any value that the adversary knows and may have constructed. All other AV_H denote values constructed by protocol participants. The secret decryption keys $\mathsf{seckey}(\ell, b)$ are not listed as a possible case for AV_H because $\mathcal{TH}_n$ puts severe restrictions on their use — they may only be used for decrypting ciphertexts; they cannot appear as subterms of more complex terms.

3.2 Constraint Variables

Given a protocol $\wp$ with its set of labels, we generate a constraint system with the following constraint variables:

- $\mathbf{S}_\ell^b$ for all statement labels ℓ occurring in the protocol (here we only consider the labels of **if**-, **let**- and **receive**-statements, not the labels occurring in expressions). Here b is a bit-string whose length equals the number asymmetric decryption operations that occur in the protocol before and including the point labeled with ℓ. Hence for a program point ℓ that is preceded by n asymmetric decryptions we have 2^n different variables $\mathbf{S}_\ell^b$.

  Let $\mathbf{Var}_\ell^\circ$ be the set of variables defined before the protocol point ℓ. Let $\mathbf{Var}_\ell^\bullet$ be the union of $\mathbf{Var}_\ell^\circ$ with the set of protocol variables that are assigned a value at ℓ (depending on whether ℓ labels an **if**-, **let**- or **receive**-statement, this set has 0, 1 or 2 elements). The possible values for $\mathbf{S}_\ell^b$ are mappings from $\mathbf{Var}_\ell^\bullet$ to sets of abstract values AV.

  The variable $\mathbf{S}_\ell^b$ records the possible values of protocol variables after a successful completion of the operation at program point ℓ. A **let**- or **if**-statement is successful if the default-/true-branch was taken. A **receive**-statement is always successful.

- $\mathbf{R}_\ell^b$ for all statement labels ℓ occurring in the protocol and b having the same possible values as for $\mathbf{S}_\ell^b$. These constraint variables are introduced to ease the presentation of the constraint system. Namely, the handling of a statement (if it succeeds) proceeds in two steps: first the constraints giving the abstraction(s) of the newly defined variable(s) are evaluated, followed by the evaluation of constraints describing the relationships between the values of different variables. The constraint variable $\mathbf{S}_\ell^b$ contains the result of these two steps. The constraint variable $\mathbf{R}_\ell^b$ contains the result of the first step only.

- $\mathbf{C}_c$ for all abstract channels $c \in \mathbf{Chan}_\mathsf{s} \cup \mathbf{Chan}_\mathsf{a}$ occurring in the protocol. The possible values of these variables are sets of abstract values AV_H. These variables will record an abstraction of the possible messages sent over the abstract channel c.

- $\mathbf{P}$. This will record the values that the adversary knows. The possible values of this variable are sets of abstract values AV.

- $\mathbf{E}_\ell^b$ for a label ℓ occurring at a key generation. This set records all abstract values that are encrypted with the key generated at ℓ for the preceding asymmetric decryption results described by b. The bit-string b has the same meaning as for the variables $\mathbf{S}_\ell^b$ (the point of interest is the occurrence of ℓ in the protocol). The possible values of these variables are sets of abstract values AV_H.

- $\mathbf{L}_{\ell,\mathsf{true}}^b$ and $\mathbf{L}_{\ell,\mathsf{false}}^b$ for labels ℓ at *let*- and *while*-statements. They denote whether the true-(default-) and false-branch of the statement are alive or not. The bit-string has the same meaning as before. The possible values of these variables are false and true.

3.3 Constraints

A constraint states that a constraint variable introduced above has to be greater than or equal to a monotone expression over constraint variables. For this to make sense, a partial order has to be defined on the possible values of constraint variables. For applying standard algorithms for constraint solving, the orders thus defined must have the structure of upper semilattice.

We order booleans by $\mathsf{false} \leq \mathsf{true}$. The sets of abstract values are ordered by subset inclusion. The mappings to sets of abstract values are ordered pointwise.

There are two sources for constraints — the protocol and the adversary. The first describes the movement of values during the computations performed by the protocol, while the second describes the capabilities of the adversary in decomposing messages. This second set of constraints is quite straightforward:

$$\mathsf{store}(AV) \in \mathbf{P} \Rightarrow AV \in \mathbf{P}$$

$$(AV_1, \ldots, AV_j) \in \mathbf{P} \Rightarrow AV_i \in \mathbf{P}$$

$$\mathsf{symenc}(AV_k, AV_t, \ell, b) \in \mathbf{P} \Rightarrow (\exists AV' \in \mathbf{P} : AV_k \cong_{\mathbf{P}} AV') \Rightarrow AV_t \in \mathbf{P}$$

$$\mathsf{pubenc}(\mathsf{AnyPubVal}, AV_t, \ell, b) \in \mathbf{P} \Rightarrow AV_t \in \mathbf{P}$$

$$\mathsf{pubenc}(AV_k, AV_t, \ell, b) \in \mathbf{P} \Rightarrow AV_k \in \mathbf{P}$$

$$\mathsf{symenc}(\mathsf{symkey}(i, \ell, b), AV_t, \ell', b') \in \mathbf{P} \Rightarrow \mathsf{symkeyname}(\ell, b) \in \mathbf{P}$$

$$\{\mathsf{X_P}, \mathsf{AnyPubVal}\} \subseteq \mathbf{P}$$

The first two constraints are obvious — the adversary can retrieve stored payloads and decompose lists. The third constraint states that the adversary can decrypt a symmetric encryption if it has the key. The relation $\cong_{\mathbf{P}}$ relates two abstract values if the sets of terms they correspond to may intersect. Because the meaning of AnyPubVal depends on the adversary's knowledge, this relation must also depend on it. The relation $\cong_{\mathbf{P}}$ is the least reflexive, symmetric and structure-respecting relation on abstract values that satisfies

$$AV \in \mathbf{P} \Rightarrow AV \cong_{\mathbf{P}} \mathsf{AnyPubVal}$$

$$\mathsf{store}(\mathsf{X_P}) \cong_{\mathbf{P}} \mathsf{AnyPubVal}$$

$$\left(\forall i : AV_i \cong_{\mathbf{P}} AV'_i\right) \wedge (AV'_1, \ldots, AV'_j) \cong_{\mathbf{P}} \mathsf{AnyPubVal} \Rightarrow (AV_1, \ldots, AV_j) \cong_{\mathbf{P}} \mathsf{AnyPubVal}$$

$$(\mathsf{AnyPubVal}, \ldots, \mathsf{AnyPubVal}) \cong_{\mathbf{P}} \mathsf{AnyPubVal}$$

$$AV_k \cong_{\mathbf{P}} AV'_k \wedge AV_t \cong_{\mathbf{P}} AV'_t \wedge \mathsf{pubenc}(AV'_k, AV'_t, \ell, b) \in \mathbf{P} \Rightarrow \mathsf{pubenc}(AV_k, AV_t, \ell, b) \cong_{\mathbf{P}} \mathsf{AnyPubVal}$$

$$AV_k \cong_{\mathbf{P}} AV'_k \wedge AV_t \cong_{\mathbf{P}} AV'_t \wedge \mathsf{symenc}(AV'_k, AV'_t, \ell, b) \in \mathbf{P} \Rightarrow \mathsf{symenc}(AV_k, AV_t, \ell, b) \cong_{\mathbf{P}} \mathsf{AnyPubVal}$$

$$\mathsf{X_S} \cong_{\mathbf{P}} \mathsf{X_P}$$

The fourth constraint for the adversary's capabilities states that if the public key used for public encryption may have been created by the adversary (which means that the secret key was also created by the adversary) then the adversary may find out the plaintext. The fifth and sixth constraints state that the adversary is capable of determining the identity of the key used to produce the ciphertext. For asymmetric encryption, this identity is the public key itself, while for symmetric encryption, it is the symkeyname. Finally, the public values $\mathsf{X_P}$ and AnyPubVal may be known to the adversary.

The set of constraints generated by an input or output process P is given by the mapping $\langle\!\langle\!\langle P \rangle\!\rangle\!\rangle$ that we are going to define below. For defining it, we also define the following mappings.

- $\langle\!\langle e\rangle\!\rangle(\mathbf{I}, b)$ gives the set of abstract values for the result of evaluating the expression e when the decryption context (after evaluating e) is b and the abstractions of already defined variables are given by the mapping $\mathbf{I}$.

- $\langle\!\langle e\rangle\!\rangle_{\mathrm{s}}(\mathbf{I})$ and $\langle\!\langle e\rangle\!\rangle_{\mathrm{f}}(\mathbf{I})$ give some necessary conditions for the evaluation of e to succeed or fail.

- $\langle\!\langle e\rangle\!\rangle_{\mathcal{E}}(\mathbf{I}, \mathbf{L})$ gives the set of constraints for the variables $\mathbf{E}^b_\ell$, as generated by e. Here $\mathbf{L}$ is a boolean showing whether this expression is live code.

- $\lfloor\!\lfloor x := e\rfloor\!\rfloor(b, \mathbf{X}, y)$, where $\mathbf{X}$ gives the abstractions of variables defined before the assignment $x := e$, and y is either x or a variable occurring in e gives a set of abstract values that certainly abstracts the value of y after the successful evaluation of e. The mapping $\lfloor\!\lfloor x := e\rfloor\!\rfloor$ is used to collect the relationships between values of variables.

The relationships between variables allow us to make the analysis more precise and when taking them into account, we need to form greatest lower bounds of pairs of sets of abstract values. While the least upper bound may be just the union of sets, the greatest lower bound cannot be simply the intersection. This reason for this is, that when two sets of abstract values $\mathbf{A}$ and $\mathbf{B}$ are both valid abstractions of some concrete value then we want their greatest lower bound $\mathbf{A} \dot{\cap} \mathbf{B}$ be a valid abstraction of that value as well. But certain concrete values may correspond to several different abstract values, for example a nonce that has become known to the adversary may occur in our abstractions either as $\mathsf{nonce}(\ell, b)$ for some ℓ and b or as $\mathsf{AnyPubVal}$.

For defining $\dot{\cap}$, we first define a *partial* binary operation $\sqcap$ on abstract values as the smallest (i.e. defined for as few arguments as possible) idempotent symmetric structure-preserving operation that satisfies $AV_H \sqcap \mathsf{AnyPubVal} = AV_H$ for any abstract value AV_H defined in (1). Now we can just define $\mathbf{A} \dot{\cap} \mathbf{B} = \{AV \sqcap AV' \mid AV \in \mathbf{A}, AV' \in \mathbf{B}\}$. We also define $\mathbf{A} \dot{\subseteq} \mathbf{B}$ iff $\mathbf{A} \dot{\cap} \mathbf{B} = \mathbf{A}$.

The mappings $\langle\!\langle e\rangle\!\rangle$, $\langle\!\langle e\rangle\!\rangle_{\mathrm{s}}$, $\langle\!\langle e\rangle\!\rangle_{\mathrm{f}}$ and $\langle\!\langle e\rangle\!\rangle_{\mathcal{E}}$ are the following. If we have left out the definition of $\langle\!\langle e\rangle\!\rangle_{\mathrm{s}}$ or $\langle\!\langle e\rangle\!\rangle_{\mathrm{f}}$ for some e then it is true. If we have left out the definition of $\langle\!\langle e\rangle\!\rangle_{\mathcal{E}}$ for some e then it is $\emptyset$. Let L_a [resp. L_s] be the set of all labels ℓ occurring in the protocol in the positions $\underline{\mathsf{keypair}}^\ell$ [resp. $\mathsf{gen\_symenc\_key}(i)^\ell$].

$$\langle\!\langle n\rangle\!\rangle(\mathbf{I}, b) = \{\mathsf{X_P}\}$$

$$\langle\!\langle n\rangle\!\rangle_{\mathrm{f}}(\mathbf{I}) = \mathsf{false}$$

$$\langle\!\langle x\rangle\!\rangle(\mathbf{I}, b) = \mathbf{I}(x)$$

$$\langle\!\langle f\rangle\!\rangle x(\mathbf{I}) = \mathsf{false}$$

$$\langle\!\langle \underline{\mathsf{keypair}}^\ell\rangle\!\rangle(\mathbf{I}, b) = \{\mathsf{seckey}(\ell, b)\}$$

$$\langle\!\langle \underline{\mathsf{keypair}}^\ell\rangle\!\rangle_{\mathrm{f}}(\mathbf{I}) = \mathsf{false}$$

$$\langle\!\langle \mathsf{store}(x)\rangle\!\rangle(\mathbf{I}, b) = \{\mathsf{store}(AV) \mid AV \in \mathbf{I}(x)\}$$

$$\langle\!\langle \mathsf{store}(x)\rangle\!\rangle_{\mathrm{f}}(\mathbf{I}) = \mathsf{false}$$

$$\langle\!\langle \mathsf{retrieve}(x)\rangle\!\rangle(\mathbf{I}, b) = \{AV \mid \mathsf{store}(AV) \in \mathbf{I}(x)\} \cup \{\mathsf{X_P} \mid \mathsf{AnyPubVal} \in \mathbf{I}(x)\}$$

$$\langle\!\langle \mathsf{retrieve}(x)\rangle\!\rangle_{\mathrm{s}}(\mathbf{I}) = \mathsf{AnyPubVal} \in \mathbf{I}(x) \vee \exists AV : \mathsf{store}(AV) \in \mathbf{I}(x)$$

$$\langle\!\langle \mathsf{retrieve}(x)\rangle\!\rangle_{\mathrm{f}}(\mathbf{I}) = \exists AV \in \mathbf{I}(x) : AV \neq \mathsf{store}(\ldots)$$

$$\langle\!\langle \mathsf{list}(x_1, \ldots, x_k)\rangle\!\rangle(\mathbf{I}, b) = \{(AV_1, \ldots, AV_k) \mid AV_i \in \mathbf{I}(x_i)\}$$

$$\langle\!\langle \mathsf{list}(x_1,\dots,x_k)\rangle\!\rangle_{\mathrm{s}}(\mathbf{I}) = \forall i : \exists AV \in \mathbf{I}(x_i) : AV \neq \mathsf{seckey}(\dots)$$

$$\langle\!\langle \mathsf{list}(x_1,\dots,x_k)\rangle\!\rangle_{\mathrm{f}}(\mathbf{I}) = \exists i : \exists AV \in \mathbf{I}(x_i) : AV = \mathsf{seckey}(\dots)$$

$$\langle\!\langle \mathsf{gen\_symenc\_key}(i)^{\ell}\rangle\!\rangle(\mathbf{I}, b) = \{\mathsf{symkey}(i,\ell,b)\}$$

$$\langle\!\langle \mathsf{gen\_symenc\_key}(i)^{\ell}\rangle\!\rangle_{\mathrm{f}}(\mathbf{I}) = \mathsf{false}$$

$$\langle\!\langle \underline{\pi_i^j}(x)\rangle\!\rangle(\mathbf{I}, b) = \{AV_i \mid (AV_1,\dots,AV_k) \in \mathbf{I}(x)\} \cup \{\mathsf{AnyPubVal} \mid \mathsf{AnyPubVal} \in \mathbf{I}(x)\}$$

$$\langle\!\langle \underline{\pi_i^j}(x)\rangle\!\rangle_{\mathrm{s}}(\mathbf{I}) = \mathsf{AnyPubVal} \in \mathbf{I}(x) \vee \exists (AV_i,\dots,AV_j) \in \mathbf{I}(x)$$

$$\langle\!\langle \underline{\pi_i^j}(x)\rangle\!\rangle_{\mathrm{f}}(\mathbf{I}) = \exists AV \in \mathbf{I}(x) : AV \neq (AV_1,\dots,AV_j)$$

$$\langle\!\langle \underline{\mathsf{pubkey}}(x)\rangle\!\rangle(\mathbf{I}, b) = \{\mathsf{pubkey}(\ell,b) \mid \mathsf{seckey}(\ell,b) \in \mathbf{I}(x)\}$$

$$\langle\!\langle \underline{\mathsf{pubkey}}(x)\rangle\!\rangle_{\mathrm{s}}(\mathbf{I}) = \exists \ell, b : \mathsf{seckey}(\ell,b) \in \mathbf{I}(x)$$

$$\langle\!\langle \underline{\mathsf{pubkey}}(x)\rangle\!\rangle_{\mathrm{f}}(\mathbf{I}) = \exists AV \in \mathbf{I}(x) : AV \neq \mathsf{seckey}(\dots)$$

$$\langle\!\langle \mathsf{gen\_nonce}\rangle\!\rangle(\mathbf{I}, b) = \{\mathsf{nonce}(\ell,b)\}$$

$$\langle\!\langle \mathsf{gen\_nonce}\rangle\!\rangle_{\mathrm{f}}(\mathbf{I}) = \mathsf{false}$$

$$\begin{aligned}\langle\!\langle \underline{\mathsf{pubenc}}^{\ell}(x_k,x_t)\rangle\!\rangle(\mathbf{I}, b) = {} & \\ & \{\mathsf{pubenc}(AV_k, AV_t, \ell, b) \mid AV_k \in \mathbf{I}(x_k), AV_t \in \mathbf{I}(x_t), AV_k = \mathsf{pubkey}(\dots)\} \cup \\ & \qquad \{\mathsf{pubenc}(\mathsf{AnyPubVal}, AV_t, \ell, b) \mid \mathsf{AnyPubVal} \in \mathbf{I}(x_k), AV_t \in \mathbf{I}(x_t)\}\end{aligned}$$

$$\langle\!\langle \underline{\mathsf{pubenc}}^{\ell}(x_k,x_t)\rangle\!\rangle_{\mathrm{s}}(\mathbf{I}) = \mathsf{AnyPubVal} \in \mathbf{I}(x_k) \vee \exists \ell', b' : \mathsf{pubkey}(\ell',b') \in \mathbf{I}(x_k)$$

$$\langle\!\langle \underline{\mathsf{pubenc}}^{\ell}(x_{\mathrm{k}},x_{\mathrm{t}})\rangle\!\rangle_{\mathcal{E}}(\mathbf{I}, \mathbf{L}) = \{\mathsf{pubkey}(\ell',b') \in \mathbf{I}(x_{\mathrm{k}}) \wedge \mathbf{L} \Rightarrow \mathbf{I}(x_{\mathrm{t}}) \subseteq \mathbf{E}_{\ell'}^{b'} \mid \ell' \in \mathsf{L}_a\}$$

$$\begin{aligned}\langle\!\langle \underline{\mathsf{privenc}}^{\ell}(x_k,x_t)\rangle\!\rangle(\mathbf{I}, b) = {} & \\ & \{\mathsf{symenc}(AV_k, AV_t, \ell, b) \mid AV_k \in \mathbf{I}(x_k), AV_t \in \mathbf{I}(x_t), AV_k = \mathsf{symkey}(\dots)\} \cup \\ & \qquad \{\mathsf{symenc}(\mathsf{AnyPubVal}, AV_t, \ell, b) \mid \mathsf{AnyPubVal} \in \mathbf{I}(x_k), AV_t \in \mathbf{I}(x_t)\}\end{aligned}$$

$$\langle\!\langle \underline{\mathsf{privenc}}^{\ell}(x_k,x_t)\rangle\!\rangle_{\mathrm{s}}(\mathbf{I}) = \mathsf{AnyPubVal} \in \mathbf{I}(x_k) \vee \exists i, \ell', b' : \mathsf{symkey}(i,\ell',b') \in \mathbf{I}(x_k)$$

$$\langle\!\langle \underline{\mathsf{privenc}}^{\ell}(x_{\mathrm{k}},x_{\mathrm{t}})\rangle\!\rangle_{\mathcal{E}}(\mathbf{I}, \mathbf{L}) = \{\mathsf{symkey}(i,\ell',b') \in \mathbf{I}(x_{\mathrm{k}}) \wedge \mathbf{L} \Rightarrow \mathbf{I}(x_{\mathrm{t}}) \subseteq \mathbf{E}_{\ell'}^{b'} \mid \ell' \in \mathsf{L}_s\}$$

$$\begin{aligned}\langle\!\langle \underline{\mathsf{privdec}}(x_k,x_t)\rangle\!\rangle(\mathbf{I}, b) = {} & \\ & \{AV_p \mid \mathsf{symenc}(AV_k, AV_p, \ell', b') \in \mathbf{I}(x_t), AV'_k \in \mathbf{I}(x_k), AV_k \cong_{\mathbf{P}} AV'_k\} \cup \\ & \qquad \{\mathsf{AnyPubVal} \mid \mathsf{AnyPubVal} \in \mathbf{I}(x_t), \mathbf{I}(x_k) \cap \mathbf{P} \neq \emptyset\} \cup \\ & \qquad\qquad \textit{if } \mathsf{AnyPubVal} \in \mathbf{I}(x_t) \textit{ then } \bigcup_{\mathsf{symkey}(i,\ell,b) \in \mathbf{I}(x_k)} \mathbf{E}_{\ell}^{b} \textit{ else } \emptyset\end{aligned}$$

$$\langle\!\langle \underline{\mathsf{privdec}}(x_k,x_t)\rangle\!\rangle_{\mathrm{s}}(\mathbf{I}) = \mathsf{AnyPubVal} \in \mathbf{I}(x_k) \vee \exists i, \ell', b' : \mathsf{seckey}(\ell',b') \in \mathbf{I}(x_k)$$

$$\begin{aligned}\langle\!\langle \underline{\mathsf{pubdec}}(x_k,x_t)\rangle\!\rangle(\mathbf{I}, b1) = {} & \\ & \{AV_p \mid \mathsf{pubenc}(\mathsf{pubkey}(\ell'',b''), AV_p, \ell', b') \in \mathbf{I}(x_t), \mathsf{seckey}(\ell'',b'') \in \mathbf{I}(x_k)\} \cup \\ & \qquad\qquad \textit{if } \mathsf{AnyPubVal} \in \mathbf{I}(x_t) \textit{ then } \bigcup_{\mathsf{seckey}(\ell,b) \in \mathbf{I}(x_k)} \mathbf{E}_{\ell}^{b} \textit{ else } \emptyset\end{aligned}$$

$$\langle\!\langle \underline{\mathsf{pubdec}}(x_k,x_t)\rangle\!\rangle(\mathbf{I}, b0) = \{\mathsf{AnyPubVal} \mid \mathsf{AnyPubVal} \in \mathbf{I}(x_t)\}$$

$$\langle\!\langle \underline{\mathsf{pubdec}}(x_k, x_t)\rangle\!\rangle_{\mathrm{s}}(\mathbf{I}) = \exists \ell', b' : \mathsf{seckey}(\ell', b') \in \mathbf{I}(x_k)$$

Here we can see the special treatment of AnyPubVal — for example, payloads can be extracted from it and projections can be taken. The result is still a public value. During encryption, AnyPubVal may serve as the encryption key (of course, such ciphertexts can be decrypted by the adversary). During decryption, when the ciphertext is AnyPubVal, we use the variables $\mathbf{E}^b_\ell$ to determine the possible plaintexts.

The distinction between participant-generated and adversary-generated ciphertexts in public-key decryption can be seen in two definitions for $\langle\!\langle \underline{\mathsf{pubdec}}(x_k, x_t)\rangle\!\rangle$. First of them assumes that the ciphertext is generated by some protocol participant, while the second assumes that the adversary is the source of the ciphertext. Both of these cases are also present in symmetric decryption, but they have been joined together, so that the analysis does not handle them separately.

The relationships between newly defined and existing variables are given by $\lfloor\!\lfloor x := e \rfloor\!\rfloor(b, \mathbf{X}, y)$. Recall that it gives for a variable y a set of abstract values that is guaranteed to abstract its concrete value. If the definition for some $\lfloor\!\lfloor x := e \rfloor\!\rfloor(b, \mathbf{X}, y)$ is missing below, it is equal to $\mathbf{X}(y)$ (i.e. no precision is gained).

$$\lfloor\!\lfloor x := \mathsf{list}(x_1, \ldots, x_k)\rfloor\!\rfloor(b, \mathbf{X}, x) = \{(AV_1, \ldots, AV_k) \mid AV_i \in \mathbf{X}(x_i)\}$$

$$\lfloor\!\lfloor x := \mathsf{list}(x_1, \ldots, x_k)\rfloor\!\rfloor(b, \mathbf{X}, x_i) = \{AV_i \mid (AV_1, \ldots, AV_k) \in \mathbf{X}(x)\} \cup \{\mathsf{AnyPubVal} \mid \mathsf{AnyPubVal} \in \mathbf{X}(x)\}$$

$$\lfloor\!\lfloor x := \underline{\pi^j_i}(y)\rfloor\!\rfloor(b, \mathbf{X}, x) = \{AV_i \mid (AV_1, \ldots, AV_k) \in \mathbf{X}(y)\} \cup \{\mathsf{AnyPubVal} \mid \mathsf{AnyPubVal} \in \mathbf{X}(y)\}$$

$$\lfloor\!\lfloor x := \underline{\pi^j_i}(y)\rfloor\!\rfloor(b, \mathbf{X}, y) = \{(AV'_1, \ldots, AV'_{i-1}, AV_i \sqcap AV'_i, AV'_{i+1}, \ldots, AV'_j) \mid AV_i \in \mathbf{X}(x), (AV'_1, \ldots, AV'_j) \in \mathbf{X}(y)\} \cup \{(\underbrace{\mathsf{AnyPubVal}, \ldots, \mathsf{AnyPubVal}}_{i-1}, AV_i, \underbrace{\mathsf{AnyPubVal}, \ldots, \mathsf{AnyPubVal}}_{j-i}) \mid AV_i \in \mathbf{X}(x), \mathsf{AnyPubVal} \in \mathbf{X}(y)\}$$

$$\lfloor\!\lfloor x := y\rfloor\!\rfloor(b, \mathbf{X}, x) = \mathbf{X}(y)$$

$$\lfloor\!\lfloor x := y\rfloor\!\rfloor(b, \mathbf{X}, y) = \mathbf{X}(x)$$

$$\lfloor\!\lfloor y := \underline{\mathsf{pubenc}}^\ell(x_k, x_t)\rfloor\!\rfloor(b, \mathbf{X}, y) = \{\mathsf{pubenc}(AV_k, AV_t, \ell, b) \mid AV_k \in \mathbf{X}(x_k), AV_t \in \mathbf{X}(x_t)\}$$

$$\lfloor\!\lfloor y := \underline{\mathsf{pubenc}}^\ell(x_k, x_t)\rfloor\!\rfloor(b, \mathbf{X}, x_k) = \{AV_k \mid \mathsf{pubenc}(AV_k, AV_t, \ell', b') \in \mathbf{X}(y), \{AV_t\} \dot{\subseteq} \mathbf{X}(x_t)\}$$

$$\lfloor\!\lfloor y := \underline{\mathsf{pubenc}}^\ell(x_k, x_t)\rfloor\!\rfloor(b, \mathbf{X}, x_t) = \{AV_t \mid \mathsf{pubenc}(AV_k, AV_t, \ell', b') \in \mathbf{X}(y), \{AV_k\} \dot{\subseteq} \mathbf{X}(x_k)\}$$

$$\lfloor\!\lfloor y := \underline{\mathsf{privenc}}^\ell(x_k, x_t)\rfloor\!\rfloor(b, \mathbf{X}, y) = \{\mathsf{symenc}(AV_k, AV_t, \ell, b \mid AV_k \in \mathbf{X}(x_k), AV_t \in \mathbf{X}(x_t)\}$$

$$\lfloor\!\lfloor y := \underline{\mathsf{privenc}}^\ell(x_k, x_t)\rfloor\!\rfloor(b, \mathbf{X}, x_k) = \{AV_k \mid \mathsf{symenc}(AV_k, AV_t, \ell', b') \in \mathbf{X}(y), \{AV_t\} \dot{\subseteq} \mathbf{X}(x_t)\}$$

$$\lfloor\!\lfloor y := \underline{\mathsf{privenc}}^\ell(x_k, x_t)\rfloor\!\rfloor(b, \mathbf{X}, x_t) = \{AV_t \mid \mathsf{symenc}(AV_k, AV_t, \ell', b') \in \mathbf{X}(y), \{AV_k\} \dot{\subseteq} \mathbf{X}(x_k)\}$$

$$\lfloor\!\lfloor y := \mathsf{store}(x)\rfloor\!\rfloor(b, \mathbf{X}, y) = \{\mathsf{store}(AV) \mid AV \in \mathbf{X}(x)\}$$

$$\lfloor\!\lfloor y := \underline{\mathsf{pubkey}}(x)\rfloor\!\rfloor(b, \mathbf{X}, y) = \{\mathsf{pubkey}(\ell, b') \mid \mathsf{seckey}(\ell, b') \in \mathbf{X}(x)\}$$

$$\lfloor\!\lfloor y := \underline{\mathsf{pubkey}}(x)\rfloor\!\rfloor(b, \mathbf{X}, x) = \{\mathsf{seckey}(\ell, b') \mid \mathsf{pubkey}(\ell, b') \in \mathbf{X}(x)\}$$

$$\lfloor\!\lfloor y := \underline{\mathsf{pubdec}}(x_k, x_t)\rfloor\!\rfloor(b1, \mathbf{X}, y) = \mathit{if}\ \mathsf{AnyPubVal} \in \mathbf{X}(x_k)\ \mathit{then}\ \mathbf{X}(y)\ \mathit{else} \bigcup_{\mathsf{seckey}(\ell', b') \in \mathbf{X}(x_k)} \mathbf{E}^{b'}_{\ell'}$$

$$\lfloor\!\lfloor y := \underline{\mathsf{privdec}}(x_k, x_t)\rfloor\!\rfloor(b, \mathbf{X}, y) = \mathit{if}\ \mathsf{AnyPubVal} \in \mathbf{X}(x_k)\ \mathit{then}\ \mathbf{X}(y)\ \mathit{else} \bigcup_{\mathsf{symkey}(i,\ell',b')\in\mathbf{X}(x_k)} \mathbf{E}_{\ell'}^{b'}$$

The usage of $\lfloor\!\lfloor x := e\rfloor\!\rfloor$ may become clearer when we look at the constraints generated by processes. This generation is done by $\langle\!\langle\!\langle P\rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C})$ where $\mathbf{I}$ is a constraint variable describing the protocol state before the execution of the process P, the bit-string b is the current decryption context, the variable $\mathbf{L}$ denotes whether this process is alive, and $\mathcal{C}$ is a set of constraints of the form $\mathbf{X}(x) \dot{\subseteq} E$ where E is a monotone expression that may contain references to $\mathbf{X}$. The constraints in $\mathcal{C}$ relate the abstract values of variables that have been defined before the execution of P. If $\mathcal{C}$ contains a constraint $\mathbf{X}(x) \subseteq E$ then the result of E is a suitable abstraction for the value of x.

Let $\mathbf{R}$ be a mapping from variables to sets of abstract values, and let $\mathcal{C}$ be a set of constraints in the form $\mathbf{X}(x) \dot{\subseteq} E$. We let $\mathfrak{L}(\mathbf{R}, \mathcal{C})$ denote the greatest solution to the constraints $\mathcal{C}$ (with $\mathbf{X}$ as the variable) that is less than or equal to $\mathbf{R}$.

The mapping $\langle\!\langle\!\langle P\rangle\!\rangle\!\rangle$ is defined as follows. If e is a not a public-key decryption then

$$\begin{aligned}\langle\!\langle\!\langle \mathbf{let}^\ell\ y := e\ \mathbf{in}\ P\ \mathbf{else}\ P'\rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C}) = \mathit{let}\ \mathcal{C}'\langle b\rangle = \mathcal{C} \cup \{\mathbf{X}(y) \dot{\subseteq} \lfloor\!\lfloor e\rfloor\!\rfloor(b, \mathbf{X}, y) \,|\, y \in \mathbf{Var}_\ell^\bullet\}\ \mathit{in}\\ \{\mathbf{L} \wedge \langle\!\langle e\rangle\!\rangle_{\mathrm{S}}(\mathbf{I}) \Rightarrow \mathbf{L}_{\ell,\mathsf{true}}^b,\ \mathbf{L} \wedge \langle\!\langle e\rangle\!\rangle_{\mathrm{F}}(\mathbf{I}) \Rightarrow \mathbf{L}_{\ell,\mathsf{false}}^b,\ \mathbf{L}_{\ell,\mathsf{true}}^b \Rightarrow \mathbf{R}_\ell^b \geq \mathbf{I}[y \mapsto \langle\!\langle e\rangle\!\rangle(\mathbf{I}, b)],\\ \mathbf{S}_\ell^b \geq \mathfrak{L}(\mathbf{R}_\ell^b, \mathcal{C}'\langle b\rangle)\} \cup \langle\!\langle e\rangle\!\rangle_{\mathcal{E}}(\mathbf{I}, \mathbf{L}_{\ell,\mathsf{true}}^b) \cup \langle\!\langle\!\langle P\rangle\!\rangle\!\rangle(\mathbf{S}_\ell^b, b, \mathbf{L}_{\ell,\mathsf{true}}^b, \mathcal{C}'\langle b\rangle) \cup \langle\!\langle\!\langle P'\rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}_{\ell,\mathsf{false}}^b, \mathcal{C})\ .\end{aligned}$$

If e is a public-key decryption then we have "two different default-branches", with decryption contexts $b1$ and $b0$:

$$\begin{aligned}\langle\!\langle\!\langle \mathbf{let}^\ell\ y := e\ \mathbf{in}\ P\ \mathbf{else}\ P'\rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C}) = \mathit{let}\ \mathcal{C}'\langle b\rangle = \mathcal{C} \cup \{\mathbf{X}(y) \dot{\subseteq} \lfloor\!\lfloor e\rfloor\!\rfloor(b, \mathbf{X}, y) \,|\, y \in \mathbf{Var}_\ell^\bullet\}\ \mathit{in}\\ \{\mathbf{L} \wedge \langle\!\langle e\rangle\!\rangle_{\mathrm{S}}(\mathbf{I}) \Rightarrow \mathbf{L}_{\ell,\mathsf{true}}^b,\ \mathbf{L} \wedge \langle\!\langle e\rangle\!\rangle_{\mathrm{F}}(\mathbf{I}) \Rightarrow \mathbf{L}_{\ell,\mathsf{false}}^b,\ \mathbf{L}_{\ell,\mathsf{true}}^b \Rightarrow \mathbf{R}_\ell^{b1} \geq \mathbf{I}[y \mapsto \langle\!\langle e\rangle\!\rangle(\mathbf{I}, b1)],\\ \mathbf{L}_{\ell,\mathsf{true}}^b \Rightarrow \mathbf{R}_\ell^{b0} \geq \mathbf{I}[y \mapsto \langle\!\langle e\rangle\!\rangle(\mathbf{I}, b0)],\ \mathbf{S}_\ell^{b1} \geq \mathfrak{L}(\mathbf{R}_\ell^{b1}, \mathcal{C}'\langle b1\rangle),\ \mathbf{S}_\ell^{b0} \geq \mathfrak{L}(\mathbf{R}_\ell^{b0}, \mathcal{C}'\langle b0\rangle)\} \cup \langle\!\langle e\rangle\!\rangle_{\mathcal{E}}(\mathbf{I}, \mathbf{L}_{\ell,\mathsf{true}}^b) \cup\\ \langle\!\langle\!\langle P\rangle\!\rangle\!\rangle(\mathbf{S}_\ell^{b1}, b1, \mathbf{L}_{\ell,\mathsf{true}}^b, \mathcal{C}'\langle b1\rangle) \cup \langle\!\langle\!\langle P\rangle\!\rangle\!\rangle(\mathbf{S}_\ell^{b0}, b0, \mathbf{L}_{\ell,\mathsf{true}}^b, \mathcal{C}'\langle b0\rangle) \cup \langle\!\langle\!\langle P'\rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}_{\ell,\mathsf{false}}^b, \mathcal{C})\ .\end{aligned}$$

We see that the following constraints are generated. If the process is alive ($\mathbf{L}$ is true) and the expression e may succeed [resp. fail] then we demand that the boolean variable reflecting that — $\mathbf{L}_{\ell,\mathsf{true}}^b$ [resp. $\mathbf{L}_{\ell,\mathsf{false}}^b$] is true, too. If e may succeed and hence $\mathbf{L}_{\ell,\mathsf{true}}^b$ is true then we let the mapping $\mathbf{R}_\ell^b$ be (at least) the mapping $\mathbf{I}$, but additionally we fix the abstraction of the left-hand side y. Here this "at least" means "equal to" because there will be no other constraints for $\mathbf{R}_\ell^b$. If $\mathbf{L}_{\ell,\mathsf{true}}^b$ is false then $\mathbf{R}_\ell^b$ has no constraints, hence it maps everything to $\emptyset$.

The set of constraints $\mathcal{C}'\langle b\rangle$ includes all the relationships between variables that are defined up to the successful execution of $y := e$. Note that the inequality signs in the constraints in $\mathcal{C}'\langle b\rangle$ has the opposite direction from the inequality signs in the constraints generated by $\langle\!\langle\!\langle P\rangle\!\rangle\!\rangle$. The constraint $\mathbf{S}_\ell^b \geq \mathfrak{L}(\mathbf{R}_\ell^b, \mathcal{C}'\langle b\rangle)$ states that $\mathbf{S}_\ell^b$ contains basically the same abstractions as $\mathbf{R}_\ell^b$, but all recorded relationships between variables have been taken into account. As this constraint is the only one for $\mathbf{S}_\ell^b$, the inequality $\mathbf{S}_\ell^b \leq \mathbf{R}_\ell^b$ always holds.

We also add the constraints for the variables $\mathbf{E}_{\ell'}^{b'}$ and we recursively invoke $\langle\!\langle\!\langle\cdot\rangle\!\rangle\!\rangle$ for the default- and the false-branch. The arguments for these recursive calls are also worth noting. We see that as we pass through the protocol, we collect the constraints expressing the relationships between variables. For the default-branch, $\mathbf{S}_\ell^b$ is the abstraction of the initial state, while for the false-branch, the same mapping $\mathbf{I}$ is used because no variable was assigned to if the evaluation of e failed. Also we collect no new relationships between variables if e fails (although it would be possible for some e, we have found that it does not change the precision of the analysis in practice).

Consider now other cases for the process P:

$$\begin{aligned}\langle\!\langle\!\langle \mathbf{if}^\ell\ x = x'\ \mathbf{then}\ P\ \mathbf{else}\ P' \rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C}) = \mathit{let}\ \mathcal{C}' = \mathcal{C} \cup \{\mathbf{X}(x) \dot{\subseteq} \mathbf{X}(x'), \mathbf{X}(x') \dot{\subseteq} \mathbf{X}(x)\}\ \mathit{in} \\ \{\mathbf{L} \Rightarrow \mathbf{L}^b_{\ell,\mathsf{false}},\ \mathbf{R}^b_\ell \geq \mathbf{I},\ \mathbf{L} \wedge (\exists AV \in \mathbf{I}(x)\, \exists AV' \in \mathbf{I}(x') : AV \cong_{\mathbf{P}} AV') \Rightarrow \mathbf{L}^b_{\ell,\mathsf{true}}, \\ \mathbf{S}^b_\ell \geq \mathfrak{L}(\mathbf{R}, \mathcal{C}')\} \cup \langle\!\langle\!\langle P \rangle\!\rangle\!\rangle(\mathbf{S}^b_\ell, b, \mathbf{L}^b_{\ell,\mathsf{true}}, \mathcal{C}') \cup \langle\!\langle\!\langle P' \rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}^b_{\ell,\mathsf{false}}, \mathcal{C})\end{aligned}$$

Here we check whether the abstractions of x and x' may intersect. We also add the equality of x and x' to the set of constraints $\mathcal{C}'$.

$$\begin{aligned}\langle\!\langle\!\langle \mathbf{send}_c[x_\mathrm{p}](x).I^* \rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C}) = \langle\!\langle\!\langle I^* \rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C}) \cup \{\mathbf{L} \Rightarrow \mathbf{I}(x_\mathrm{p}) \subseteq \mathbf{P}, \\ \mathbf{L} \wedge (c \in \mathbf{Chan}_\mathsf{s} \cup \mathbf{Chan}_\mathsf{a}) \Rightarrow \mathbf{I}(x) \subseteq \mathbf{C}_c,\ \mathbf{L} \wedge (c \in \mathbf{Chan}_\mathsf{a} \cup \mathbf{Chan}_\mathsf{i}) \Rightarrow \mathbf{I}(x) \subseteq \mathbf{P}\}\end{aligned}$$

A send-command always succeeds, the intended recipient becomes known to the adversary and the message is recorded as occurring on the channel c and/or becoming known to the adversary.

$$\begin{aligned}\langle\!\langle\!\langle \mathbf{receive}^\ell_c[x_\mathrm{p}](x).P \rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C}) = \{\mathbf{L} \Rightarrow \mathbf{L}^\ell_{b,\mathsf{true}},\ \mathbf{S}^b_\ell \geq \mathbf{R}^b_\ell\} \cup \langle\!\langle\!\langle P \rangle\!\rangle\!\rangle(\mathbf{S}^b_\ell, b, \mathbf{L}^b_{\ell,\mathsf{true}}, \mathcal{C}) \cup \\ \begin{cases} \{\mathbf{L} \Rightarrow \mathbf{R}^b_\ell \geq \mathbf{I}[x \mapsto \mathbf{C}_c, x_\mathrm{p} \mapsto \{\mathsf{X_P}\}]\}, & \text{if } c \in \mathbf{Chan}_\mathsf{s} \cup \mathbf{Chan}_\mathsf{a} \\ \{\mathbf{L} \Rightarrow \mathbf{R}^b_\ell \geq \mathbf{I}[x \mapsto \{\mathsf{AnyPubVal}\}, x_\mathrm{p} \mapsto \{\mathsf{X_P}\}]\}, & \text{if } c \in \mathbf{Chan}_\mathsf{i} \\ \{\mathbf{L} \Rightarrow \mathbf{R}^b_\ell \geq \mathbf{I}[x \mapsto \{\mathsf{X_S}\}, x_\mathrm{p} \mapsto \{\mathsf{X_P}\}]\}, & \text{if } c \in \mathbf{Chan}_\mathsf{u} \end{cases}\end{aligned}$$

We see that a message from the adversary is abstracted as $\mathsf{AnyPubVal}$ and a message from the user as a secret payload. The sender of the message is already known to the adversary, hence x_p is a public integer. No new constraints are added, hence the invocation of $\mathfrak{L}$ is not needed for $\mathbf{S}^b_\ell$.

$$\langle\!\langle\!\langle I_1 \mid \cdots \mid I_n \rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C}) = \bigcup_{i=1}^{n} \langle\!\langle\!\langle I_i \rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C}) \qquad \langle\!\langle\!\langle \mathcal{JJ} \rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C}) = \emptyset\ .$$

If I_i is the program for the machine P_i then the set of constraints for the protocol is the union of $\langle\!\langle\!\langle I_i \rangle\!\rangle\!\rangle(\{\}, \varepsilon, \mathsf{true}, \emptyset)$ over all i, where $\{\}$ is the mapping with empty domain.

Given a protocol $\wp$ and a label ℓ occurring in this protocol (labeling a subprocess P), let $\mathbf{I}^b_\ell$ denote the variable $\mathbf{S}^{b'}_{\ell'}$ (or the empty mapping) that occurs as the first argument in the call $\langle\!\langle\!\langle P^\ell \rangle\!\rangle\!\rangle(\mathbf{I}, b, \mathbf{L}, \mathcal{C})$, invoked during the construction of constraints for $\wp$. That is, $\mathbf{I}^b_\ell$ gives the abstract values of variables before entering the subprocess labeled with ℓ in the context b.

The following theorem states how the security of a protocol can be established using our analysis.

Theorem 1. *Let $\wp$ be a protocol and let Let $\mathbf{S}^b_\ell$, $\mathbf{E}^b_\ell$, $\mathbf{P}$, $\mathbf{C}_c$, $\mathbf{L}^b_{\ell,\mathsf{b}}$ be such that the constraints given above are fulfilled. If the following conditions hold then the composition of machines $\mathcal{TH}_n$ and P_i ($1 \leq i \leq n$) preserves the secrecy of payloads, i.e., the payloads are cryptographically secret if $\mathcal{TH}_n$ is replaced by its cryptographic realization.*

(I) If the protocol contains a statement of the form $\mathbf{if}^\ell\ x = x' \ldots$ then $\mathsf{X_S} \notin \mathbf{I}^b_\ell(x)$, $\mathsf{X_S} \notin \mathbf{I}^b_\ell(x')$, $\mathsf{store}(\mathsf{X_S}) \notin \mathbf{I}^b_\ell(x)$ and $\mathsf{store}(\mathsf{X_S}) \notin \mathbf{I}^b_\ell(x')$ for any b.

(II) If $\mathsf{X_S} \in \mathbf{S}^b_\ell(x)$ for some b, x, and this x occurs as an argument to some operation where the abstract values at entry are given by $\mathbf{S}^b_\ell$, then this operation is store or a send to a user.

(III) $\mathsf{X_S} \notin \mathbf{P}$.

(IV) If $\mathsf{AV} \in \mathbf{E}^b_\ell$ and a symm. key of order i is generated at ℓ then the order of AV is less than i.

- *The order of* $\mathsf{symkey}(i, \ell, b)$ *is* i. *The order of a tuple is the maximum order of its members. The order of other abstract values is* 0.

(V) $\mathsf{symkey}(i, \ell, b) \notin \mathbf{P}$ *for any* i, ℓ, b.

4 Implementation

We find the (componentwise) least solution for the aforementioned collection of inequalities. The least fixed point is computed iteratively, using a version of the solver from [23], which is specifically tailored to systems of constraints. The computation might not terminate but we believe that this is not a problem for real protocols; in fact, we have never encountered this situation when we applied our tool to common protocols of the literature. The only case in which computation is not guaranteed to terminate is if the protocol is able to create terms of arbitrary complexity all by itself, without the help from the adversary. We also believe that the potentially exponential number of variables in the size of the protocol is not a problem in practice because protocol descriptions are short.

There are ways to deal with the divergence of the fixed-point computation (add suitable widenings). Also, we believe that the exponential number of sets can be represented in a more compact way, if necessary.

The value of $\mathfrak{L}(\mathbf{R}, \mathcal{C})$ is also computed iteratively, using the same solution method. The mapping $\mathbf{X}$ is initialized with $\mathbf{R}$ and the iteration proceeds downwards.

The constraint generator and solver have been implemented in O'Caml (version 3.09 was used to compile it to native code). We have tested the analyser on several protocols from the literature, namely Needham-Schroeder public key [42], its fix by Lowe [38], Otway-Rees [43], Yahalom, and its modification by Burrows et. al [16]. The goal of all these protocols is to exchange a symmetric key between two parties. We use our analysis to find out whether it is safe to use the exchanged key to protect secret payloads in transit over public networks. Therefore we have added an extra message to the end of the protocol sessions in all of these protocols. This extra message is a communication of a secret payload from one party to another, encrypted under the freshly exchanged key. Our analysis considers all these protocols secure, except for the original (and indeed flawed) Needham-Schroeder protocol. If one allows old session keys to become known to the adversary then there may be attacks which are not discovered by our analyser because the BPW model does not consider the leakage of secret keys that have been used (this would cause a so-called commitment problem which makes a proof of computational soundness in the sense of BRSIM/UC impossible). Attacks against the modified Yahalom have been published [49] but these attacks did not affect message secrecy properties — they do not allow an adversary to learn the new key or inject its own key. The running times of the analyser on a computer with 1 GHz Intel Celeron processor and 256 MB of main memory are between one and eight seconds for these protocols with less than two seconds for Needham-Schroeder-Lowe and both versions of Yahalom.

5 Correctness of the Analysis

Theorem 1 is a straightforward corollary of a lemma similar to subject reduction. We are going to give the statement of that lemma here, its proof is given in the appendix.

When arguing about the correctness, we need to distinguish public and secret payloads. Hence we change the semantics of the system a little bit and assume that each payload is labeled with its secrecy level. An integer received from the protocol user is labeled as secret; a constant integer or the apparent sender of some message is labeled as public. When the payloads are stored in the database of terms then the labels are stored as well. They are also retrieved together with labels. When two integers are compared their secrecy levels are ignored.

Let $\mathfrak{R}$ be a run of the protocol (a finite sequence of steps). Let $\mathcal{O}$ be the state of the database of $\mathcal{TH}_n$ at the end of this run. Let x be a protocol variable whose definition occurs under k replications. If the variable x has been assigned a value inside the i_1-th replica of the outermost replication, i_2-th replica of the next replication, etc., then we denote this value $\mathcal{O}(x, [i_1, \ldots, i_k])$. This value is either a handle to a term in $\mathcal{O}$, a secret integer, or a public integer. Similarly, if a program point ℓ inside k replications and (syntactically) preceded by n public-key decryptions then let $\mathcal{O}(\ell, [i_1, \ldots, i_k])$ be the n-bit string whose bits describe the source of n ciphertexts whose decryptions are visible at the point ℓ in the replica $[i_1, \ldots, i_k]$. Note that a ciphertext is a term in the database $\mathcal{O}$ and its creator is simply the first principal that had a handle to it.

For a set $\mathcal{T}$ of terms in the database $\mathcal{O}$ we let its *downwards closure* $\overline{\mathcal{T}}$ be the smallest set of terms containing $\mathcal{T}$ and being closed with respect to list projection, decryption with keys in $\overline{\mathcal{T}}$, and extracting the public keys and symmetric key names from ciphertexts. For an abstract value AV we define its semantics $[\![AV]\!]_{\mathcal{O}}$ with respect to the contents of the database $\mathcal{O}$. The semantics is either a set of terms in $\mathcal{O}$ or a set of payloads.

- $[\![\mathsf{X_P}]\!]_{\mathcal{O}} = \{\text{“public } n\text{”} \mid n \in \mathbb{N}\}$.
- $[\![\mathsf{X_S}]\!]_{\mathcal{O}} = \{\text{“secret } n\text{”} \mid n \in \mathbb{N}\}$.
- $[\![\mathsf{store}(AV)]\!]_{\mathcal{O}}$ is the set of all terms of type data in $\mathcal{O}$ whose argument belongs to $[\![AV]\!]_{\mathcal{O}}$.
- $[\![\mathsf{nonce}(\ell, b)]\!]_{\mathcal{O}}$ is the set of all terms of type nonce in $\mathcal{O}$ that are generated by the gen_nonce-expressions at the protocol point ℓ at replicas $[i_1, \ldots, i_k]$, such that $b = \mathcal{O}(\ell, [i_1, \ldots, i_k])$
- $[\![\mathsf{symkey}(i, \ell, b)]\!]_{\mathcal{O}}$, $[\![\mathsf{symkeyname}(\ell, \mathsf{b})]\!]_{\mathcal{O}}$, $[\![\mathsf{seckey}(\ell, b)]\!]_{\mathcal{O}}$, and $[\![\mathsf{pubkey}(\ell, b)]\!]_{\mathcal{O}}$ — defined the same way as $[\![\mathsf{nonce}(\ell, b)]\!]_{\mathcal{O}}$ (only replace nonce with skse, pkse, ske, or pke).
- $[\![(AV_1, \ldots, AV_j)]\!]_{\mathcal{O}}$ is the set of all terms of type list in $\mathcal{O}$ whose length is j and whose i-th component term $(1 \le i \le j)$ belongs to $[\![AV_i]\!]_{\mathcal{O}}$.
- $[\![\mathsf{pubenc}(AV_k, AV_p, \ell, b)]\!]_{\mathcal{O}}$ is the set of all terms of type enc, such that
  - they have been created by the pubenc-expressions at the protocol point ℓ at replicas $[i_1, \ldots, i_k]$, such that $b = \mathcal{O}(\ell, [i_1, \ldots, i_k])$;
  - the term corresponding to the public key must belong to $[\![AV_k]\!]_{\mathcal{O}}$;
  - the term corresponding to the plaintext must belong to $[\![AV_p]\!]_{\mathcal{O}}$.
- $[\![\mathsf{symenc}(AV_k, AV_p, \ell, b)]\!]_{\mathcal{O}}$ is defined similarly, where enc is replaced with symenc.
- $[\![\mathsf{AnyPubVal}]\!]_{\mathcal{O}}$ is the downwards closure of the set of all terms that the adversary knows.

Let $\tilde{\mathbf{P}}$ be the largest set that $\tilde{\mathbf{P}} \subseteq \mathbf{P} \backslash \{\mathsf{AnyPubVal}\}$ and $(AV_1, \ldots, AV_j) \in \tilde{\mathbf{P}}$ implies $AV_i \in \tilde{\mathbf{P}}$ for $1 \le i \le j$. Informally, $\tilde{\mathbf{P}}$ is obtained from $\mathbf{P}$ by deleting AnyPubVal and also any abstract value that is a list, one of whose components (after flattening lists) is AnyPubVal. The set $\tilde{\mathbf{P}}$ is a better characterization than $\mathbf{P}$ for the set of terms created by honest participants and learned by the adversary.

Definition 1 (Adversarially Well-constructed Terms). A term T from the downwards closure of the set of all terms known to the adversary is *adversarially well-constructed* with respect to $\mathbf{P}$ if one of the following holds:

- $\exists AV \in \tilde{\mathbf{P}}$, such that $T \in [\![AV]\!]_{\mathcal{O}}$.
- All immediate subterms of T (the immediate subterm of a public key or a symmetric key name is the corresponding secret key) are known to the adversary and are also adversarially well-constructed. Also, if T is of type nonce, ske, enc, $\mathsf{garbage}$, skse, symenc then T must have been constructed by the adversary.

That is, a term is adversarially well-constructed if the adversary knows how to construct this term from the terms that the analysis has found him to know.

Lemma 2 (Subject reduction). *Let $\wp$ be a protocol and let Let $\mathbf{S}_\ell^b$, $\mathbf{E}_\ell^b$, $\mathbf{P}$, $\mathbf{C}_c$, $\mathbf{L}_{\ell,\mathsf{b}}^b$ be such that the constraints given in Sec. 3.3 are fulfilled. Let $\mathfrak{R}$ be a run of the protocol $\wp$. Let $\mathcal{O}$ be the state of the database of $\mathcal{TH}_n$ at the end of $\mathfrak{R}$. The following claims hold.*

$\boxed{P}$ *If a term T is known to adversary then it is adversarially well-constructed wrt. $\mathbf{P}$.*

$\boxed{X}$ *If $\mathcal{O}(x, [i_1, \ldots, i_k]) = T$ (here T may be both a term or an immediate value) and the replica $[i_1, \ldots, i_k]$ passes through the point ℓ with the operation at ℓ succeeding and the value of x being defined, then there exists $AV \in \mathbf{S}_\ell^{\mathcal{O}(\ell,[i_1,\ldots,i_k])}(x)$, such that $T \in [\![AV]\!]_{\mathcal{O}}$.*

$\boxed{C}$ *If a term T is communicated over an abstract channel $c \in \mathbf{Chan}_\mathsf{s} \cup \mathbf{Chan}_\mathsf{a}$ then there exists $AV \in \mathbf{C}_c$, such that $T \in [\![AV]\!]_{\mathcal{O}}$.*

$\boxed{E}$ *If T_k is the term representing an asymmetric or symmetric key generated at the program point ℓ in the replica $[i_1, \ldots, i_k]$) and T_p is a term that occurs as the plaintext in an encryption where T_k is the key, then at least one of the following holds:*

- *there exists $AV \in \mathbf{E}_\ell^{\mathcal{O}(\ell,[i_1,\ldots,i_k])}$, such that $T_p \in [\![AV]\!]_{\mathcal{O}}$.*
- *T_k and T_p are both known to the adversary and the term representing encryption of T_p with T_k is constructed by the adversary.*

$\boxed{L}$ *If ℓ is a branching point in the protocol (a* **let**- *or* **if**-*statement) and if the B-branch was taken at the replica $[i_1, \ldots, i_k]$ (here B is either true/default or false), then $\mathbf{L}_{\ell,B}^{\mathcal{O}(\ell,[i_1,\ldots,i_k])} = \mathsf{true}$.*

The lemma is proved by induction over the length of $\mathfrak{R}$.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[2] Martín Abadi. Secrecy by Typing in Security Protocols. *Journal of the ACM*, 46(5):749–786, September 1999.

[3] Martín Abadi and Bruno Blanchet. Secrecy types for asymmetric communication. *Theoretical Computer Science*, 298(3):387–415, 2003.

[4] Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, January 2005.

[5] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[6] Michael Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2004.

[7] Michael Backes and Birgit Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *IEEE Journal on Selected Areas in Communications*, 22(10):2075–2086, 2004.

[8] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004. Full version in IACR Cryptology ePrint Archive 2004/059, Feb. 2004, `http://eprint.iacr.org/`.

[9] Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on on Dependable and Secure Computing*, 2(2):109–123, 2005.

[10] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, `http://eprint.iacr.org/`.

[11] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2003. Extended version in IACR Cryptology ePrint Archive 2003/145, Jul. 2003, `http://eprint.iacr.org/`.

[12] David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 2004.

[13] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663. Springer, 2005.

[14] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, 2006. To appear.

[15] Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static Validation of Security Protocols. *Journal of Computer Security*, 13(3):347–390, 2005.

[16] Michael Burrows, Martín Abadi, and Roger M. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[17] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.

[18] Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334, 2004. `http://eprint.iacr.org/`.

[19] Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.

[20] Anupam Datta, Ante Derek, John Mitchell, Vitalij Shmatikov, and Matthieu Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2005.

[21] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[22] Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 34–39, 1983.

[23] Christian Fecht and Helmut Seidl. An Even Faster Solver for General Systems of Equations. In Radhia Cousot and David A. Schmidt, editors, *Static Analysis, Third International Symposium, SAS '96*, volume 1145 of *LNCS*, pages 189–204, Aachen, Germany, September 1996. Springer-Verlag.

[24] Andrew D. Gordon and Alan Jeffrey. Authenticity by Typing for Security Protocols. *Journal of Computer Security*, 11(4):451–520, 2003.

[25] Andrew D. Gordon and Alan Jeffrey. Typing correspondence assertions for communication protocols. *Theoretical Computer Science*, 300(1-3):379–409, 7 May 2003.

[26] Andrew D. Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3-4):435–483, 2004.

[27] J. D. Guttman, F. J. Thayer Fabrega, and L. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 186–195, 2001.

[28] Jonathan Herzog, Moses Liskov, and Silvio Micali. Plaintext awareness via key registration. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 548–564. Springer, 2003.

[29] Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.

[30] Richard Kemmerer, Catherine Meadows, and Jon Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[31] Peeter Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.

[32] Peeter Laud. Handling Encryption in Analyses for Secure Information Flow. In Pierpaolo Degano, editor, *Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003*, volume 2618 of *LNCS*, pages 159–173, Warsaw, Poland, April 2003. Springer-Verlag.

[33] Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.

[34] Peeter Laud. Secrecy types for a simulatable cryptographic library. In *Proc. 12th ACM Conference on Computer and Communications Security*, pages 26–35, 2005.

[35] Peeter Laud and Varmo Vene. A Type System for Computationally Secure Information Flow. In Maciej Liśkiewicz and Rüdiger Reischuk, editors, *15th International Symposium on Fundamentals of Computation Theory (FCT) 2005*, volume 3623 of *LNCS*, pages 365–377, Lübeck, Germany, August 2005. Springer-Verlag.

[36] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[37] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[38] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96*, volume 1055 of *LNCS*, pages 147–166, Passau, Germany, March 1996. Springer-Verlag.

[39] Michael Merritt. *Cryptographic Protocols.* PhD thesis, Georgia Institute of Technology, 1983.

[40] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[41] J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733, 1998.

[42] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[43] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[44] Lawrence Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[45] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000. Extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, `http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz`.

[46] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001. Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, `http://eprint.iacr.org/`.

[47] Steve Schneider. Security properties and CSP. In *Proc. 17th IEEE Symposium on Security & Privacy*, pages 174–187, 1996.

[48] Christoph Sprenger, Michael Backes, David Basin, Birgit Pfitzmann, and Michael Waidner. Cryptographically sound theorem proving. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, 2006.

[49] Paul F. Syverson. A Taxonomy of Replay Attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 187–191, Franconia, New Hampshire, June 1994. IEEE Computer Society.

[50] Reinhard Wilhelm, Shmuel Sagiv, and Thomas W. Reps. Shape Analysis. In David A. Watt, editor, *Compiler Construction, 9th International Conference*, volume 1781 of *LNCS*, pages 1–17, Berlin, Germany, March 2000. Springer-Verlag.

A Proof of the Subject Reduction

A.1 Possible Steps of the Protocol Implementation

In the main body of the paper we mentioned that the proof of Lemma 2 is by induction over the length of the run $\mathfrak{R}$. Let us give a more precise account here on what the possible steps of the structure consisting of $\mathcal{TH}_n$ and $\mathsf{P}_1, \ldots, \mathsf{P}_n$ are. As we mentioned before, the internal state of each P_i consists of a sequence of pairs (P, E) where P is a process and E is its *environment*, giving values to all free variables of P. If the same variable occurs in the domains of environments of different processes, then it must be given the same value by all these environments. The values of variables are either integers or term handles (also represented by integers). We assume that the variables are typed, i.e. it is statically known whether the contents of a variable is a payload or a handle. The typing can be enforced by a trivial type system containing just two types.

At most one of the participants may be *active* at any time. If there exists an active participant P_i, its state additionally contains

- the index j of the currently running process in its sequence of processes,
- the running process P together with its environment E,
- the received message m together with its channel c and apparent sender u.

Finally, the state of the combination of $\mathcal{TH}_n$ and $\mathsf{P}_1, \ldots, \mathsf{P}_n$ also contains the database $\mathcal{O}$ of the machine $\mathcal{TH}_n$, and the contents of buffers of secure and authentic channels between participants.

A combination of $\mathcal{TH}_n$ and $\mathsf{P}_1, \ldots, \mathsf{P}_n$ with no active P_i can perform the following kinds of steps:

1. The adversary may invoke a local adversary command. This command is executed by $\mathcal{TH}_n$, possibly changing $\mathcal{O}$, and the result is returned to the adversary. No participant is activated.

2. The adversary may invoke a send-command. All messages (in terms of $\mathcal{TH}_n$) are pairs consisting of the abstract channel name c and the "real" message m. The recipient P_i of that message is activated, its index of the currently running process is initialized with 0, the running process P and its environment are not yet initialized, the received message m, abstract channel c, and the apparent sender u are initialized from the received message. The message is a handle; the apparent sender is a public integer. This step will remove a message from a buffer of a channel between participants, if that channel was a secure or an authentic one.

3. A user may send a message to the corresponding participant. This participant will be activated in the same way as when receiving a message from the network. The message m will be a secret integer, the apparent sender u will be 0 (public integer), the abstract channel c will be "the channel from/to the user".

A combination of $\mathcal{TH}_n$ and $\mathsf{P}_1, \ldots, \mathsf{P}_n$ with an active P_i can perform the following kinds of steps. Here everything is initiated by P_i.

4. If there is no running (P, E) then increment the index j of the currently running process. If that index becomes larger than the length of the sequence of processes of P_i, then become inactive. Otherwise consider the abstract channel mentioned in the receive-statement at the beginning of the process pointed to by j. If it is equal to c then let the running process P and its environment E be the j-th element of the sequence of processes (without the receive-command in the beginning of that process). In the environment E, we additionally map the variables in the receive-statement to m and u. If the abstract channel mentioned in the receive-statement is not equal to c then do nothing (i.e. start the next step).

In the following we assume that the running process with its environment (P, E) is defined. The next step depends on the first command of P.

5. If P is a parallel composition of input processes then put these processes, each of them together with the environment E, back to the sequence of processes of P_i. Put them just before the position j. If the process that was at the j-th position was not replicated, then remove it from that sequence of processes. Then become inactive.

6. If P is send-command then let P' be the rest of the process P. The process P' must be a parallel composition of input processes. First construct the outgoing message by pairing the abstract channel and the "real" message mentioned in the send-command. Then handle P' (as described above). But right before becoming inactive send the outgoing message away; the receiver was mentioned in the send-command. This step also adds to the buffer of some channel between participants, if that channel was a secure or an authentic one.

7. If P is $\mathcal{II}$ then undefine P and E.

8. If P is a let-command then evaluate the expression, possibly interacting with $\mathcal{TH}_n$ to evaluate the expression. If successful, update the environment E. Take the default- or false-branch, depending on the success of evaluating the expression. If the constant expression was evaluated then the result is a public integer.

9. If P is an if-command then compare and take either the true- or false-branch.

Initially, the processes in the state of each P_i are given by the protocol specification. The environments are empty. The database $\mathcal{O}$ is empty. The buffers of the channels are also empty. No participant is active.

A.2 Relationships between variables

We will show that if some constraint variable $\mathbf{R}_\ell^b$ is a valid abstraction of the values of protocol variables after the protocol point ℓ in context b then $\mathbf{S}_\ell^b = \mathcal{L}(\mathbf{R}_\ell^b, \mathcal{C})$ is also a valid abstraction, where $\mathcal{C}$ is the set of constraints describing the relations between variables collected thus far.

For the given protocol $\wp$, a label ℓ and a possible context b after a successful evaluation of the command at the protocol point ℓ, let $\mathcal{C}_\ell^b$ be the set of constraints used to compute $\mathbf{S}_\ell^b$ from $\mathbf{R}_\ell^b$.

Lemma 3. *Let T, AV_1 and AV_2 be such that $T \in [\![AV_1]\!]_\mathcal{O} \cap [\![AV_2]\!]_\mathcal{O}$. Then there exists AV, such that $\{AV_1\} \dot{\cap} \{AV_2\} = \{AV\}$ and $T \in [\![AV]\!]_\mathcal{O}$.*

Proof. Induction over the shape of T. If T has no subterms then either $AV_1 = AV_2$ or one of these two abstract values is equal to $\mathsf{AnyPubVal}$. In the first case $AV = AV_1 = AV_2$. In the second case AV is also equal to one of AV_1 or AV_2.

If T has subterms, but at least one of AV_1 or AV_2 is equal to $\mathsf{AnyPubVal}$ then AV is equal to the other abstract value among AV_1 and AV_2 and we are done. Otherwise the types of AV_1 and AV_2 are equal and they also correspond to the type of T. Let T' be an immediate subterm of T, let AV_1' and AV_2' be the corresponding immediate subvalues of AV_1 and AV_2. We find AV' so, that $\{AV'\} = \{AV_1'\} \dot{\cap} \{AV_2'\}$ and $T' \in [\![AV']\!]_\mathcal{O}$. Using these values AV' we construct the abstract value AV with necessary properties. $\square$

Lemma 4. *Let the run $\mathfrak{R}$ be such that its last step was a successful evaluation of a* **let**- *or a* **if**-*statement at the label ℓ in the replica $[i_1, \dots, i_k]$. Let $b = \mathcal{O}(\ell, [i_1, \dots, i_k])$. Assume that Lemma 2 holds for the run $\mathfrak{R}$ without the last step. Let $\mathbf{X}$ be a mapping from $\mathbf{Var}_\ell^\bullet$ to sets of abstract values, such that for all $x \in \mathbf{Var}_\ell^\bullet$ there exists some $AV \in \mathbf{X}(x)$ such that $\mathcal{O}(x, [i_1, \dots, i_k]) \in [\![AV]\!]_\mathcal{O}$. Let $x \in \mathbf{Var}_\ell^\bullet$ and let $\mathbf{X}(x) \dot{\subseteq} E$ be a constraint in $\mathcal{C}_\ell^b$. Then there exists some $AV \in E(\mathbf{X})$, such that $\mathcal{O}(x, [i_1, \dots, i_k]) \in [\![AV]\!]_\mathcal{O}$.*

Proof. Induction over the length of the path from the beginning of the protocol to ℓ. By the induction assumption, the claim of the lemma holds if $\mathbf{X}(x) \dot{\subseteq} E$ is a constraint that has been introduced at some ℓ' preceding ℓ — $\mathbf{X}$ is also a suitable mapping at ℓ', none of the variables mentioned in $\mathbf{X}(x) \dot{\subseteq} E$ have changed their values since ℓ', and the lemma holds at ℓ'.

Let $\mathbf{X}(x) \dot{\subseteq} E$ be introduced to the set of constraints at the protocol point ℓ. If E is $\mathbf{X}(x)$ then the claim of the lemma obviously holds. Otherwise ℓ has to label either an **if**- or a **let**-statement.

If ℓ labels a statement **if** $x = x' \dots$ then E must be $\mathbf{X}(x')$ (all other cases have been handled above). The premises of the lemma state that this **if**-statement is successful, i.e. returns true, hence $\mathcal{O}(x, [i_1, \dots, i_k]) = \mathcal{O}(x', [i_1, \dots, i_k])$. Also $\mathcal{O}(x', [i_1, \dots, i_k]) \in [\![AV]\!]_\mathcal{O}$ for some $AV \in \mathbf{X}(x')$ by the premises of the lemma.

If ℓ labels a statement *let* $y := e$ then the case analysis over all possible e convinces us that $E(\mathbf{X})$ must contain some AV, such that $\mathcal{O}(x, [i_1, \ldots, i_k]) \in [\![AV]\!]_{\mathcal{O}}$. $\square$

Lemma 5. *Let the run $\mathfrak{R}$ be such that its last step was a successful evaluation of a* **let**- *or a* **if**-*statement at the label ℓ in the replica $[i_1, \ldots, i_k]$. Let $b = \mathcal{O}(\ell, [i_1, \ldots, i_k])$. Assume that Lemma 2 holds for the run $\mathfrak{R}$ without the last step. If for all variables $x \in \mathbf{Var}_\ell^\bullet$ there exists $AV \in \mathbf{R}_\ell^b(x)$, such that $\mathcal{O}(x, [i_1, \ldots, i_k]) \in [\![AV]\!]_{\mathcal{O}}$, then for each variable x there also exists $AV \in \mathbf{S}_\ell^b(x)$, such that $\mathcal{O}(x, [i_1, \ldots, i_k]) \in [\![AV]\!]_{\mathcal{O}}$.*

Proof. Define the mappings f_i, g_i, where $i \in \mathbb{N}$, from $\mathbf{Var}_\ell^b$ to sets of abstract values by

$$f_i(x) = \dot{\bigcap_{\mathbf{X}(x) \dot{\subseteq} E \in \mathcal{C}_\ell^b}} E(g_{i-1}) \qquad \begin{aligned} g_0(x) &= \mathbf{R}_\ell^b(x) \\ g_i(x) &= f_i(x) \mathbin{\dot\cap} g_{i-1}(x) \end{aligned}$$

By the fix-point theorems of Tarski and Kleene we know that $\mathbf{S}_\ell^b(x) = \dot{\bigcap}_i g_i(x)$. As the sets $\mathbf{R}_\ell^b(x)$ are finite, there exists some i that $\mathbf{S}_\ell^b = g_i$.

By induction on i we can show that for each x and for each i there exists some $AV \in f_i(x)$ and $AV' \in g_i(x)$, such that $\mathcal{O}(x, [i_1, \ldots, i_k]) \in [\![AV]\!]_{\mathcal{O}}$ and $\mathcal{O}(x, [i_1, \ldots, i_k]) \in [\![AV']\!]_{\mathcal{O}}$. Indeed, the induction base ($i = 0$) is given by the premise of the lemma. The induction step from g_{i-1} to f_i follows from lemma 4 for each right hand side E of a constraint $\mathbf{X}(x) \dot{\subseteq} E$ in $\mathcal{C}_\ell^b$ and by lemma 3 for their intersection. The step from f_i to g_i follows from lemma 3. $\square$

A.3 Induction over the length of $\mathfrak{R}$

Induction base: the length of the run is 0. Then there are no terms, no variables with assigned values, and no exchanged messages. All claims of lemma are thus trivially true. Induction step: assume that the lemma holds for the run without the last step. No consider all possible last steps and show that the claims of the lemma still continue to hold.

The last step is of the 1st kind. The adversary may either create a new term or parse an existing term. The claims $\boxed{\text{X}}$, $\boxed{\text{C}}$ and $\boxed{\text{L}}$ clearly hold for the whole run, because this step does not change anything mentioned in these claims. The command given to $\mathcal{TH}_n$ may have one of the following effects:

- A new term is created, that is not an encryption. Then $\boxed{\text{E}}$ also holds for the whole run. The immediate subterms of this term are all adversarially well-constructed by $\boxed{\text{P}}$ of the induction assumption, hence $\boxed{\text{P}}$ holds, too.

- A new term representing public- or symmetric-key encryption is created. Then $\boxed{\text{P}}$ holds for the same reasons as in the previous case. Also, the key and the plaintext are known to the adversary, hence the second case of $\boxed{\text{E}}$ holds for it.

- The command was for creating a term, but it was not created because the respective term already existed. This term must be a payload or a list. It is still adversarially well-constructed because it does not have to be created by the adversary in this case. Hence $\boxed{\text{P}}$ holds for the whole run. $\boxed{\text{E}}$ also holds, because no new encryptions were created.

- The command was for parsing a term T, giving its component T'. Then $\boxed{\text{E}}$ holds for the whole run. If all subterms of T are adversarially well-constructed then T is adversarially well-constructed and $\boxed{\text{P}}$ holds for the whole run. Otherwise there exists some $AV \in \tilde{\mathbf{P}}$ such that $T \in [\![AV]\!]_{\mathcal{O}}$. By the definition of $[\![\cdot]\!]_{\mathcal{O}}$, there must exist some subterm AV' of AV such that $T' \in [\![AV']\!]_{\mathcal{O}}$. Indeed, $\mathsf{AnyPubVal}$ is the only abstract value for which $[\![\cdot]\!]_{\mathcal{O}}$ is not defined via structural induction, and $AV \neq \mathsf{AnyPubVal}$. By the constraints representing the power of the adversary, $AV' \in \mathbf{P}$ (this may need more elaboration). If AV' is $\mathsf{AnyPubVal}$ then T' was already known to the adversary, hence it is adversarially well-constructed by $\boxed{\text{P}}$ of the induction assumption. Otherwise T' is satisfies the first bullet point of the definition of adversarially well-constructedness. Thus $\boxed{\text{P}}$ is satisfied for the whole run.

The last step is of the 2nd kind. No effect on any quantity mentioned in the lemma.

The last step is of the 3rd kind. No effect on any quantity mentioned in the lemma.

The last step is of the 4th kind. This step affects only the values of variables, hence $\boxed{\text{P}}$, $\boxed{\text{E}}$, $\boxed{\text{C}}$ and $\boxed{\text{L}}$ hold for the whole run because they hold for the run without the last step. According to the semantics of the processes, the channel c is the source of the message m. If c is a secure or authentic channel then there exists $AV \in \mathbf{C}_c$, such that $m \in [\![AV]\!]_{\mathcal{O}}$. By the constraints for receive-commands, AV also belongs to $\mathbf{R}^b_\ell(x)$ (for any b), where x is the variable that is going to hold the message m. If c is an insecure channel then m is known to the adversary, implying $m \in [\![\mathsf{AnyPubVal}]\!]_{\mathcal{O}}$, and $\mathsf{AnyPubVal} \in \mathbf{R}^b_\ell(x)$. If c denotes the channel from the user of the protocol then m is a secret integer which belongs to the meaning of $\mathsf{X_S}$ which belongs to $\mathbf{R}^b_\ell(x)$. The variable x_{P} holding the apparent sender will get a value that is a public integer, and $\mathsf{X_P} \in \mathbf{R}^b_\ell(x_{\mathrm{P}})$.

The last step is of the 5th kind. No effect on any quantity mentioned in the lemma.

The last step is of the 6th kind. This step may add new terms to the channels, and to the adversary's knowledge. There is no effect on any quantity mentioned in $\boxed{\text{X}}$, $\boxed{\text{E}}$, or $\boxed{\text{L}}$. The constraints for the send-command should make it obvious that $\boxed{\text{C}}$ and $\boxed{\text{P}}$ also hold for the whole run if they held before the last step.

The last step is of the 7th kind. No effect on any quantity mentioned in the lemma.

The last step is of the 8th kind. Let $\mathbf{let}^\ell\ y := e\ \mathbf{in}\ P\ \mathbf{else}\ P'$ be the process where the step is made, let it be inside the replica $[i_1, \ldots, i_k]$. Let $b = \mathcal{O}(\ell, [i_1, \ldots, i_k])$. This step does not affect quantities mentioned in $\boxed{\text{P}}$ or $\boxed{\text{C}}$. We are evaluating an expression e and assigning the result to the variable x at the program point ℓ. Depending on the success of evaluating e, either the default or the false-branch is taken. By the induction assumption, $\mathbf{L}_0 = \mathsf{true}$ where $\mathbf{L}_0$ is the variable controlling whether the program point ℓ is reached (the third argument of $\langle\!\langle\!\langle P \rangle\!\rangle\!\rangle$). From the definition of $\langle\!\langle\!\langle \cdot \rangle\!\rangle\!\rangle$ we see that $\mathbf{L}^b_{\ell,\mathsf{true}}$ [resp. $\mathbf{L}^b_{\ell,\mathsf{false}}$] is true iff $\langle\!\langle e \rangle\!\rangle_{\mathrm{s}}(\mathbf{I}^b_\ell)$ [resp. $\langle\!\langle e \rangle\!\rangle_{\mathrm{f}}(\mathbf{I}^b_\ell)$] is true. By induction assumption $\boxed{\text{X}}$, for each variable x defined before that execution step in that replica, $\mathbf{I}^b_\ell(x)$ contains some AV, such that $\mathcal{O}(x, [i_1, \ldots, i_k]) \in [\![AV]\!]_{\mathcal{O}}$. When we consider all possible cases for e and the definitions of $\langle\!\langle e \rangle\!\rangle_{\mathrm{s}}$ and $\langle\!\langle e \rangle\!\rangle_{\mathrm{f}}$ then we see that if e succeeds [resp. fails] then $\langle\!\langle e \rangle\!\rangle_{\mathrm{s}}(\mathbf{I}^b_\ell)$ [resp. $\langle\!\langle e \rangle\!\rangle_{\mathrm{f}}(\mathbf{I}^b_\ell)$] must be true. Hence $\boxed{\text{L}}$ holds after the last step of $\mathfrak{R}$.

The set of encrypted terms under some key can only be affected if e is $\underline{\mathsf{pubenc}}$ or $\underline{\mathsf{privenc}}$. Again, the induction assumption $\boxed{\text{X}}$ implies that if $\boxed{\text{E}}$ held before the last step of $\mathfrak{R}$ then it continues to hold when this last step is made.

From the definition of $\langle\!\langle\!\langle \cdot \rangle\!\rangle\!\rangle$ we see that if $\mathbf{L}_0$ is true then we have the constraint $\mathbf{R}^b_\ell \geq \mathbf{I}^b_\ell[y \mapsto \langle\!\langle e \rangle\!\rangle(\mathbf{I}, b)]$. Considering all possible cases for e and making use of the induction assumption $\boxed{\text{X}}$ for $\mathbf{I}^b_\ell$ (and if e is decryption, then also $\boxed{\text{E}}$) we see that the result of e is a value that belongs to $[\![AV]\!]_{\mathcal{O}}$ for some $AV \in \langle\!\langle e \rangle\!\rangle(\mathbf{I}, b) = \mathbf{R}^b_\ell(y)$. Such an $AV \in \mathbf{R}^b_\ell(x)$, that the value of x is a member of $[\![AV]\!]_{\mathcal{O}}$, also exists for all other variables x in $\mathbf{Var}_\ell^\bullet$. Hence the premises of lemma 5 are fulfilled and the same claim holds if we replace $\mathbf{R}_\ell^b$ by $\mathbf{S}_\ell^b$. Hence $\boxed{\mathrm{X}}$ holds after the last step of $\mathfrak{R}$.

The last step is of the 9th kind. This step affects only the control flow of the protocol, hence $\boxed{\mathrm{P}}$, $\boxed{\mathrm{E}}$, and $\boxed{\mathrm{C}}$ hold for the whole run because they hold for the run without the last step. The claim $\boxed{\mathrm{X}}$ directly follows from Lemma 5. If the compared variables x and x' are integers then $\mathsf{X_S}$ or $\mathsf{X_P}$ is a member of $\mathbf{I}_\ell^b(x)$ (for the correct b) by $\boxed{\mathrm{X}}$ in the induction assumption, and $\mathsf{X_S}$ or $\mathsf{X_P}$ is also a member of $\mathbf{I}_\ell^b(x')$. The comparison of these elements of $\mathbf{I}_\ell^b(x)$ and $\mathbf{I}_\ell^b(x')$ by $\cong_\mathbf{P}$ returns true. Hence $\mathbf{L}_{\ell,\mathsf{true}}^b = \mathsf{true}$ and $\boxed{\mathrm{L}}$ holds for the whole run. If the compared variables x and x' are equal term handles, both pointing to the term T, then there must exist $AV \in \mathbf{I}_\ell^b(x)$ and $AV' \in \mathbf{I}_\ell^b(x')$, such that $T \in [\![AV]\!]_\mathcal{O} \cap [\![AV']\!]_\mathcal{O}$. Then lemma 6 states that $AV \cong_\mathbf{P} AV'$; we apply that lemma for the run without the last step. For the run without the last step the claims of the lemma hold by induction assumption. Hence $\mathbf{L}_{\ell,\mathsf{true}}^b = \mathsf{true}$ and $\boxed{\mathrm{L}}$ holds for the whole run. □

Lemma 6. *Assume that the lemma 2 holds for a certain protocol run. Let $\mathcal{O}$ be the database of $\mathcal{TH}_n$ at the end of that run. Let T be a term in the database, and AV, AV' be two abstract values, such that $T \in [\![AV]\!]_\mathcal{O} \cap [\![AV']\!]_\mathcal{O}$. Then $AV \cong_\mathbf{P} AV'$.*

Proof. Induction over the structure of T. Base: If T has no subterms then there are at most two abstract values whose semantics may contain T and one of them is $\mathsf{AnyPubVal}$. The second one has the constructor determined by the type of T and arguments (which may be only ℓ and b (and i)) determined by the location where T was created. Hence, if neither AV nor AV' is $\mathsf{AnyPubVal}$ then they must be equal abstract values. If one of them, say AV is $\mathsf{AnyPubVal}$ then T is known to the adversary, hence by $\boxed{\mathrm{P}}$ it must be adversarially well-constructed. If AV' is also equal to $\mathsf{AnyPubVal}$ then we are done. Otherwise consider the two possibilities in the definition of adversarially well-constructedness. If there exist some abstract value in $\tilde{\mathbf{P}}$ whose semantics contains T, then this abstract value must equal AV'. If there exist no such abstract value in $\tilde{\mathbf{P}}$ then T must be constructed by the adversary in which case there is no abstract value, other than $\mathsf{AnyPubVal}$ whose semantics contains T.

Step: T has subterms. Its type must thus be one of data, list, pke, pkse, enc, or symenc. If the type is pke or pkse then there also exists at most one abstract value besides $\mathsf{AnyPubVal}$ whose semantics contains T, and the same argument as for the induction basis applies. Assume now that the type of T is one of data, list, enc, or symenc.

If neither AV nor AV' is $\mathsf{AnyPubVal}$ then they must have the same outermost constructor. Indeed, different constructors of abstract values correspond to different types of terms in the database $\mathcal{O}$. Hence $AV \cong_\mathbf{P} AV'$ iff the corresponding subvalues of AV and AV' are related by $\cong_\mathbf{P}$. But the semantices of these corresponding subvalues also intersect, their intersection contains at least the corresponding subterm of T. By the induction assumption, the corresponding subterms of AV and AV' are related by $\cong_\mathbf{P}$.

If $AV = \mathsf{AnyPubVal}$ and $AV' \neq \mathsf{AnyPubVal}$ then T is known to the adversary, hence it is adversarially well-constructed. Consider the following possibilities.

- The type of T is enc or symenc and T was constructed by one of protocol participants. The outermost constructor of AV' must then be pubenc or symenc. The adversarially well-constructedness of T must be caused by the existence of some $AV'' \in \tilde{\mathbf{P}}$, such that $T \in [\![AV'']\!]_\mathcal{O}$. The term AV'' has the same outermost constructor as AV'. The definition of $\cong_\mathbf{P}$ gives us $AV'' \cong_\mathbf{P} \mathsf{AnyPubVal}$. The same argument as in the previous paragraph gives us that the corresponding subterms of AV' and AV'' are related by $\cong_{\mathbf{P}}$. As the outermost constructor of AV' and AV'' is either pubenc or symenc, the definition of $\cong_{\mathbf{P}}$ gives us $AV' \cong_{\mathbf{P}} \mathsf{AnyPubVal} = AV$.

- The type of T is enc or symenc and T was constructed by the adversary. Then the only abstract value whose semantics contains T is $\mathsf{AnyPubVal}$ and $T \in [\![AV']\!]_{\mathcal{O}}$ is impossible.

- The type of T is data. Then AV' must be $\mathsf{store}(\mathsf{X_P})$ or $\mathsf{store}(\mathsf{X_S})$. By the definition of $\cong_{\mathbf{P}}$, $\mathsf{store}(\mathsf{X_P}) \cong_{\mathbf{P}} \mathsf{AnyPubVal}$. If $AV' = \mathsf{store}(\mathsf{X_S})$ then $AV' \in \mathbf{P}$ by the definition of adversarial well-constructedness.

- The type of T is list. The outermost constructor of AV' must be the tuple constructor. If there exists some $AV'' \in \tilde{\mathbf{P}}$, such that $T \in [\![AV'']\!]_{\mathcal{O}}$, then the argument is the same as for pubenc and symenc — the outermost constructor of AV'' is the tuple constructor, the corresponding components of AV' and AV'' are related by $\cong_{\mathbf{P}}$, and $AV'' \cong_{\mathbf{P}} \mathsf{AnyPubVal}$.

  If there exists no such AV'' then all subterms of T (call them $T_1, \ldots, T_k$) must be adversarially well-constructed, i.e. the adversary must know them and therefore $T_i \in [\![\mathsf{AnyPubVal}]\!]_{\mathcal{O}}$. If $AV' = (AV'_1, \ldots, AV'_k)$ then $T_i \in [\![AV'_i]\!]_{\mathcal{O}}$. By the induction assumption, $AV'_i \cong_{\mathbf{P}} \mathsf{AnyPubVal}$. and $AV' \cong_{\mathbf{P}} (\mathsf{AnyPubVal}, \ldots, \mathsf{AnyPubVal})$. By one of the properties of $\cong_{\mathbf{P}}$, $AV' \cong_{\mathbf{P}} \mathsf{AnyPubVal}$.

If $AV \neq \mathsf{AnyPubVal}$ and $AV' = \mathsf{AnyPubVal}$ then swap AV and AV' and apply the above argument. If $AV = AV' = \mathsf{AnyPubVal}$ then $AV \cong_{\mathbf{P}} AV'$. □