



Deciding a Fragment of (α, β) -Privacy

Laouen Fernet  and Sebastian Mödersheim 
DTU Compute, Richard Petersens Plads, Building 321
2800 Kongens Lyngby, Denmark
{lpkf,samo}@dtu.dk

October 2021

Abstract

We show how to automate fragments of the logical framework (α, β) -privacy which provides an alternative to bisimilarity-based and trace-based definitions of privacy goals for security protocols. We consider the so-called message-analysis problem, which is at the core of (α, β) -privacy: given a set of concrete messages and their structure, which models can the intruder rule out? While in general this problem is undecidable, we give a decision procedure for a standard class of algebraic theories.

Keywords: Privacy, Formal Methods, Security protocols, Automated verification.

1 Introduction

The problem of privacy in security protocols is relevant in many fields, such as electronic voting, digital health information, mobile payments and distributed systems in general. Privacy is a security goal of its own, it cannot be described as regular secrecy. For example, in voting it is not the values of the votes that are secret, since there is a public tally, but rather the *relation* between a voter and a vote. It is best if privacy is taken into account during the design of communication protocols. But even then, it is difficult to get enough guarantees about privacy goals. Formal methods are a successful way of addressing the issue. By studying a protocol at an abstract level, they can be used to check digital applications against possible misuse.

The symbolic modeling of protocols allows one to define various privacy goals. The standard approach uses the notion of *observational equivalence* [1, 2]: it is common to consider privacy as a bisimilarity between processes in the applied π -calculus. For instance, for electronic voting protocols, a privacy goal could be that two processes differing only by a swap of votes are *indistinguishable* [3, 4, 5]. There are many examples of communication protocols that are not secure with regards to privacy. This is the case also for protocols which

have been designed to provide some privacy goals. Indeed, recent papers show privacy issues in voting protocols (Helios [3, 4]) as well as contact-tracing applications (GAEN API [6], SwissCovid [7, 8]). While tools exist to provide automated verification [9, 10], it can be hard to formalize a privacy goal as a bisimilarity property, so automated verification is actually challenging. In such cases, it is hard to specify all desirable privacy goals using the notion of observational equivalence. Additionally, the standard approach cannot guarantee that the privacy goals verified cover all possibilities of misuse. These limits are the motivation for studying a new approach that is declarative and more intuitive.

(α, β) -privacy [11, 12] is an approach based on first-order logic with Herbrand universes, which allows for a novel way of specifying privacy goals. Instead of specifying pairs of things that should be indistinguishable to the intruder, one instead positively specifies what relations about the private data the intruder is *allowed* to learn and it is then considered a violation if the intruder is actually able to find out more.

The authors of [12] mainly argue that (α, β) -privacy is a more declarative way to specify goals without emphasis on questions of automation. For instance, they describe the goal of a voting protocol as releasing to the intruder the number of votes for each candidate or option and that this can actually justify the more technical “encoding” into bisimilarity-based approaches with the vote-swap idea mentioned before.

We now argue that actually the direct automation of (α, β) -privacy fragments can have advantages over bisimilarity approaches. The reason is that (α, β) -privacy is a reachability problem [13]: there is a state-transition system where every state is characterized by two Herbrand formulae α and β , namely what payload information α is currently published as well as the technical information β like exchanged messages between honest agents and intruder and what the intruder knows about the structure of these messages. The privacy question is now whether in any *reachable* state, β allows to rule out a model of α .

Thus, the main challenge lies in checking the (α, β) -privacy property for a given state, while in bisimilarity approaches, the main challenge lies in checking for every state S that is reachable in one process if there exists a reachable state S' in the other process so that S and S' are in a certain relation. This includes that the intruder knowledge in these states is statically equivalent, i.e., the intruder cannot tell S and S' apart. Bisimilarity thus means a challenge on top of static equivalence that is hard to handle in automated methods, while in (α, β) -privacy, reachability is trivial, but verifying privacy in the reached states is in general undecidable.

In this paper we show that for the fragment of *message-analysis problems* identified in [12] (and a suitable intruder theory), the check for (α, β) -privacy in each state is akin—and typically not more complex—than a static equivalence problem of the same size. For this fragment, (α, β) -privacy thus allows us to get rid of all the troubles of bisimilarity and reduce everything to a static-equivalence-style problem.

We present our first contributions in Section 3 by introducing the notions of

destructor theories and frames with shorthands. In Section 4, we present our main contribution under the form of several algorithms constituting a decision procedure. Proofs for our results are presented in Appendix A.

2 Preliminaries

2.1 Herbrand Logic

Much of the preliminaries are adapted from [12]. The approach of (α, β) -privacy is based on Herbrand logic [14], which is First-Order Logic (FOL) with Herbrand universes. A reachable state of a protocol will later be characterized by two formulae α and β in Herbrand logic.

In Herbrand logic, an alphabet $\Sigma = \Sigma_f \uplus \Sigma_i \uplus \Sigma_r$ consists of Σ_f the set of *free function symbols*, Σ_i the set of *interpreted function symbols* and Σ_r the set of *relation symbols*. The main difference to standard FOL (that has no free function symbols Σ_f) is that the universe is fixed by the set of terms that can be built using Σ_f . More precisely, let \mathcal{V} be a countable set of *variable symbols*, disjoint from Σ . We denote with $\mathcal{T}_\Sigma(\mathcal{V})$ the set of all *terms* that can be built from the function symbols in Σ and the variables in \mathcal{V} , i.e., a term is either a variable x or a function applied to subterms $f(t_1, \dots, t_n)$. We simply write \mathcal{T}_Σ when $\mathcal{V} = \emptyset$, and call its elements *ground terms* (over signature Σ). Let \approx be a congruence relation on \mathcal{T}_{Σ_f} .

The *Herbrand universe* U (over Σ and \mathcal{V}) is defined in the quotient algebra $\mathcal{A}_\approx = \mathcal{T}_{\Sigma_f / \approx}$, i.e., $U = \{\llbracket t \rrbracket_\approx \mid t \in \mathcal{T}_{\Sigma_f}\}$, where $\llbracket t \rrbracket_\approx = \{t' \in \mathcal{T}_{\Sigma_f} \mid t \approx t'\}$. The algebra interprets every n -ary function symbol $f \in \Sigma_f$ as a function $f^A : U^n \mapsto U$ such that $f^A(\llbracket t_1 \rrbracket_\approx, \dots, \llbracket t_n \rrbracket_\approx) = \llbracket f(t_1, \dots, t_n) \rrbracket_\approx$.

A (Σ, \mathcal{V}) -*interpretation* \mathcal{I} maps every interpreted function symbol $f \in \Sigma_i$ to a function $\mathcal{I}(f) : U^n \mapsto U$, every relation symbol $r \in \Sigma_r$ to a relation $\mathcal{I}(r) \subseteq U^n$, and every variable $x \in \mathcal{V}$ to an element $\mathcal{I}(x) \in U$. We extend \mathcal{I} to a function on $\mathcal{T}_\Sigma(\mathcal{V})$ as expected:

$$\mathcal{I}(f(t_1, \dots, t_n)) = f^A(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n)) \text{ for } f \in \Sigma_f$$

$$\mathcal{I}(f[t_1, \dots, t_n]) = \mathcal{I}(f)(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n)) \text{ for } f \in \Sigma_i$$

Note that we write $f[t_1, \dots, t_n]$ for $f \in \Sigma_i$ with square parentheses to visually distinguish interpreted functions from free functions. The rest of the syntax and semantics is like in standard FOL. We write $\mathcal{I} \models \phi$ when a formula ϕ over Σ and \mathcal{V} is true in a (Σ, \mathcal{V}) -interpretation \mathcal{I} , and we then call \mathcal{I} a (Σ, \mathcal{V}) -*model*. We may just say *interpretation* and *model* when Σ and \mathcal{V} are clear from the context. We also say ϕ *entails* ψ and write $\phi \models \psi$, if all ϕ -models are ψ -models.

We employ the standard syntactic sugar and write, for example, $\forall x.\phi$ for $\neg\exists x.\neg\phi$ and $x \in \{t_1, \dots, t_n\}$ for $x = t_1 \vee \dots \vee x = t_n$. Slightly abusing notation, we will also consider a substitution $[x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$ as a formula $x_1 = t_1 \wedge \dots \wedge x_n = t_n$. This allows us to write $\theta \models \phi$ for a substitution θ and a formula ϕ that has no symbols to interpret other than variables in the domain of θ . In particular, we can write $\sigma' \models \sigma$ when the substitution σ' is an instance of σ . We denote with ε the identity substitution.

2.2 Frames

We use frames to represent the knowledge of the intruder. The idea is that the intruder has recorded a number of messages and can refer to them using labels. We identify a subset $\Sigma_{op} \subseteq \Sigma_f$ of free functions, that we call *cryptographic operators*. They are used to represent a black-box model of cryptography, which is defined with a set of algebraic equations.

Definition 1 (Frame). *A frame is written as $F = \{l_1 \mapsto t_1, \dots, l_k \mapsto t_k\}$, where the l_i are distinguished constants and the t_i are terms that do not contain any l_i . We call $\{l_1, \dots, l_k\}$ and $\{t_1, \dots, t_k\}$ the domain and the image of the frame, respectively. The set $\mathcal{R}_F = \mathcal{T}_{\Sigma_{op}}(\{l_1, \dots, l_k\})$ is the set of recipes, i.e., the least set that contains l_1, \dots, l_k and that is closed under all the cryptographic operators of Σ_{op} . We will simply write \mathcal{R} when F is clear from the context.*

A frame F can be regarded as a substitution that replaces every l_i of its domain with the corresponding t_i . For a recipe r , we thus write $F\{r\}$ for the term obtained by applying this substitution to r . A *generable term* is any term t for which there is a recipe r with $t \approx F\{r\}$. Note that by default, the intruder does not know all constants but they can be explicitly included in the frame if needed.

Two frames F_1 and F_2 with the same domain are *statically equivalent*, written $F_1 \sim F_2$, if the intruder cannot distinguish them, i.e., when for all pairs of recipes r_1 and r_2 it holds that $F_1\{r_1\} \approx F_1\{r_2\} \iff F_2\{r_1\} \approx F_2\{r_2\}$. It is possible to axiomatize in Herbrand logic the notions of frames, recipes, generable terms, and static equivalence of frames [12].

2.3 (α, β) -Privacy

The idea of (α, β) -privacy is to declare a *payload-level formula* α over an alphabet $\Sigma_0 \subset \Sigma$ at the abstract level, defining intentionally released information (for instance the number of votes cast in an election), and a *technical-level formula* β over the full Σ , including all information visible to the intruder (e.g., cryptographic messages of a voting system and information about their structure). Intuitively, we want that the intruder does not learn from β anything on the payload-level that does not already follow from α , i.e., every model of α can be extended to a model of β :

Definition 2 (Model-theoretical (α, β) -privacy). *Let Σ be a countable signature, $\Sigma_0 \subset \Sigma$ a payload alphabet, α a formula over Σ_0 and β a formula over Σ such that $fv(\alpha) = fv(\beta)$, both α and β are consistent and $\beta \models \alpha$. We say that (α, β) -privacy holds iff for every $(\Sigma_0, fv(\alpha))$ -model $\mathcal{I} \models \alpha$ there exists a $(\Sigma, fv(\beta))$ -model $\mathcal{I}' \models \beta$, such that \mathcal{I} and \mathcal{I}' agree on the interpretation of all interpreted function and relation symbols of Σ_0 and all free variables of α .*

3 The Fragment

In the (α, β) -privacy framework, we have a state transition system where every state contains at least a pair of formulae α and β , as well as other information to represent the current state of some honest agents. Privacy is then a *reachability* problem [13], i.e., whether we can reach a state where β allows the intruder to exclude at least one model of α . We focus in this paper only on the problem for a single state, i.e., deciding (α, β) -privacy for a given pair (α, β) .

Even this is undecidable due to the expressiveness of Herbrand logic. We therefore restrict ourselves in this paper to (α, β) -pairs of a particular form that is called *message-analysis problem* in [12], which consists of two restrictions. The first restriction here is that the payload alphabet Σ_0 is a finite set of free constants that are part of Σ_{op} . We say in this case that α is *combinatoric*. Thus the Herbrand universe U of Σ_0 is also finite, and every model of α is just a mapping from $fv(\alpha)$ to U . We will write θ for such a mapping in the following. For example if $\alpha \equiv x \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\} \wedge y \in \{\mathbf{a}, \mathbf{b}\} \wedge x \neq y$, then $\theta = [x \mapsto \mathbf{a}, y \mapsto \mathbf{b}]$ is a model of α . We also call $fv(\alpha)$ the *privacy variables*, and say the domain of a privacy variable x are those values from Σ_0 that x can have in any model of α . We denote with Θ the set of all models of α . The second restriction is that in every reachable state of the system, the intruder knowledge can be characterized by a frame *struct* where the messages can contain variables from α , and a frame *concr* = $\theta(\text{struct})$, where θ is a model of α representing the true values of the privacy variables in this state, and thus *concr* are the concrete messages that the intruder observes. The formula β then consists of α , the definition of *struct* and *concr*, and stipulates that $\text{struct} \sim \text{concr}$.

Example 1. Consider a structural frame

$$\text{struct} = \{ l_1 \mapsto \text{script}(k, x), l_2 \mapsto \text{script}(k, y), l_3 \mapsto \text{script}(k, z) \}$$

and the model $\theta = [x \mapsto 0, y \mapsto 1, z \mapsto 0]$, where the variables x, y, z in *struct* represent some votes that have been symmetrically encrypted (*script*) by a trusted authority with a key k . (We formally introduce this algebraic theory in Example 3.) Let the payload formula be $\alpha \equiv x, y, z \in \{0, 1\}$. The intruder is not able to learn the values of the votes without the key. However, they¹ can observe that $\text{concr}\{ l_1 \} \approx \text{concr}\{ l_3 \}$. Using static equivalence between *struct* and *concr*, the intruder deduces that $\text{struct}\{ l_1 \} \approx \text{struct}\{ l_3 \}$. The only way to unify the equation, with respect to \approx , is with $x = z$. This constitutes a breach of privacy, as it does not follow from α (some models have been excluded). There are also other relations that can be derived at this point: $x \neq y$ and $y \neq z$.

The problem we solve in this paper is thus: given an (α, β) -pair that is a message-analysis problem, check whether (α, β) -privacy holds. Note here a fundamental difference with respect to approaches based on static equivalence of frames where privacy means that the intruder cannot distinguish two frames

¹We use the pronoun “they” for gender-neutral expression.

that represent different possibilities. In (α, β) -privacy, in contrast, we have a symbolic frame *struct* and an instance *concr*, and the intruder *knows* that *concr* is the instance of *struct* that represents what really happened. Thus the intruder can exclude every model of α under which *struct* and *concr* would be distinguishable.

Interestingly, the problem of (α, β) -privacy in a message-analysis problem *is* related to static equivalence of frames. As [12] observes, in theory one could compute *all* models of the given α (there are finitely many since α is combinatoric) and compute $concr_i = \theta_i(struct)$ for every model θ_i ; then (α, β) -privacy holds iff the $concr_i$ are all statically equivalent. This is practically not feasible, since the number of models is in general in the order of $|\Sigma_0|^{fv(\alpha)}$. The algorithms we present here will typically not be more complex than standard static equivalence algorithms (in the same algebraic theory). However, in corner cases our current implementation can produce in general an exponential set of recipes. It is part of the future work to investigate whether this can be avoided with another representation that avoids the enumeration of combinations.

Reachable States Before we go into detail about the algorithm itself, we want to briefly sketch what kinds of protocol descriptions can be considered, so that in every reachable state we have a message-analysis problem. This is only a sketch because we lack the space to make a fully-fledged definition. What we can support with message-analysis problems is basically what one would have in strand spaces: protocols where every role can be described as a linear sequence of message exchanges. This corresponds in the applied π -calculus to processes for roles that do not contain any repetition, parallelism, or branching. That is, when checking incoming messages with an `if` or `let` statement, the `else` branch has to be empty (i.e., when the conditions are not satisfied, the protocol aborts). In this case, the intruder always learns the outcome of the check, and for privacy it is sometimes interesting to consider protocols that can hide this, e.g., sending in the negative case a decoy-answer [15]. This is a generalization of the message-analysis problem, i.e., the intruder in general does no longer know the structure of a message for sure, but only that it is one of several possibilities, say $struct_1, \dots, struct_n$, and figuring out which $struct_i$ it is may allow for breaking the privacy. This requires an extension to our algorithms that is not overly difficult, but we lack the space to present it here. However, with message-analysis problems we cover the realm of standard security protocols that could be written in Alice-and-Bob notation.

In addition to normal variables for received messages, we have the mentioned privacy variables (recall they are non-deterministically chosen from a given subset of Σ_0). The formalism for describing the state transition system should thus include a mechanism to specify the choice of such variables, and what information about them is released by augmenting α upon state transitions. Note that the intruder is active and can send a message determined by a recipe over the domain of *struct* in that state. Since *struct* contains privacy variables, the intruder can “experiment” by sending a message with a privacy variable to an

honest agent, and thus observe if there is an answer (i.e., passing checks that the agent makes) and learn the message structure of the answer.

Example 2. Consider a door with a tag reader. Agents a, b, c can use a personal tag to open the door; their tags each have a symmetric key $k(a)$, $k(b)$ and $k(c)$, respectively (where k is a private free function). The toy protocol is that the reader sends a nonce and the tag replies with the encrypted nonce. For instance the following state is reachable in two protocol executions: the structural knowledge is $struct = \{ \{ l_1 \mapsto n, l_2 \mapsto \text{crypt}(k(x_1), n), l_3 \mapsto n', l_4 \mapsto \text{crypt}(k(x_2), n') \} \}$, where n, n' represent nonces and x_1, x_2 are variables for agent names, and the concrete instantiation is $\theta = [x_1 \mapsto a, x_2 \mapsto a]$, i.e., both interactions were with the same agent $x_1 = x_2 = a$. The privacy goal of *unlinkability* can be expressed by a payload formula that every agent variable can be any of the agents, i.e., in the example state we have $\alpha \equiv x_1, x_2 \in \{a, b, c\}$. Thus, the intruder should not be able to tell whether replies come from the same agent. If the intruder is just passively listening (as in the example state above), unlinkability indeed holds (since the nonces n and n' are different). However, if the intruder impersonates a reader and replays a nonce n to a tag, we would get to the state $struct = \{ \{ l_1 \mapsto n, l_2 \mapsto \text{crypt}(k(x_1), n), l_3 \mapsto n, l_4 \mapsto \text{crypt}(k(x_2), n) \} \}$. Here, they can deduce that $\text{crypt}(k(x_1), n) \approx \text{crypt}(k(x_2), n)$ and thus $x_1 = x_2$. This could be fixed by including also a nonce from the tag in the message, but note this is only a toy protocol, and one would need to also solve distance bounding.

3.1 Destructive Theories

Even with the restriction to message-analysis problems, (α, β) -privacy is still undecidable, since the *word problem* (whether $s \approx t$, given s and t) in algebraic theories is. We restrict ourselves here to theories we call *destructor theories*, a concept similar to subterm-convergent theories. The main difference is that we like to distinguish constructors like encryption and destructors like decryption and be able to verify if the application of a destructor was successful.

This verification is motivated by the fact that most modern cryptographic primitives allow one to check whether a decryption is successful or not, e.g., by including MACs or specific padding. In some protocol verification approaches, this is modeled by applying encryption again to the result of the decryption and comparing with the original term, i.e., checking $\text{crypt}(\text{pub}(k), \text{dcrypt}(\text{priv}(k), c)) \approx c$. This seems a bit absurd and would not work with randomized encryption in general. We therefore model destructors to yield an error value if it is applied to terms for which it does not work. Given that this error message does not normally occur in protocols, we can regard this as destructors having the return type `Maybe Msg` in Haskell notation, i.e., returning `Just r` if successful or `Nothing` in case of an error. This allows us to discard all “garbage terms” and makes reasoning a bit simpler.

Definition 3 (Destructor theory). A destructor theory *consists of*

- a set $\Sigma_{pub} \subseteq \Sigma_f$ of public functions that the intruder is able to apply; it is

further partitioned into constructors and destructors. Let in the following constr and destr range over constructors and destructors, respectively.

- a set E of algebraic equations of the form $\text{destr}(k, \text{constr}(t_1, \dots, t_n)) = t_i$, where $i \in \{1, \dots, n\}$, $\text{fv}(k) \subseteq \text{fv}(t_1, \dots, t_n)$ and the symbols of E are disjoint from Σ_0 . The first argument of a destructor is called a key.²

We also require that for any two equations $\text{destr}(k, \text{constr}(t_1, \dots, t_n)) = t_i$ and $\text{destr}(k', \text{constr}'(t'_1, \dots, t'_m)) = t'_j$ of E , it must be the case that either

- $\text{constr} \neq \text{constr}'$ or
- $k = k'$, $n = m$, and $t_1 = t'_1, \dots, t_m = t'_m$.

i.e., when we deal with the same constructor, the respective subterms and keys must be the same (but the extracted t_i and t'_j may be different).

Finally, every destructor occurs in only one equation.

Let \approx_0 be the least congruence relation on ground terms induced by E . We define the congruence \approx of the destructor theory as the least congruence relation over ground terms that subsumes \approx_0 and such that for all ground terms k and m $\text{destr}(k, m) \approx \text{error}$ whenever $\text{destr}(k, m) \not\approx_0 m'$ for all destructor-free m' . Here error is a distinguished constant in $\Sigma \setminus \Sigma_0$.

Finally, we require that in all given frames, the image contains no destructors (and the algorithms will preserve this property).

Note that the error behavior cannot directly be represented by algebraic equations because of the negative side-condition. However, observe that the underlying theory E gives rise to a term rewriting system (replacing $=$ with \rightarrow) that is convergent: termination is obvious and for confluence observe that there are no critical pairs (see, e.g., [16]). This gives immediately a decision procedure for the word problem in \approx_0 (normalize and compare syntactically) and in \approx (build the \approx_0 -normal forms, replace all remaining destructor-subterms by error ; again compare syntactically).

3.2 Unification and All That

In general, we will deal with terms that contain variables, albeit only privacy variables, i.e., ranging over constants of Σ_0 . Thus destructor-free symbolic terms cannot give rise to a redex and we can use the standard syntactic unification algorithm on destructor-free terms—with one exception. We need to adapt the unification of variables slightly: the unification of x with t is only possible if either t is a constant in the domain of x , or another variable y such that their domains have a non-empty intersection; their domains are then restricted to this intersection. Since a substitution $[x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$ can be expressed as a set of equations $\{x_1 = t_1, \dots, x_n = t_n\}$, we allow to use the notation

²For some destructors, e.g., opening a pair, one does not need a key; for uniformity one could use here a fixed public constant as a dummy value, but slightly abusing notation, we just omit the key argument in such a case.

$unify(\sigma_1, \dots, \sigma_n)$ for a most general unifier (MGU) of all equations from the σ_i .

3.3 The *ana* Function

Finally, we can repack the destructor equations into a function *ana* that, given a term with a constructor, yields which destructors may be applicable:

Definition 4.

$$ana(\text{constr}(t_1, \dots, t_n)) = (k, \{(\text{destr}, t_i) \mid \text{destr}(k, \text{constr}(t_1, \dots, t_n)) = t_i \in E\})$$

Intuitively, given a term that can be decrypted, ana returns the key required for decryption and all derivable terms according to the algebraic equations.

Example 3. The theory we use in examples throughout this paper is as follows (adapted from [12]). Let $\Sigma = \Sigma_f \uplus \Sigma_i \uplus \Sigma_r$ be an alphabet and \mathcal{V} a set of variables. We consider the cryptographic operators defined in Table 1 and $\Sigma_{pub} = \Sigma_{op}$.

- $\text{pub}(s)$ and $\text{priv}(s)$ represent an asymmetric key pair from a secret seed (where the lack of destructors reflects that it is hard to find the seed from the keys);
- $\text{crypt}(p, r, t)$ and $\text{dcrypt}(p', t)$ formalize asymmetric encryption with randomness;
- $\text{sign}(p', t)$ and $\text{retrieve}(p', t)$ formalize digital signatures;
- $\text{scrypt}(k, t)$ and $\text{dscrypt}(k, t)$ formalize symmetric cryptography;
- pair , proj_1 and proj_2 formalize serialization;
- h is a cryptographic hash function (where the lack of destructors reflects that it is hard to find a pre-image).

Table 1: Example set Σ_{op}

| Constructors | Destructors | Properties |
|--------------|---------------------------------------|--|
| pub, priv | | |
| crypt | dcrypt | $\text{dcrypt}(\text{priv}(s), \text{crypt}(\text{pub}(s), r, t)) = t$ |
| sign | retrieve | $\text{retrieve}(\text{pub}(s), \text{sign}(\text{priv}(s), t)) = t$ |
| scrypt | dscrypt | $\text{dscrypt}(k, \text{scrypt}(k, t)) = t$ |
| pair | proj ₁ , proj ₂ | proj ₁ (pair(t_1, t_2)) = t_1 proj ₂ (pair(t_1, t_2)) = t_2 |
| h | | |

In case there is no key required, the argument is omitted as written in the equations in Table 1. We introduce a “dummy” key k_0 known by the intruder

covering this case for the return value of *ana*.

$$ana(t) = \begin{cases} (\text{priv}(s), \{\{\text{dencrypt}, t'\}\}) & \text{if } t = \text{crypt}(\text{pub}(s), r, t') \\ (k, \{\{\text{dscript}, t'\}\}) & \text{if } t = \text{script}(k, t') \\ (\text{pub}(s), \{\{\text{retrieve}, t'\}\}) & \text{if } t = \text{sign}(\text{priv}(s), t') \\ (k_0, \{\{\text{proj}_1, t_1\}, \{\text{proj}_2, t_2\}\}) & \text{if } t = \text{pair}(t_1, t_2) \\ (k_0, \{\}) & \text{otherwise} \end{cases}$$

3.4 Frames with Shorthands

We define an extension of the concept of frames to easily handle decryption of terms. A frame with shorthands consists in a frame with additional labels, which are actually recipes over the initial labels.

Definition 5 (Frame with shorthands). *A frame with shorthands is written as $F' = \{\{l_1 \mapsto t_1, \dots, l_k \mapsto t_k, m_1 \mapsto s_1, \dots, m_n \mapsto s_n\}\}$, where $F = \{\{l_1 \mapsto t_1, \dots, l_k \mapsto t_k\}\}$ is a frame, the m_j are recipes over the l_i and $F \{\{m_j\}\} \approx s_j$. We call the mappings $m_1 \mapsto s_1, \dots, m_n \mapsto s_n$ shorthands. The domain of a frame with shorthands is defined to be the domain of the underlying frame.*

We will treat these m_j like the labels l_i . As a consequence, the set $\mathcal{R}_{F'}$ is now $\mathcal{T}_{\Sigma_{op}}(\{l_1, \dots, l_k, m_1, \dots, m_n\})$, i.e., all the shorthands can be used. This gives the same recipes as \mathcal{R}_F , but the shorthands make a difference when we restrict ourselves to *constructive recipes*, i.e., recipes without destructors which we define as $\mathcal{R}_F^c = \mathcal{T}_{\Sigma_{op}^c}(\{l_1, \dots, l_k\})$ and $\mathcal{R}_{F'}^c = \mathcal{T}_{\Sigma_{op}^c}(\{l_1, \dots, l_k, m_1, \dots, m_n\})$ where Σ_{op}^c are the constructors. Thus $\mathcal{R}_{F'}$ can use destructors from the shorthands, but otherwise only constructors, and thus in general $\mathcal{R}_{F'}^c \supseteq \mathcal{R}_F^c$. Similarly, we say that a term t is *constructive* if it does not contain any destructor.

Recall that initially all terms in a frame's image are constructive. Our algorithms will ensure that all s_j added through shorthands are also constructive.

Example 4. Let $k \in \mathcal{T}_{\Sigma}(\mathcal{V})$ and $x \in \mathcal{V}$. Consider the frames

$$\begin{aligned} F &= \{\{l_1 \mapsto \text{script}(k, x), l_2 \mapsto k\}\} \\ F' &= \{\{l_1 \mapsto \text{script}(k, x), l_2 \mapsto k, m_1 \mapsto x\}\} \end{aligned}$$

where $m_1 = \text{dscript}(l_2, l_1)$. Here F' is the frame F with the shorthand $m_1 \mapsto x$. Indeed, we have that $F \{\{\text{dscript}(l_2, l_1)\}\} = \text{dscript}(k, \text{script}(k, x)) \approx x$.

4 Decision Procedure

We now give a decision procedure for the fragment of (α, β) -privacy that we have defined in the previous section: a message-analysis problem with respect to a destructor theory. We are thus given a triple $(\alpha, \text{struct}, \theta)$ where α expresses the privacy goal at this state and the models of α can be characterized by

substitutions from the free variables of α to constants of Σ_0 . The substitution θ is one of the models, namely what is the reality, i.e., the true value of the free variables of α . Finally, *struct* is a frame with privacy variables representing all the messages that the intruder received in the exchange with honest agents up to this state. This means that the intruder knows the structure of each message, because the protocol description is public and there is no branching; what the intruder might not know is the value θ of the privacy variables (as well as constants representing strong keys and nonces). The intruder also knows the concrete messages $concr = \theta(struct)$. The question our algorithm will answer is what models of α the intruder can exclude from this, i.e., the $\theta' \models \alpha$ such that $concr \not\sim \theta'(struct)$. To avoid enumerating all models (there are exponentially many in general) and to be able to easily integrate our algorithm with reasoning about other constraints, the algorithm returns a set of equations and inequations that can be derived by the intruder.

4.1 Composition

4.1.1 Composition in a Structural Frame

This first piece of the procedure is concerned with the intruder composing messages, i.e., using only constructive recipes. Note that the intruder can also use shorthands that represent the result of previous decryption operations. This composition task is similar in many intruder algorithms: either the goal term t is directly in the knowledge or it is of the form $f(t_1, \dots, t_n)$ where f is a public constructor and the t_i can be composed recursively. The novelty of our algorithm here is that both the terms in *struct* and t may contain privacy variables, and composition may reveal information about these variables to the intruder. For a variable $x \in \mathcal{V}$, the intruder knows all values in the domain of x . Thus, if the variable occurs in a term to compose with only public constructors, they can compare all possibilities and see which one is correct, i.e., to what constant the variable x is mapped. Much of this evaluation must be postponed to a later stage of the algorithm. For now the composition algorithm just computes under which values of the variables the goal term t can be produced, i.e., it returns a set of pairs (r, σ) of a recipe r and a substitution σ where σ is an MGU under which r produces the goal t .

Example 5. As in Example 1, $struct = \{ |l_1 \mapsto \mathbf{scrypt}(k, x), l_2 \mapsto \mathbf{scrypt}(k, y), l_3 \mapsto \mathbf{scrypt}(k, z) \}$ and $\theta = [x \mapsto 0, y \mapsto 1, z \mapsto 0]$. The intruder has several ways to compose the term $\mathbf{scrypt}(k, x)$, depending on which model of α is true:

$$composeUnder(\theta, struct, \mathbf{scrypt}(k, x)) = \{(l_1, \varepsilon), (l_2, [x \mapsto y]), (l_3, [x \mapsto z])\}$$

The other algorithms will actually rule out $[x \mapsto y]$ since $\theta \not\models x = y$.

We argue that the algorithm is correct, in the sense that the pairs found by this algorithm really allow to compose the term in the given frame, under a unifier; the algorithm finds all constructive recipes together with an MGU.

Algorithm 1: Composition in a structural frame

```
1 composeUnder( $\theta, struct, t$ ) =  
2   let  $RU = \{(l, \sigma) \mid l \mapsto t' \in struct, \sigma = unify(t = t')\}$  in  
3   if  $t \in \mathcal{V}$  then  
4      $RU \cup \{(\theta(t), [t \mapsto \theta(t)])\}$   
5   else if  $t = f(t_1, \dots, t_n)$  and  $f \in \Sigma_{pub}$  then  
6      $RU \cup \{(f(r_1, \dots, r_n), \sigma) \mid (r_1, \sigma_1) \in composeUnder(\theta, struct, t_1),$   
7        $\dots,$   
8        $(r_n, \sigma_n) \in composeUnder(\theta, struct, t_n),$   
9        $\sigma = unify(\sigma_1, \dots, \sigma_n)\}$   
10  else  
11   $RU$ 
```

Theorem 1 (Correctness of *composeUnder*). *Let θ be a substitution, $struct$ be a frame and $t \in \mathcal{T}_\Sigma(\mathcal{V})$. Then*

1. $\forall (r, \sigma) \in composeUnder(\theta, struct, t), \sigma(struct\{r\}) = \sigma(t)$.
2. $\forall r \in \mathcal{R}^c, \exists \tau, \tau(struct\{r\}) = \tau(t) \implies (\exists \sigma, (r, \sigma) \in composeUnder(\theta, struct, t) \text{ and } \tau \models \sigma)$.

4.1.2 Composition in a Ground Frame

At the concrete level, the terms in the frame are all *ground*, i.e., they do not contain variables. The intruder does not have to reason about possible variable instantiations but only cares about the recipes they can use. This can be seen as a special case of the previous algorithm. We will use the function *compose* which does the same as *composeUnder* but drops the unifiers attached to the recipes (they are always the identity, for a ground frame and a ground term).

4.2 Analysis

The next step in our procedure is to augment the frame with shorthands as far as possible with messages the intruder can decrypt. This follows again common lines of intruder deduction reasoning, namely performing a saturation [17], but there are several crucial differences here. While the standard approach in static equivalence of frames just looks at each frame in isolation and computes a set of subterms that are derivable, we need to look at both *concr* and *struct* side by side here, because some analysis steps may only be possible for some instances of *struct*. Roughly speaking, if a decryption step is possible in *concr* but not in all instances of *struct*, we can exclude those instances, and vice-versa, if a decryption step is possible in some instances of *struct*, but not in *concr*, we can exclude those.

The intruder analyzes *struct* and adds shorthands for terms that can be decrypted. This will make all derivable subterms available with only composition (constructive recipes).

Example 6. Let $k_1, k_2, a \in \Sigma_0$ and $x, y, z \in \mathcal{V}$. Consider the substitution $\theta = [x \mapsto k_1, y \mapsto a, z \mapsto k_1]$ and the frame $struct = \{ \{ l_1 \mapsto \mathbf{s}crypt(x, y), l_2 \mapsto z \} \}$. Then the analysis extends the frame by adding a shorthand like so: $struct_{ana} = \{ \{ l_1 \mapsto \mathbf{s}crypt(z, y), l_2 \mapsto z, \mathbf{d}crypt(l_2, l_1) \mapsto y \} \}$. Since the decryption is successful in $concr = \theta(struct)$, the intruder is able to compose the key in *struct* with the same recipe l_2 . This also enables the intruder to learn that $x = z$. Note that x is changed to z in the frame because $concr\{ \mathbf{d}crypt(l_2, l_1) \} \not\approx \mathbf{error}$, so we can rule out all instances of x and z so that $struct\{ \mathbf{d}crypt(l_2, l_1) \} \approx \mathbf{error}$. However, there are more relations that could be deduced. For instance, the intruder is now able to check the pair of recipes $(l_2, \mathbf{d}crypt(l_2, l_1))$ with composition only (using the shorthand). The intruder can therefore learn that also $x \neq y$, but this is handled by the final algorithm *findRelations* below.

Consider the same *struct* but with $\theta = [x \mapsto k_1, y \mapsto a, z \mapsto k_2]$, so that the above analysis step is not possible. When trying to compose the key x , the algorithm *composeUnder* returns $(l_2, [x \mapsto z])$ as a possibility. This does not work in *concr*, so the intruder cannot actually obtain a new term, but conclude that $x \neq z$.

We define a recursive function *analyzeRec* that will apply one analysis step from calling *ana*, add terms if the decryption was successful, and call itself to perform the other analysis steps. To tackle the problem, we first consider that the intruder knowledge has been split into three frames. That way, we can make the distinction between the terms that have to be analyzed in the future, the terms that might be decrypted later, and the terms that have already been completely analyzed. Note that we do need to consider the terms “on hold”, i.e., that might be decrypted later, because the intruder might learn at a later point how to compose the required key.

The wrapper function *analyze* simply calls *analyzeRec* with the arguments properly initialized. All terms are initially considered “new” because they have to be analyzed. There are, at the start, no elements “on hold” or “done”. The intruder does not know any equations between the variables at the beginning, so we indicate the identity substitution ε as the initial value. Moreover, we also indicate an empty set as the initial value of the set *Ex* of substitutions excluding some models of the variables (“exceptions”).

The result of applying *ana* gives the key required to decrypt the term, and a set *FT* of pairs (function, term) of derivable terms. If the decryption fails in *concr*, i.e., the key cannot be composed at the concrete level, then it also fails in *struct* and no new terms can be added. However, since composition of the key at the structural level might be possible even in this case, the unifiers allowing to compose the key in *struct* exclude some models. We add such substitutions to the set *Ex*. Note that in the algorithms we write l as a label even though it can actually be a recipe, because we treat the recipes from shorthands as regular labels.

If the decryption is successful in *concr*, then it is also successful in *struct* and we can define recipes for the new terms. The shorthands added at this point use the destructors paired with the new terms, and some recipe found for composing the key in *concr*. The choice of this recipe is irrelevant: we also add a shorthand in *D* for the key, if there is not one already in the frame, so that we can later check the different ways to compose it. The keyword “**pick**” in the definition below refers to this choice, it means “take any one element from the set”.

We put the new mappings in a frame LT_{new} and add this to the new terms to analyze. We do not need to add terms for which the intruder already has a label or shorthand. All terms that were on hold also need to be analyzed again, as the intruder might be able to successfully decrypt them with the new knowledge. We apply the substitution σ_{new} , required to compose the key with the different recipes the intruder found in *concr* for the corresponding ground key, to all terms in the split frame so that the shorthands are correct. We update the equations that the intruder found by unifying with the previous substitution σ .

The analysis adds shorthands for any successful decryption of terms. The function *analyze* also preserves the property of static equivalence between *struct* and *concr*. Recall that Θ denotes the set of models of α . Our results are expressed over Θ so that they can be used to check whether some models can be excluded. The algorithm presented here does not simply return the analyzed frame, but also a unifier σ and a set of substitutions *Ex*. The intruder knows that the concrete instantiation of variables is an instance of σ and can exclude all substitutions in *Ex*. These properties are formally expressed in Theorem 2.

Theorem 2 (Correctness of *analyze*). *Let θ be a substitution, $struct$ be a frame and $(struct_{ana}, \sigma, Ex) = analyze(\theta, struct)$. Then*

1. $\forall r \in \mathcal{R}, struct_{ana}\{r\} \approx \sigma(struct\{r\})$.
2. $\forall r \in \mathcal{R}, \exists r' \in \mathcal{R}_{struct_{ana}}^c, struct_{ana}\{r'\} \approx \sigma(struct\{r\})$.
3. $\forall \theta' \in \Theta, \theta'(struct) \sim \theta(struct) \implies \theta' \models \sigma \wedge \bigwedge_{\sigma' \in Ex} \neg \sigma'$.
4. $\forall \theta' \in \Theta, \theta' \models \sigma \implies (\theta'(struct) \sim \theta(struct) \iff \theta'(struct_{ana}) \sim \theta(struct_{ana}))$

Theorem 3 (Termination of *analyze*). *Let θ be a substitution and $struct$ be a frame. Then the call $analyze(\theta, struct)$ terminates.*

4.3 Intruder Findings

The final algorithm we present generates a formula ϕ , which contains all equations and inequations between variables that the intruder is able to derive from their knowledge. We argue that, after analysis, all checks that the intruder can do to compare *struct* and *concr* are covered by only composing the terms

Algorithm 2: Analysis of a structural frame

```

1 analyze( $\theta$ , struct) =
2    $\lfloor$  analyzeRec( $\theta$ , struct,  $\{\} \}, \{\} \}, \varepsilon, \{\})$ 
3 analyzeRec( $\theta$ , N, H, D,  $\sigma$ , Ex) =
4   if N =  $\{\} \}$  then
5      $\lfloor$  (H  $\cup$  D,  $\sigma$ , Ex)
6   else
7     let  $\{\!| \mapsto t \!\}$   $\cup$  LT = N
8       (k, FT) = ana(t)
9       struct = N  $\cup$  H  $\cup$  D
10      concr =  $\theta$ (struct)
11      SR = composeUnder( $\theta$ , struct, k)
12      GR = compose(concr,  $\theta$ (k))
13       $\sigma_{new}$  = unify( $\{\sigma \mid (r, \sigma) \in SR, r \in GR\}$ )
14      Exnew =  $\{\sigma \mid (r, \sigma) \in SR, r \notin GR\}$  in
15      if GR =  $\{\}$  then
16         $\lfloor$  analyzeRec( $\theta$ , LT,  $\{\!| \mapsto t \!\}$   $\cup$  H, D,  $\sigma$ , Ex  $\cup$  Exnew)
17      else
18        pick r  $\in$  GR
19        let LTnew =  $\{\!| f(r, l) \mapsto t' \mid (f, t') \in FT,$ 
20           $\forall r', r' \mapsto t' \notin struct \!\}$  in
21        analyzeRec( $\theta$ ,
22           $\sigma_{new}$ (LTnew  $\cup$  LT  $\cup$  H),
23           $\{\} \}$ ,
24           $\sigma_{new}$ ( $\{\!| \mapsto t \!\}$ 
25             $\cup$   $\{\!| r \mapsto k \mid \forall r', r' \mapsto k \notin struct \!\} \cup$  D),
26          unify( $\sigma$ ,  $\sigma_{new}$ ),
27          Ex  $\cup$  Exnew)

```

in the frames. We show that this procedure allows automated verification of (α, β) -privacy goals.

We specify a function *findRelations* that starts by analyzing the frame before trying to find more relations. The analysis of *struct* includes the analyzed frame *struct_{ana}* as well as a unifier and a set of substitutions, excluding some models of the variables. These relations have to be included in the formula ϕ , since it already constitutes some deduction that the intruder was able to make.

First, the intruder tries to compose the terms inside *concr* in different ways. If the intruder has several ways to compose a term, i.e., the composition algorithm returned several recipes, then pairs of recipes from these possibilities must also produce the same corresponding term in *struct*. This gives a number of equations.

Second, the intruder tries to compose the terms inside *struct* in different

ways, under some unifiers. If they are able to compose a term in several ways, then we check whether the pairs of recipes produce the same corresponding term in *concr*. If it is the case, then there is nothing to deduce, as this follows from static equivalence. However, if a pair of recipes distinguishes the frames, i.e., we have found (l, r) such that $\text{concr}\{l\} \not\approx \text{concr}\{r\}$, then the intruder knows that the unifier attached to r can be excluded. They can deduce the negation of the unifier, i.e., a disjunction of inequations.

4.3.1 Pairs from Equivalence Classes

When we want to compare all elements of a set $R = \{r_1, \dots, r_n\}$ for equality, it is obviously sufficient to pick one element, say r_1 , and compare the pairs $(r_1, r_2), \dots, (r_1, r_n)$. The function *pairsEcs* does just that, i.e., given R returns such a set of pairs.

Algorithm 3: Relations between variables

```

1 findRelations( $\theta, \text{struct}$ ) =
2   let ( $\text{struct}_{\text{ana}}, \sigma, Ex$ ) = analyze( $\theta, \text{struct}$ )
3      $\text{concr}_{\text{ana}} = \theta(\text{struct}_{\text{ana}})$ 
4      $\text{pairs} = \bigcup_{l \mapsto t \in \text{concr}_{\text{ana}}} \text{pairsEcs}(\text{compose}(\text{concr}_{\text{ana}}, t))$ 
5      $\text{eqs} = \{\text{struct}_{\text{ana}}\{r_1\} = \text{struct}_{\text{ana}}\{r_2\} \mid (r_1, r_2) \in \text{pairs}\}$ 
6      $\text{ineqs} = Ex \cup \{\sigma' \mid l \mapsto t \in \text{struct}_{\text{ana}},$ 
7        $(r, \sigma') \in \text{composeUnder}(\theta, \text{struct}_{\text{ana}}, t),$ 
8        $\text{concr}_{\text{ana}}\{l\} \not\approx \text{concr}_{\text{ana}}\{r\}\}$  in
9    $\text{unify}(\sigma, \text{eqs}) \wedge \bigwedge_{\sigma' \in \text{ineqs}} \neg \sigma'$ 

```

We formalize the correctness of the decision procedure that has been described. We argue that the algorithm *findRelations* is sound and complete, i.e., the formula ϕ can be used to automatically verify privacy for a message-analysis problem by applying our algorithms. Note that the step of verifying whether ϕ actually excludes models of α can be performed with existing SAT solvers.

Theorem 4 (Correctness of *findRelations*). *Let (α, β) be a message-analysis problem, where $\text{struct} = \{l_1 \mapsto t_1, \dots, l_k \mapsto t_k\}$ for some $t_1, \dots, t_k \in \mathcal{T}_\Sigma(\text{fv}(\alpha))$ and $\text{concr} = \theta(\text{struct})$ for some $\theta \in \Theta$. Let $\phi \equiv \text{findRelations}(\theta, \text{struct})$. Then*

$$(\alpha, \beta)\text{-privacy holds} \iff \forall \theta' \in \Theta, \theta' \models \phi$$

5 Conclusions

We have designed a decision procedure for message-analysis problems in (α, β) -privacy with destructor theories. This procedure is not all that different from algorithms for static equivalence of frames [17]: we split in composition and decryption, have a saturation procedure for decryption, and finally check if we can compose a term in one saturated frame in a different way while the other

frame gives a different result. However, we do not decide static equivalence, rather, one frame, *struct*, has privacy variables, the other, *concr*, is a ground instance of *struct*, and the question is if the intruder can learn something about this instantiation. In particular whatever works in *concr*, must work in *struct*; thus if it works only under some unifier σ , then we rule out all models that are not instances of σ , and vice-versa, if something works in *struct* under σ but not in *concr*, then we rule out all instances of σ .

The fact that the algorithm just returns a substitution that must be the case and a set of substitutions that we can rule out allows for a flexible integration into more complex scenarios. First, we can allow for further variables over finite domains, but that are not part of α . This can be for instance when there are choices that are not themselves relevant for the privacy goals like a session identifier: if the intruder finds them out during analysis, this is not directly a violation of privacy, but if that allows for ruling out some model of α , then it is.

Second, when an agent process can branch on a condition (see for instance the discussion of the AF-protocols in [12]), then the reachable states in general have a form that generalizes message-analysis problems, namely there are several possible frames *struct_i* and associated conditions ϕ_i , and the intruder knows that

$$((\phi_1 \wedge \text{struct}_1 = \text{struct}) \vee \dots \vee (\phi_n \wedge \text{struct}_n = \text{struct})) \wedge \text{struct} \sim \text{concr} .$$

Here, we can apply almost the same algorithms for each *struct_i* with *concr*, except that here we may rule out all models of $\alpha \wedge \phi_i$, meaning we know $\neg\phi_i$.

For future work, we plan to obtain a fully-fledged analysis tool, i.e., exploring the entire set of reachable states, and consider here in particular symbolic representations to avoid exponential blow-ups.

Further, we want to relax the constraints about the algebraic equations. Instead of using only destructor theories, we want to allow for a larger class of protocols to be machine-checked with the framework described, in particular the properties of exponentiation needed for Diffie-Hellman.

Acknowledgments Thanks to Luca Viganò and Sébastien Gondron for useful comments. This work has been supported by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu).

References

- [1] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *15th ACM Conference on Computer and Communications Security*, pages 109–118. ACM, 2008.
- [2] V. Cortier and S. Delaune. A method for proving observational equivalence. In *2009 22nd IEEE Computer Security Foundations Symposium*, pages 266–276. IEEE, 2009.

- [3] D. Bernhard, O. Pereira, B. Smyth, and B. Warinschi. Adapting Helios for provable ballot privacy. In *ESORICS'11: 16th European Symposium on Research in Computer Security, volume 6879 of LNCS*, pages 335–354. Springer, 2011.
- [4] V. Cortier, F. Dupressoir, C. C. Drăgan, B. Schmidt, P. Y. Strub, and B. Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *2017 IEEE Symposium on Security and Privacy*, pages 993–1008. IEEE, 2017.
- [5] M. Moran and D. S. Wallach. Verification of STAR-vote and evaluation of FDR and ProVerif. *Lecture Notes in Computer Science*, 10510:422–436, 2017.
- [6] Boutet, A. et al. Contact Tracing by Giant Data Collectors: Opening Pandora’s Box of Threats to Privacy, Sovereignty and National Security. University works, <https://hal.inria.fr/hal-03116024>, 2020.
- [7] V. Iovino, S. Vaudenay, and M. Vuagnoux. On the effectiveness of time travel to inject COVID-19 alerts. Cryptology ePrint Archive, Report 2020/1393, 2020.
- [8] S. Vaudenay and M. Vuagnoux. Analysis of SwissCovid, 2020. URL: <https://lasec.epfl.ch/people/vaudenay/swisscovid/swisscovid-ana.pdf>.
- [9] D. Basin, J. Dreier, and R. Sasse. Automated symbolic proofs of observational equivalence. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1144–1155. ACM, 2015.
- [10] B. Blanchet. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends in Privacy and Security*, 1(1–2):1–135, 2016.
- [11] T. Groß, S. Mödersheim, and L. Viganò. Defining privacy is supposed to be easy. In *19th 2013 International Conferences on Logic for Programming, Artificial Intelligence and Reasoning*, pages 619–635. Springer, 2013.
- [12] S. Mödersheim and L. Viganò. Alpha-beta privacy. *ACM Trans. Priv. Secur.*, 22(1):1–35, 2019.
- [13] Sébastien Gondron, Sebastian Mödersheim, and Luca Viganò. Privacy as reachability. Technical report, DTU, 2021. <http://www2.compute.dtu.dk/~samo/abg.pdf>.
- [14] M. Genesereth and T. Hinrichs. Herbrand logic. Technical Report LG-2006-02, Stanford University, 2006.
- [15] M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.

- [16] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
- [17] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Lecture Notes in Computer Science*, 3142:46–58, 2004.

A Proofs

Theorem 1 (Correctness of *composeUnder*). *Let θ be a substitution, *struct* be a frame and $t \in \mathcal{T}_\Sigma(\mathcal{V})$. Then*

1. $\forall (r, \sigma) \in \text{composeUnder}(\theta, \text{struct}, t), \sigma(\text{struct}\{r\}) = \sigma(t)$.
2. $\forall r \in \mathcal{R}^c, \exists \tau, \tau(\text{struct}\{r\}) = \tau(t) \implies (\exists \sigma, (r, \sigma) \in \text{composeUnder}(\theta, \text{struct}, t) \text{ and } \tau \models \sigma)$.

Sketch. 1. The idea is to proceed by induction on the structure of t . For the pairs found by comparing with labels or composing a variable, the property holds trivially. For the additional pairs found with terms $f(t_1, \dots, t_n)$ composed with a public function, the point is that the pairs returned for the arguments are correct by induction. The property is then verified for composing t because it reduces to mapping the unifiers returned to all arguments.

2. The idea is to proceed by induction on the structure of $r \in \mathcal{R}^c$. For a label, there is a pair (r, ε) returned so the property holds. For a recipe that is a composition, i.e., $r = f(r_1, \dots, r_n)$ for some f and some $r_1, \dots, r_n \in \mathcal{R}^c$, the point is that the recipes are paired with MGUs by induction. The property is then verified for r because a substitution τ such that $\tau(\text{struct}\{r\}) = \tau(t)$ also unifies the arguments inside the function application, so the algorithm can compute an MGU from the results of the recursive calls.

□

Theorem 2 (Correctness of *analyze*). *Let θ be a substitution, *struct* be a frame and $(\text{struct}_{\text{ana}}, \sigma, Ex) = \text{analyze}(\theta, \text{struct})$. Then*

1. $\forall r \in \mathcal{R}, \text{struct}_{\text{ana}}\{r\} \approx \sigma(\text{struct}\{r\})$.
2. $\forall r \in \mathcal{R}, \exists r' \in \mathcal{R}_{\text{struct}_{\text{ana}}}^c, \text{struct}_{\text{ana}}\{r'\} \approx \sigma(\text{struct}\{r\})$.
3. $\forall \theta' \in \Theta, \theta'(\text{struct}) \sim \theta(\text{struct}) \implies \theta' \models \sigma \wedge \bigwedge_{\sigma' \in Ex} \neg \sigma'$.
4. $\forall \theta' \in \Theta, \theta' \models \sigma \implies (\theta'(\text{struct}) \sim \theta(\text{struct}) \iff \theta'(\text{struct}_{\text{ana}}) \sim \theta(\text{struct}_{\text{ana}}))$

Proof. 1. When analyzing $l \mapsto \text{constr}(t_1, \dots, t_n)$, the frame is augmented with mappings of the form $\text{destr}(r, l) \mapsto t_i$ following the destructor theory. Thus, the “labels” added are recipes over the domain of *struct*. These

shorthands are correct when applying σ , which is required to compose the keys for decryption steps. The frame $struct_{ana}$ is the frame $\sigma(struct)$ with shorthands.

2. We proceed by induction on the structure of r . We consider the occurrence of a destructor \mathbf{destr} such that no subrecipe for the arguments of \mathbf{destr} contains destructors.
 - If the destructor is applied to a label and the decryption is successful, then a shorthand $\mathbf{m} = \mathbf{destr}(r_k, \mathbf{l}) \mapsto t'$ has been added in the frame, i.e., $\sigma(struct\{\mathbf{m}\}) \approx t'$, where r_k is some recipe for the key k such that $\mathbf{destr}(k, t) = t' \in E$.
 - If the destructor is applied to a constructor, i.e., for some r_k, r_1, \dots, r_n , $r = \mathbf{destr}(r_k, \mathbf{constr}(r_1, \dots, r_n))$, and the decryption is successful, then the recipe can be simplified to one of the r_i yielding the same term.
 - If the decryption is not successful, then we can replace the application of \mathbf{destr} by the constant \mathbf{error} , which represents failed decryption

We have covered all cases since the subrecipes do not contain destructors. By induction, we can replace all occurrences of destructors in the recipe, i.e., we can define a constructive recipe r' which is the same as r but all occurrences of destructors and have been replaced by the methods listed above.

3. We first show that the intruder can exclude all models that are not instances of σ . The substitution σ has been built from unification of some σ_i in successful analysis steps, i.e., where $(r_i, \sigma_i) \in \mathit{composeUnder}(\theta, struct, k)$ was a possibility to compose a decryption key k , and $r_i \in \mathit{compose}(\theta(struct), \theta(k))$ is also a recipe for the corresponding key $\theta(k)$ in $\theta(struct)$. It suffices to show that $\theta' \models \sigma_i$ for all σ_i . From Theorem 1 follows that σ_i is the MGU under which k can be derived in θ , i.e., $\theta'(struct\{r_i\}) \not\approx \theta'(k)$ for any θ' that is not an instance of σ_i . Since the intruder can see that r_i produces the correct decryption key in $\theta(struct)$, all models that are not consistent with σ_i can be excluded.

We next show that all models that are instances of a substitution $\sigma' \in Ex$ can be excluded by the intruder as well. The substitution σ' has been found during analysis of some mapping $\mathbf{l} \mapsto t$ where the key k can be composed in the current $struct$ under some unifier but $\theta(k)$ cannot be composed in $\theta(struct)$. There exists $(r_k, \sigma') \in \mathit{composeUnder}(\theta, struct, k)$ for some recipe r_k . There is a destructor \mathbf{destr} for the decryption under consideration. We define the recipe $r = \mathbf{destr}(r_k, \mathbf{l})$ for this decryption step. The decryption fails in $\theta(struct)$, so $\theta(struct\{r\}) \approx \theta(struct\{\mathbf{error}\})$. Since $\theta'(struct) \sim \theta(struct)$, we also have that $\theta'(struct\{r\}) \approx \theta'(struct\{\mathbf{error}\})$. However, the decryption is successful in $struct$, so $\sigma'(struct\{r\}) \not\approx \sigma'(struct\{\mathbf{error}\})$. Therefore, θ' is not an instance of σ' , because if it were there would be a pair of recipes, namely (r, \mathbf{error}) , to distinguish the frames.

4. Let $\theta' \in \Theta$ such that $\theta' \models \sigma$. Using property 1. and the fact that $\theta' \models \sigma$, we have that for any recipe r , $\theta'(struct_{ana}\{r\}) \approx \theta'(struct\{r\})$. This also holds in particular for θ . Therefore, $\theta'(struct) \sim \theta(struct)$ if and only if $\theta'(struct_{ana}) \sim \theta(struct_{ana})$ because any pair of recipes distinguishing $\theta'(struct)$ and $\theta(struct)$ would also distinguish the analyzed frames, and vice-versa. □

Theorem 3 (Termination of *analyze*). *Let θ be a substitution and $struct$ be a frame. Then the call $analyze(\theta, struct)$ terminates.*

Proof. By definition, *analyze* calls *analyzeRec*, so what we really want to show is that the call to *analyzeRec* terminates. We now consider that the frame *struct* has been split into three frames N, H, D and denote with σ and Ex the unifier and the set of substitutions passed as arguments to *analyzeRec*, respectively. The size of a term $t \in \mathcal{T}_\Sigma(\mathcal{V})$ is defined as 1 for a variable and $size(f(t_1, \dots, t_n)) = 1 + \sum_{i=1}^n size(t_i)$ for a function application. We abuse the notation and write $size(N \cup H)$ to mean the sum of the size of all terms in $N \cup H$. We consider the tuple $(size(N \cup H), \#N)$. When analyzing the mapping $l \mapsto t \in N$:

- If the decryption of t fails, $l \mapsto t$ is removed from N and put in H . Then $size(N \cup H)$ stays the same but $\#N$ has decreased by 1.
- If the decryption of t succeeds, $l \mapsto t$ is removed from N and put in D . The new terms from the analysis and the terms that were on hold are put in N . Then $size(N \cup H)$ has decreased by at least 1 (t is not present anymore but some of its subterms might be).

The lexicographic order on $(\mathbb{N}, \leq) \times (\mathbb{N}, \leq)$ forms a well-order and the sequence of tuples for the recursive calls is a strictly decreasing sequence bounded by $(0, 0)$, so such a sequence is finite and the call terminates. □

Theorem 4 (Correctness of *findRelations*). *Let (α, β) be a message-analysis problem, where $struct = \{l_1 \mapsto t_1, \dots, l_k \mapsto t_k\}$ for some $t_1, \dots, t_k \in \mathcal{T}_\Sigma(fv(\alpha))$ and $concr = \theta(struct)$ for some $\theta \in \Theta$. Let $\phi \equiv findRelations(\theta, struct)$. Then*

$$(\alpha, \beta)\text{-privacy holds} \iff \forall \theta' \in \Theta, \theta' \models \phi$$

Proof. Let $(struct_{ana}, \sigma, Ex) = analyze(\theta, struct)$. First, recall that we have (α, β) -privacy holds $\iff \forall \theta' \in \Theta, \theta'(struct) \sim \theta(struct)$. We show that $\forall \theta' \in \Theta, \theta'(struct) \sim \theta(struct) \iff \theta' \models \phi$. The models that are not instances of σ can already be excluded and violate the privacy of α because $\phi \models \sigma$. We now consider $\theta' \in \Theta$ such that $\theta' \models \sigma$.

- If $\theta'(struct) \not\sim \theta(struct)$: then $\theta'(struct_{ana}) \not\sim \theta(struct_{ana})$ from Theorem 2, so there exists a pair of recipes (r_1, r_2) that distinguishes the frames. From Theorem 2, we can assume without loss of generality that r_1, r_2 are constructive. Moreover, either one the recipes is a label (or from

a shorthand) or both recipes have the same constructor at the top-level and one pair of the recipes for the arguments distinguishes the frames. So we can further assume that r_1 is a label (or from a shorthand). This justifies the fact that *findRelations* will perform a check for this pair of recipes.

- If $\theta'(struct_{ana}\{r_1\}) \not\approx \theta'(struct_{ana}\{r_2\})$ and for the concrete observation $\theta(struct_{ana}\{r_1\}) \approx \theta(struct_{ana}\{r_2\})$: then θ' cannot be an instance of the substitution σ unifying, among others, the following equation: $struct_{ana}\{r_1\} = struct_{ana}\{r_2\}$. The algorithm returns ϕ such that $\phi \models \sigma$, so $\theta' \not\models \phi$.
- If $\theta'(struct_{ana}\{r_1\}) \approx \theta'(struct_{ana}\{r_2\})$ and for the concrete observation $\theta(struct_{ana}\{r_1\}) \not\approx \theta(struct_{ana}\{r_2\})$: then θ' is an instance of some substitution σ' found when checking inequations. The algorithm returns ϕ such that $\phi \models \neg\sigma'$, so $\theta' \not\models \phi$.
- If $\theta'(struct) \sim \theta(struct)$: then $\theta'(struct_{ana}) \sim \theta(struct_{ana})$ from Theorem 2. For every $t \in \mathcal{T}_\Sigma$ and $(r_1, r_2) \in pairsEcs(compose(\theta(struct_{ana}), t))$, we have by definition of *compose* that $\theta(struct_{ana}\{r_1\}) \approx \theta(struct_{ana}\{r_2\})$. Since $\theta'(struct_{ana}) \sim \theta(struct_{ana})$, then $\theta'(struct_{ana}\{r_1\}) \approx \theta'(struct_{ana}\{r_2\})$. Therefore, $\theta' \models \sigma$, where σ unifies all equations found from calling *compose* on terms in $\theta(struct_{ana})$. Let *ineqs* be the set of substitutions *Ex* found during analysis union with the substitutions found by the *findRelations* algorithm. If θ' were an instance of some $\sigma' \in ineqs$, then $\theta'(struct_{ana}) \not\sim \theta(struct_{ana})$ and thus $\theta'(struct) \not\sim \theta(struct)$ following Theorem 2. This would contradict the assumption, so $\theta' \models \neg\sigma'$. Therefore, $\theta' \models \sigma \wedge \bigwedge_{\sigma' \in ineqs} \neg\sigma'$ which is exactly $\theta' \models \phi$.

□