Computationally Sound Secrecy Proofs by Mechanized Flow Analysis

Michael Backes[1⋆] and Peeter Laud[2⋆⋆]

[1] Saarland University, Saarbrücken, Germany backes@cs.uni-sb.de
[2] Tartu University, Tartu, Estonia peeter.laud@ut.ee

1 Introduction

A large body of work exists for machine-assisted analysis of cryptographic protocols in the formal (Dolev-Yao) model, i.e., by abstracting cryptographic operators as a free algebra. In particular, proving secrecy by typing has shown to be a 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 [7] that was proved correct with respect to the computational model, i.e., if a protocol typechecks then it keeps secret the quantities handed to it by the protocol users in the computational sense. However, the major drawback of [7] was that type inference was not implemented so that this work did not entail an automated procedure for analyzing secrecy aspects of cryptographic protocols with cryptographic soundness guarantees.

We remedy this shortcoming by presenting a mechanized approach for soundly proving secrecy properties of cryptographic protocols by type inference. Our approach allows for more precise analyses compared with the type system of [7], is fully automated, and holds for an unbounded number of sessions. So far, we have successfully applied our implementation to different protocols of the Clark-Jacob library, in particular to the Needham-Schroeder(-Lowe), Yahalom and Otway-Rees protocols. Our approach is inspired by methods from control flow analysis and proceeds by recording for variables occurring at the protocol the locations where the values of these variables may have been created, and the shape (as terms) of these values. Our results (and the one of [7]) rely on the simulatable cryptographic library of Backes, Pfitzmann, and Waidner which constitutes one variant of Dolev-Yao models that entails proofs which are cryptographically sound in the strong sense of (blackbox) reactive simulatability/UC, a notion that essentially means strong composability and preservation of various security properties. While first security proofs of several common protocols have been hand-crafted using this library [3, 2], recent work has shown that the library is accessible to theorem proving techniques as well [8]. Our work shows that soundly proving secrecy properties via type inference is possible using this library, and it identifies cryptographically sound secrecy by typing as a promising direction for future work in general.

⋆ Supported by the German Research Foundation (DFG), grant #3194/1-1.
⋆⋆ Supported by Estonian Science Foundation, grant #6095, and by EU Integrated Project AEOLUS (contract no. IST-15964)

2 Execution Model

We use the same setup of a system as in [7]. In short, the abstract version of the cryptographic library 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, parsing, and sending of messages. The protocol 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.

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

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

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 of interest. The language contains public-key and symmetric-key encryption as the cryptographic primitives (and additionally nonces and lists). 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; a symmetric key may only encrypt keys of lower level. (Soundness in the case of encryption cycles has been recently proved as well, but under a stronger cryptographic definition.) The store- and $\mathsf{retrieve}$ expressions are used to convert payloads (data that can be communicated with the user) to handles and back. In $\mathbf{receive}_c[x_{\mathrm{p}}](x)$ and $\mathbf{send}_c[x_{\mathrm{p}}](x)$, 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 *channel name* c. A channel may be secure, authentic or insecure, or it may denote the communication with the user. 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. An active P_i additionally contains the received message and the currently running (output) process. When P_i receives the message, it is handed over to the first input process in its list of processes. A process may either accept the message or reject it by executing $\mathcal{JJ}$. A rejected message is passed to the next process in the list. When a process accepts the message, it executes until it has become a list of input processes I^*. This list I^* is 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.

3 Elements of the Constraint System

We want to collect the abstraction of possible values of the protocol variables, of the messages on channels, and of the knowledge of the adversary. Our constraint system contains a variable for each of these abstractions. Suitable inequalities between these variables are then introduced. All the inequalities have the form $E \subseteq X$ where E is a monotonic expression over the constraint system's variables, and X is a variable. Such constraints can be solved using iterative methods. An abstraction is a finite set of the following abstract values:

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

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

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 where these values have been created. The arguments b are bit-strings, they are also used to distinguish points of creation. Their meaning will be explained in the next paragraphs. 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.

As pointed out by Abadi and Blanchet [1] (the same ideas were included in [7]), it is important to distinguish two cases when analyzing asymmetric decryption — the ciphertext may be created by a protocol participant (and we can find out the possible plaintexts by recording which values were encrypted by which keys) or by the adversary, as the public key is in general known to it. In the latter case, the plaintext is abstracted by AnyPubVal. To distinguish these two cases we analyze the parts of the protocol following both possibilities separately, i.e., once assuming that the ciphertext came from a protocol participant, and once assuming that it was created by the adversary. Hence, for each protocol variable x we have 2^n variables in the constraint system where n is the number of public-key decryptions occurring before the definition of x (each variable may be defined only once). We denote these variables by $\mathbf{X}_x^b$ where b is a bit-string of length n. Having the i-th bit in b equal 1 (resp. 0) means that we assume that the ciphertext in the i-th public-key decryption was generated by a protocol participant (resp. the adversary). The elements b in abstract values have the same meaning. Here the length of b is the number of public-key decryptions above the program point ℓ.

As mentioned above, we also have a variable $\mathbf{P}$ in our constraint system that records an abstraction of the adversary's knowledge. The variable $\mathbf{P}$ does not appear in the left-hand sides of constraints whose right-hand side is not $\mathbf{P}$; all inputs from the adversary to other places are denoted by $\mathsf{AnyPubVal}$. As such, we do not have to add constraints expressing that the adversary can construct new terms. The constraints reflecting the computations of the adversary only correspond to the taking apart of messages.

For each secure or authentic channel c there is a variable $\mathbf{C}_c$. The data sent over insecure channels is recorded in $\mathbf{P}$. For each key generation (symmetric or asymmetric) at the program point ℓ we also have variables $\mathbf{E}^b_\ell$ recording which plaintexts may have been recorded with this key (here b has the same meaning as before). For each program point we also want to record whether it is reachable or not: for each ℓ of some **if**- or **let**-command we have the variables $\mathbf{L}^b_{\ell,\mathsf{true}}$ and $\mathbf{L}^b_{\ell,\mathsf{false}}$. The possible values of these variables are booleans (with $\mathsf{false} \leq \mathsf{true}$) and they record whether the true/default-branch or false-branch of that **if**- or **let**-command may be taken or not.

Examples of Constraints. Some more interesting assignments generate the following sets of constraints. Here b fixes the results of public-key decryptions before this assignment, and $\mathcal{X}$ maps protocol variables defined so far to constraint variables (in accordance with b).

$$\langle\!\langle y := \underline{\mathsf{pubenc}}^{\ell}(x_k, x_t)\rangle\!\rangle(\mathcal{X}, b) =$$

$$\left\{\begin{array}{l} \{\mathsf{pubenc}(AV_k, AV_t, \ell, b) \mid \\ \quad AV_k \in \mathcal{X}(x_k), AV_t \in \mathcal{X}(x_t), AV_k = \mathsf{pubkey}(\ldots)\} \subseteq \mathbf{X}^b_y, \\ \{\mathsf{pubenc}(\mathsf{AnyPubVal}, AV_t, \ell, b) \mid \\ \quad \mathsf{AnyPubVal} \in \mathcal{X}(x_k), AV_t \in \mathcal{X}(x_t)\} \subseteq \mathbf{X}^b_y \end{array}\right\} \cup$$

$$\{\mathsf{pubkey}(\ell', b') \in \mathcal{X}(x_k) \Rightarrow \mathcal{X}(x_t) \subseteq \mathbf{E}^{b'}_{\ell'} \mid \ell' \text{ is asymm. key generation}\}$$

$$\langle\!\langle y := \underline{\mathsf{pubdec}}(x_k, x_t)\rangle\!\rangle(\mathcal{X}, b) =$$

$$\left\{\begin{array}{l} \{AV_p \mid \mathsf{pubenc}(\mathsf{pubkey}(\ell'', b''), AV_p, \ell', b') \in \mathcal{X}(x_t), \\ \qquad \mathsf{seckey}(\ell'', b'') \in \mathcal{X}(x_k)\} \subseteq \mathbf{X}^{b1}_y, \\ \mathsf{AnyPubVal} \in \mathcal{X}(x_t) \Rightarrow \mathsf{AnyPubVal} \in \mathbf{X}^{b0}_y \end{array}\right\} \cup$$

$$\{\mathsf{AnyPubVal} \in \mathcal{X}(x_t) \wedge \mathsf{seckey}(\ell', b') \in \mathcal{X}(x_k) \Rightarrow \mathbf{E}^{b'}_{\ell'} \subseteq \mathbf{X}^{b1}_y \mid \ell', b'\}$$

A **let**-command generates the following constraints. Here $\bar{\mathbf{L}}$ is the constraint variable reflecting whether this command executes, and $\langle\!\langle e\rangle\!\rangle_{\mathrm{s}}$ [resp. $\langle\!\langle e\rangle\!\rangle_{\mathrm{f}}$] give some necessary conditions for e to succeed [resp. fail].

$$\langle\!\langle \mathbf{let}^\ell\ y := e\ \mathbf{in}\ P\ \mathbf{else}\ P' \rangle\!\rangle(\mathcal{X}, b, \bar{\mathbf{L}}) =$$
$$\{\bar{\mathbf{L}} \wedge \langle\!\langle e \rangle\!\rangle_{\mathrm{s}}(\mathcal{X}) \Rightarrow \mathbf{L}^b_{\ell,\mathsf{true}},\ \bar{\mathbf{L}} \wedge \langle\!\langle e \rangle\!\rangle_{\mathrm{f}}(\mathcal{X}) \Rightarrow \mathbf{L}^b_{\ell,\mathsf{false}}\} \cup$$
$$\big(\mathbf{L}^b_{\ell,\mathsf{true}} \Rightarrow \langle\!\langle y := e \rangle\!\rangle(\mathcal{X}, b)\big) \cup \langle\!\langle P \rangle\!\rangle(\mathcal{X}[y \mapsto \mathbf{X}^b_y], b, \mathbf{L}^b_{\ell,\mathsf{true}}) \cup \langle\!\langle P' \rangle\!\rangle(\mathcal{X}, b, \mathbf{L}^b_{\ell,\mathsf{false}}),$$

If e is $\underline{\mathsf{pubdec}}(\ldots)$ then the constraints are a bit different — the invocation of $\langle\!\langle P \rangle\!\rangle(\mathcal{X}[y \mapsto \mathbf{X}^b_y], b, \mathbf{L}^b_{\ell,\mathsf{true}})$ is replaced with

$$\langle\!\langle P \rangle\!\rangle(\mathcal{X}[y \mapsto \mathbf{X}^{b1}_y], b1, \mathbf{L}^b_{\ell,\mathsf{true}}) \cup \langle\!\langle P \rangle\!\rangle(\mathcal{X}[y \mapsto \mathbf{X}^{b0}_y], b0, \mathbf{L}^b_{\ell,\mathsf{true}}) \ .$$

$$\langle\!\langle \mathbf{send}_c[x_{\mathrm{p}}](x).I^* \rangle\!\rangle(\mathcal{X}, b, \bar{\mathbf{L}}) = \langle\!\langle I^* \rangle\!\rangle(\mathcal{X}, b, \bar{\mathbf{L}}) \cup \{\bar{\mathbf{L}} \Rightarrow \mathcal{X}(x_{\mathrm{p}}) \subseteq \mathbf{P},$$
$$\bar{\mathbf{L}} \wedge (c \text{ is sec. or auth.}) \Rightarrow \mathcal{X}(x) \subseteq \mathbf{C}_c, \bar{\mathbf{L}} \wedge (c \text{ is auth. or insec.}) \Rightarrow \mathbf{X}(x) \subseteq \mathbf{P}\}$$

$$\langle\!\langle (!)\mathbf{receive}_c[x_{\mathrm{p}}](x).P \rangle\!\rangle(\mathcal{X}, b, \bar{\mathbf{L}}) = \langle\!\langle P \rangle\!\rangle(\mathcal{X}[x \mapsto \mathbf{X}^b_x, x_{\mathrm{p}} \mapsto \mathbf{X}^b_{x_{\mathrm{p}}}], b, \bar{\mathbf{L}}) \cup$$
$$\{\bar{\mathbf{L}} \Rightarrow \mathsf{X_P} \in \mathbf{X}^b_{x_{\mathrm{p}}}, \bar{\mathbf{L}} \Rightarrow \mathcal{S} \subseteq \mathbf{X}^b_x\}$$

Here $\mathcal{S}$ is either $\mathbf{C}_c$ (c is sec. or auth.) or $\{\mathsf{AnyPubVal}\}$ (c is insec.) or $\{\mathsf{X_S}\}$ (c is from user).

The generated constraints — inequalities of the form $E \leq X$, where X is a constraint variable and E is a monotone expression over constraint variables — are solved using an iterative solver from the Goblin-project [9] which implements the technique of [6], tailored to systems of constraints; we are grateful to Vesal Vojdani for making that solver available to us. The iterative procedure of solving the constraints does not have to always terminate, as the sets of terms assigned to constraint variables may get arbitrarily large. However, this only happens if the protocol is able to generate terms of arbitrary complexity "all by itself", without the help from the adversary. This can happen if the execution of the protocol causes a term to be read from a channel (by a protocol participant), processed, and a larger term, but with the same shape, to be written back to the same channel. We do not believe that to ever happen in real protocols.

4 Relating Abstract Analyses and Cryptographic Secrecy

We finally give the precise theorem stating the condition under which our abstract analysis entails secrecy in the cryptographic sense.

Theorem. *Let $\mathbf{X}^b_x$, $\mathbf{P}$, $\mathbf{C}_c$, $\mathbf{E}^b_\ell$, $\mathbf{L}^b_{\ell,t}$ be a solution to the constraint system. Assume further that the following statements hold:*

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

(II) If $\mathsf{X_S} \in \mathbf{X}^b_x$ for some b and x then this x may only be store*-d or sent to the user.*

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

(IV) *If* $\mathsf{AV} \in \mathbf{E}_\ell^b$ *and the key generation at* ℓ *is* $\mathsf{gen\_symenc\_key}(i)$ *then the order of symmetric keys occurring unprotected in* AV *is less than* i.

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

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.*

These conditions are similar to the conditions (I)–(V) in [7].

References

1. M. Abadi, B. Blanchet. Secrecy types for asymmetric communication. Theoretical Computer Science, 298(3):387–415, 2003.
2. M. Backes. A Cryptographically Sound Security Proof of the Otway-Rees Protocol. In proc. ESORICS 2004, pp. 89–108.
3. M. Backes, B. Pfitzmann. A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol. In proc. FST TCS 2003, pp. 1–12.
4. M. Backes, B. Pfitzmann. Relating Symbolic and Cryptographic Secrecy. In proc. IEEE S& P 2005, pp. 171–182.
5. M. Backes, B. Pfitzmann, M. Waidner. A Universally Composable Cryptographic Library. In proc. ACM CCS 2003, pp. 220–230.
6. C. Fecht, H. Seidl. An Even Faster Solver for General Systems of Equations. In proc. SAS '96, pp. 189–204.
7. P. Laud. Secrecy Types for a Simulatable Cryptographic Library. In proc. ACM CCS 2005, pp. 26–35.
8. C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, M. Waidner. Cryptographically Sound Theorem Proving. In proc. CSFW 2006.
9. V. Vojdani. Mitmelõimeliste C-programmide kraasimine analüsaatoriga Goblin (Linting multi-threaded C programs with the Goblin). MSc thesis, Tartu University, 2006.