A Sound Abstraction of the Parsing Problem (Extended Version)

DTU Technical Report DTU-Compute-12-2014

Sebastian Mödersheim
DTU Compute
Technical University of Denmark
DK-2800 Lyngby
Email: samo@dtu.dk

Georgios Katsoris
DTU Compute
Technical University of Denmark
DK-2800 Lyngby
Email: georgios.katsoris@gmail.com

***Abstract*—In formal verification, cryptographic messages are often represented by algebraic terms. This abstracts not only from the intricate details of the real cryptography, but also from the details of the non-cryptographic aspects: the actual formatting and structuring of messages.**

We introduce a new algebraic model to include these details and define a small, simple language to precisely describe message formats. We support fixed-length fields, variable-length fields with offsets, tags, and encodings into smaller alphabets like Base64, thereby covering both classical formats as in TLS and modern XML-based formats.

We define two reasonable properties for a set of formats used in a protocol suite. First, each format should be un-ambiguous: any string can be parsed in at most one way. Second, the formats should be pairwise disjoint: a string can be parsed as at most one of the formats. We show how to easily establish these properties for many practical formats.

By replacing the formats with free function symbols we obtain an abstract model that is compatible with all existing verification tools. We prove that the abstraction is sound for un-ambiguous, disjoint formats: there is an attack in the concrete message model if there is one in the abstract message model. Finally we present highlights of a practical case study on TLS.

***Keywords*-Security protocols, formal verification, message formats, soundness, compositional reasoning**

I. Introduction

Formal verification approaches have proved to be successful in verifying security properties of distributed systems that exchange cryptographic messages: security protocols, web services, or Crypto-APIs. Most approaches use a Dolev-Yao-style model, representing cryptographic messages as terms in a free term algebra (sometimes modulo some equations). Here constant symbols represent "atomic" messages like identifiers or (atomic) keys, and function symbols represent cryptographic operators. One thus treats the cryptography like black boxes, where the intruder can encrypt and decrypt messages only when knowing the respective keys.

The reason for the success lies in the relative simplicity of these abstract models, which allows for efficient automated verification tools, compositional reasoning or refinement approaches. Some computational soundness results exist; they show that (under certain conditions) such an abstraction of the cryptography is actually sound [1], [2].

We focus here on an aspect that has received much less attention: the non-cryptographic operators. This is the question of how messages are formatted and structured. Abstract approaches usually use an operator $\mathsf{cat}(t_1, t_2)$ to denote the concatenation of two messages t_1 and t_2. This completely ignores how an actual implementation structures this information, so that it is later possible to *parse* a given string and extract a t_1 and a t_2 part.

An example where these details of message formats can give rise to vulnerabilities are type-flaw attacks, where an attacker exploits the similarity of two formats to make a recipient accept a message in a different context (and meaning) than the sender meant it. This is particularly relevant when we use the same long-term keys for different protocols that are deployed in parallel [3]. Another example are injection attacks: if an intruder-chosen string is filled into a message schema, this may break the (intended) structure of the schema.

These vulnerabilities on the non-cryptographic level are the low-hanging fruit as attacks seem much more common and successful than attacks on the cryptography. Our aim is however not to "find more attacks" (that are quite obvious anyway when one is aware of the problem). Rather, we give a general soundness result for a large class of message formats: for these, the verification in abstract term models is actually sound.

A. Contributions

We consider in this paper two paradigms for structuring messages that cover many practically relevant protocols. The first may be called the data-structure paradigm and is used for instance in TLS: here we have a concatenation of fields that can be either of fixed or of variable length. A variable-length field starts with an *offset* that tells the length of the rest of the field. This offset itself is of fixed length, e.g. when using a two-byte offset, the field can be at most 65535 bytes long. The second paradigm we consider are XML-style messages. Here we can easily structure messages

by tags marking the begin and end of elements. To avoid collision of the actual data with the XML-symbols, one uses encodings into smaller alphabets such as hexadecimal encoding or Base64.

We first give a simple language to describe such message formats with a fixed number of fields. We give a simple sufficient condition for a format to be un-ambiguous (a string cannot be parsed in two different ways). We give a parser for un-ambiguous formats. We then give also a checking procedure that implements a sufficient check for two formats to be disjoint, i.e., no string can be parsed as both formats. Requiring that formats are un-ambiguous and pairwise disjoint are reasonable conditions: it must be impossible to parse any string in more than one way, neither within one format or as two different formats.

Based on this, we define an algebra that models precisely the behavior of the non-cryptographic operators, but abstracts from the cryptographic ones since our focus is not a computational soundness result. For this algebra we use a novel way of modeling. We first define a class of *cryptographic* algebras with byte-strings as the universe and where operators are interpreted as functions on these byte strings, modeling exactly the real world implementations. The behavior of the non-cryptographic operators like concatenation is fixed in the obvious way, while the implementation of the cryptographic operators is arbitrary. We then abstract from the real-world cryptography by defining a congruence relation $t_1 \approx t_2$ that holds for two terms t_1 and t_2 if they are equal in *all* cryptographic algebras. In this way we get an $\approx$ relation that combines the details of message structuring while abstracting from the cryptography by selecting those equations that hold regardless of the implementation of the cryptography.

The resulting algebra $\mathbb{D}$, the term algebra modulo $\approx$, is too complex to work with automated verification, especially due to its inherent semantic nature. The main result is now:

- if we consider a system where all participants except the intruder use only formats for structuring messages (i.e., instead of working with low-level operators like concatenation),
- if all the used formats are un-ambiguous and pairwise disjoint, and
- if the system has an attack (when interpreting terms in algebra $\mathbb{D}$)
- then there is also an attack in a free algebra model (that does not know the low-level operators).

This conclusion means it is sound to use any of the common verification tools like ProVerif [4] or AVANTSSAR [5].

Finally, we also illustrate with a case study on the TLS protocol, how our approach can be practically used on complex, real-world examples.

B. Structure of the Paper

In Section II we define the class of crypto algebras. In Section III we define formats, how to parse them and the sufficient check for disjointness. In Section IV we define our concrete algebra $\mathbb{D}$ and prove that the sub-algebra $\mathbb{D}'$ without low-level operators behaves like a free algebra. In Section V we define a Dolev-Yao style model based on $\mathbb{D}$ as well as constraint systems over this model. In Section VI we give the main soundness result. In Section VII we discuss a problem of lexical analysis. In Section VIII we present the highlights of the TLS case study. Finally, in Section IX we discuss related work and conclude.

II. Towards A New Term Model

In this section we define a class of "crypto algebras" that interpret function symbols as real cryptographic and non-cryptographic operations on strings. This is the basis for the novel term model of Section IV that abstracts the cryptographic operators, but keeps all the details of non-cryptographic aspects. For this model we then prove our abstraction result. All algebras are based on the same signature that we define first.

A. Signature

We fix a signature Σ. It contains a countable set $\Sigma_0 = \{c_1, c_2, \ldots\} \subseteq \Sigma$ of constants. We distinguish two kinds of constants: a finite set of *literals* $\mathbb{L}$ and a countably infinite set of *uninterpreted constants* $\mathbb{U}$. The literals are strings like XML-tags (that will later literally represent that string), while uninterpreted constants like n_7 may represent a fresh nonce and have no fixed interpretation as a string. The literals include the constant ϵ that denotes the string of length 0. We will later assume that the intruder initially knows all literals.

Next, the signature contains black-box operations on messages.

- $\mathsf{scrypt}(k, m, r)$ for symmetric encryption of clear-text m with symmetric key k, and randomization value r. Usually, we will omit r in the notation since it is simply a randomly chosen value by whoever performs the encryption to avoid deterministic encryption.
- $\mathsf{crypt}(k, m, r)$ similarly is asymmetric encryption where k is a public key.
- $\mathsf{sign}(k, m, r)$ similarly is asymmetric encryption where k is a private key.
- For public/private key pairs, we assume that they are always created from some seed value s by the functions $\mathsf{pub}(s)$ and $\mathsf{priv}(s)$.
- $\mathsf{h}(m)$ for the cryptographic hash of a message m.
- $\mathsf{mac}(k, m)$ for a key-ed hash of m using symmetric key k.
- $\mathsf{cat}(m_1, m_2)$ for concatenation of messages m_1 and m_2. As we will define below, $\mathsf{cat}(\cdot, \cdot)$ is associative and we may thus simply write $m_1 \cdot m_2 \cdot \ldots \cdot m_k$

for $\mathsf{cat}(m_1, \mathsf{cat}(m_2, \mathsf{cat}(\ldots, m_k)))$ and even omit the "multiplication" operator $\cdot$ when clear from the context.
- $\mathsf{enc}(m)$ an encoding function for mapping into a smaller alphabet, e.g., a Base64 encoding. For simplicity, we assume here that only one such encoding is used, but our results can easily be extended to several such encodings.
- $\mathsf{off}_k(m)$ to "compute the offset" of m: it yields a k-byte string representing the length of m. (For this to work, m cannot be arbitrary large; we define the precise behavior later.)
- We will in Section III augment the signature Σ with a set of function symbols $\mathsf{form}_1(\cdot), \ldots, \mathsf{form}_n(\cdot)$ to represent the clear-text structure more abstractly than with low-level operators like cat.

We build *ground terms* from this signature as expected; the set of ground terms is denoted $\mathcal{T}_\Sigma$. Given a set V of variable symbols (disjoint from Σ) we denote with $\mathcal{T}_\Sigma(V)$ the set of terms containing also variables of V. Denote with $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ the substitution of variables x_i to terms t_i where no variable x_i can occur in any term t_j. We extend substitutions to functions on terms as expected.

Let us also say that a ground term is *abstract* if it does not contain any of operators $\mathsf{cat}(\cdot,\cdot)$, $\mathsf{enc}(\cdot)$, and $\mathsf{off}.(\cdot)$. In fact, the main theorem below will show that we may safely avoid reasoning about any of these "low-level clear-text operators" and use only abstract terms.

We have so far only described the signature of the terms. We now attach a meaning to each symbol in a very concrete way as functions on strings.

B. Crypto Algebras

Let $\mathbb{B}^*$ denote the set of all byte-strings. We now consider a *class* of algebras that use $\mathbb{B}^*$ as the universe and interpret every function symbols $f \in \Sigma$ with arity n as a function $f^{\mathcal{C}} : (\mathbb{B}^*)^n \to \mathbb{B}^*$. Thus each $f^{\mathcal{C}}$ represents the meaning of that symbol on actual messages (note that for every constant $c \in \Sigma_0$, $c^{\mathcal{C}}$ is simply a byte-string in $\mathbb{B}^*$).

We later define a common black-box intruder model for the cryptography, e.g., the intruder can decrypt $\mathsf{scrypt}(k, m)$ to derive m iff he can derive k from his knowledge; vice-versa he can only produce the encryption $\mathsf{scrypt}(k, m)$ when he knows both k and m. We do not make any formal link between the abstract symbol scrypt and the concrete encryption function $\mathsf{scrypt}^{\mathcal{C}}$ (as it is done in *cryptographic* soundness results). In fact, we actually do not exclude "absurd" models of cryptography like $\mathsf{scrypt}^{\mathcal{C}}(k, m) = m$. (However, in such models the black-box intruder model may not properly reflect the properties of the real implementation.)

We do however make some requirements about the $f^{\mathcal{C}}$ functions that implement the non-cryptographic operators, e.g., that $\mathsf{cat}^{\mathcal{C}}$ is really string concatenation. This will be made precise in Definition 1.

We fix a set of variables $\mathcal{V}$ (disjoint from Σ). For every variable $x \in V$, we also fix a set $|x|$ to be an arbitrary, non-empty subset of the natural numbers. This represents the allowed length of the strings that can be substituted for x, e.g., $|x| = \mathbb{N}$ when x can hold strings of any length. Similarly, for all constants $c \in \Sigma_0$ we fix $|c| = \{l\}$ to be a singleton set. Here ϵ is the only constant of length $\{0\}$. For strings $str \in \mathbb{B}^*$ we define $|str| = l$ to be the length in bytes as is standard.

Definition 1 (Crypto Algebra): Let $\mathcal{C}$ be a Σ-algebra, and let $|\cdot|_{\mathcal{C}} : \mathcal{T}_\Sigma(\mathcal{V}) \to \mathbb{P}(\mathbb{N})$ be a length function. We say $(\mathcal{C}, |\cdot|_{\mathcal{C}})$ is a *crypto algebra* iff the following holds:
- The universe of $\mathcal{C}$ is $\mathbb{B}^*$.
- Thus, every function symbol $f \in \Sigma$ of arity n is interpreted in $\mathcal{C}$ as a function $f^{\mathcal{C}} : (\mathbb{B}^*)^n \to \mathbb{B}^*$. We write $t^{\mathcal{C}}$ for the interpretation of a ground term t in $\mathcal{C}$, i.e., $f(t_1, \ldots, t_n)^{\mathcal{C}} = f^{\mathcal{C}}(t_1^{\mathcal{C}}, \ldots, t_n^{\mathcal{C}})$.
- Recall that the literals $\mathbb{L} \subseteq \Sigma_0$ are verbatim strings. For each such literal s, let $s^{\mathcal{C}} = s$.
- For all variables and constants $t \in \mathcal{V} \cup \Sigma_0$, $|t| = |t|_{\mathcal{C}}$.
- The length of a ground term is a singleton set, namely the length in bytes of its interpretation: $|t|_{\mathcal{C}} = \{\ |t^{\mathcal{C}}|\ \}$.
- For general terms (that are not necessarily ground) we define $[\![\cdot]\!]_{\mathcal{C}} : \mathcal{T}_\Sigma(\mathcal{V}) \to \mathcal{P}(\mathbb{B}^*)$ as follows:

$$\begin{aligned} [\![x]\!]_{\mathcal{C}} &= \textstyle\bigcup_{l \in |x|} \mathbb{B}^l \\ [\![f(t_1, \ldots, t_n)]\!]_{\mathcal{C}} &= \{ f^{\mathcal{C}}(s_1, \ldots, s_n) \mid \\ &\qquad s_1 \in [\![t_1]\!]_{\mathcal{C}} \wedge \ldots \wedge s_n \in [\![t_n]\!]_{\mathcal{C}} \} \end{aligned}$$

  Note that for every ground term t, $[\![t]\!]_{\mathcal{C}} = \{t^{\mathcal{C}}\}$.
- We require that $|t|_C = \{|s| \mid s \in [\![t]\!]_{\mathcal{C}}\}$
- The interpretation of cat is string concatenation, i.e., $\mathsf{cat}^{\mathcal{C}}(s_1, s_2) = s_1 \cdot s_2$.
- We require that $\mathsf{off}^{\mathcal{C}}$ indeed yields the length of a given string. Formally, for strings s, s' and length $k \in \mathbb{N}$, from $\mathsf{off}_k^{\mathcal{C}}(s) = s'$ follows $|s'| = k$ and if $|s| < 256^k$ then s' is a k-byte representation of $|s|$. (One must fix either big-endian or little-endian representation here. If $|s| \geq 256^k$, then the length is not representable and the implementation may choose any k-byte value to return.)
- Finally, we require that $\mathsf{enc}(\cdot)$ gives an encoding into a smaller alphabet $\mathbb{X} \subsetneq \mathbb{B}$: we require that $\mathsf{enc} : \mathbb{B}^* \to \mathbb{X}^*$ is a bijective (1-to-1) mapping. Examples would be hexadecimal or Base64. The model indeed reflects that the hexadecimal encoding of a string s has length $2|s|$, while the Base64-encoding has length $4\ \lceil |s|/3 \rceil$. For simplicity, we do not consider more than one encoding, but all results can be extended accordingly. ■

For most of this paper (unless where noted otherwise), we will consider a fixed crypto algebra $(\mathcal{C}, |\cdot|_{\mathcal{C}})$ and omit $\cdot_{\mathcal{C}}$ subscripts when no confusion arises. We may also refer to $\mathcal{C}$ alone as a crypto algebra, leaving implicit that it has an associated length function $|\cdot|_{\mathcal{C}}$. We conclude this section with the concept of a string-substitution θ that is a mapping

from variables to strings such that $|\theta(x)| \in |x|$. We extend θ to a mapping from terms to strings by $\theta(f(t_1,\ldots,t_n)) = f^{\mathcal{C}}(\theta(t_1),\ldots,\theta(t_n))$.

III. Formats

In order to formalize the abstraction of the concrete message structures, we now introduce symbols $\mathsf{form}_1(\cdot),\ldots,\mathsf{form}_n(\cdot)$ to represent the different formats abstractly. We define a small simple language to define formats.

Definition 2: A *format declaration* for an operator form has the shape:

$$\mathsf{form}(x_1,\ldots,x_n) = \mathit{fld}_1 \cdot \mathit{fld}_2 \ldots \mathit{fld}_m$$

Here, the x_i are variables (assuming we have again fixed a set $|x_i|$ of possible lengths for each variable x_i). The fld_i are called *fields* and each field can be any of the following:

- a literal constant $c \in \mathbb{L} \setminus \{\epsilon\}$,
- a variable x_i $(1 \le i \le n)$,
- an offset construction $\mathsf{off}_k(x_i) \cdot x_i$ where $k \in \mathbb{N}$, $1 \le i \le n$, and $|x| \subseteq \{0,\ldots,256^k - 1\}$. This means that the length of the value x is literally written as a k-byte field, followed by the value x itself,
- an encoding $\mathsf{enc}(x_i)$ of a variable $(1 \le i \le n)$.

For simplicity, we assume that forms are *linear* in the sense that none of the x_i occurs in more than one field.[1]

For a format declaration $\mathsf{form}_i(x_1,\ldots,x_n) = \ldots$ the *admissible length* of the j-th argument is $|x_j|$. We thus say that the term $\mathsf{form}_i(t_1,\ldots,t_n)$ is *legal*, if $|t_i| \subseteq |x_i|$. For a term t of arbitrary shape we say that t is legal if every $\mathsf{form}_{\_}(\cdot)$-subterm of t is legal. We generally forbid illegal terms in the rest of this paper, i.e., we restrict $\mathcal{T}_\Sigma(\mathcal{V})$ to the subset of legal terms (for given length definitions of variables and functions) without introducing a new symbol. ■

The rationale for excluding illegal terms is that terms simply cannot be filled into a form when they do not have an appropriate size. A buffer overflow is a typical example of a flawed implementation that does not check the size restriction of fields. The restriction to legal terms thus excludes such flawed implementations. This is in fact not a restriction on the intruder abilities we define below, since we allow him arbitrary use of low-level functions like cat.

The definition of formats is specific to the protocol or protocol suite that one wants to consider. Given a definition of a set of formats, we extend the crypto algebras of Definition 1 as follows.

For every declaration $\mathsf{form}_i(x_1,\ldots,x_n) = \mathit{fld}_1 \cdot \ldots \cdot \mathit{fld}_m$ and every $s_1 \in [\![x_1]\!]_{\mathcal{C}},\ldots,s_n \in [\![x_n]\!]_{\mathcal{C}}$, we set

$$\mathsf{form}^C(s_1,\ldots,s_n) = [x_1 \mapsto s_1,\ldots,x_n \mapsto s_n](\mathit{fld}_1 \cdot \ldots \cdot \mathit{fld}_m).$$

[1]The extension to non-linear patterns is straightforward: the parser just additionally needs to check that the respective substrings are equal.

Example 1: Recall that we may omit $\mathsf{cat}(\cdot,\cdot)$; let $|x| = \{16\}$, $|y| = \{0,\ldots,65535\}$ and $|z| = \mathbb{N}$:

$$\begin{aligned}
\mathsf{form}_1(x,y,z) &= \mathsf{myform}\ x\ \mathsf{off}_2(y)\ y\ z \\
\mathsf{form}_2(x,z) &= \langle \mathsf{myxform} \rangle \\
&\qquad \langle \mathsf{nonce} \rangle \mathsf{enc}(x) \langle / \mathsf{nonce} \rangle \\
&\qquad \langle \mathsf{name} \rangle \mathsf{enc}(z) \langle / \mathsf{name} \rangle \\
&\quad \langle / \mathsf{myxform} \rangle
\end{aligned}$$

From the definition of $[\![\cdot]\!]$ follows how to construct a parser for form_1 and form_2. For form_1, we first require that the string starts with the *tag* myform. Then the next 16 bytes are parsed as x. Then we read two bytes, telling us the length l of y; then we read y to be the next l bytes. Finally, all the rest of the message is read as z. If the string does not start with the tag, or we reach the end of the string while more is expected, the parsing fails and the format cannot be of kind form_1.

For form_2, first note that our definition of formats does not include whitespaces, and they would lead to confusion in formats like form_1, while XML *does* allow whitespaces between all tokens. However, for machine-generated XML-messages whitespaces are redundant and can be omitted.[2] Since the technical presentation in this paper is already complicated enough, we work here with XML-formats without whitespaces, and only briefly discuss in Section VII how to extend our result to allow whitespaces. We assume that the alphabet $\mathbb{X}$ (that $\mathsf{enc}(\cdot)$ encodes to) does not contain any symbols with syntactical meaning in XML such as the angle brackets and the slash. Thus, parsing requires the XML tags $\langle \mathsf{myxform} \rangle$ and $\langle \mathsf{nonce} \rangle$. Then we read the longest string that contains only characters of the alphabet $\mathbb{X}$. This is in fact the only way to parse a string since said alphabet cannot contain the next character after $\mathsf{enc}(x)$, namely $\langle$. The string read of $\mathsf{enc}(x)$ is then appropriately decoded and assigned to variable x. Again, the parsing fails if x does not have length 16 as specified by the format. The rest of parsing this format is of course similar. ■

The examples indicate that format description can be used to systematically derive parser implementations that enforce specified restrictions. An immediate question is whether a given format is un-ambiguous. For instance if $|x| = |y| = \mathbb{N}$, then format $\mathsf{form}(x,y) = x\ y$ is ambiguous—as we define it next.

A. Parsing and Ambiguity

We define for a format $\mathsf{form}_i(x_1,\ldots,x_n)$ that a string s *can be parsed for* form_i *as* θ iff

- θ is a string-substitution with domain $\{x_1,\ldots,x_n\}$,
- $|\theta(x_i)| \in |x_i|$,
- $\theta(\mathsf{form}_i(x_1,\ldots,x_n)) = \theta(\mathit{fld}_1 \cdot \ldots \cdot \mathit{fld}_n) = s$.

[2]There is one exception: between tags and attributes, at least one whitespace is required, e.g., $\langle$a href = "..."$\rangle$ requires a whitespace between a and href. In this case we could define a standard, e.g., one space.

Define that a format $\mathsf{form}_i(x_1, \ldots, x_n)$ is *ambiguous* iff there is a string s that can be parsed in two different ways, i.e., as string substitutions θ_1 and θ_2 such that $\theta_1(x_i) \neq \theta_2(x_i)$ for some $1 \leq i \leq n$.

In the example $\mathsf{form}(x, y) = x\ y$ where $|x| = |y| = \mathbb{N}$ for instance the single character c can be parsed either as $\theta_1 = [x \mapsto \mathsf{c}, y \mapsto \epsilon]$ or as $\theta_2 = [x \mapsto \epsilon, y \mapsto \mathsf{c}]$.

It is easy to generally exclude such ambiguities: an item that has variable length shall either be sent with an offset construction, or as the last element of a format ("taking the remainder of the message"), or under the encoding $\mathsf{enc}(x)$ followed by a character in $\mathbb{B} \setminus \mathbb{X}$.

Definition 3: A form declaration $\mathsf{form}_i(x_1, \ldots, x_n) = \mathit{fld}_1 \cdot \ldots \cdot \mathit{fld}_m$ is said to have *clear boundaries* if for every field fld_i with $i < m$ holds:

- $\mathit{fld}_i = x_j$ implies
  - $|x_j| = \{l\}$ (i.e., x_j has a fixed length)
  - or $i > 1$ and $\mathit{fld}_{i-1} = \mathsf{off}_k(x_j)$ (i.e., x_j has variable length with an offset construction).
- $\mathit{fld}_i = \mathit{enc}(x_j)$ implies that for every $s \in [\![\mathit{fld}_{i+1}]\!]$, s is not the empty string and starts with a letter in $\mathbb{B} \setminus \mathbb{X}$.

For formats with clear boundaries we can easily write a parser that is given as argument a list of fields and a string (as a list of characters)–see Fig. 1. This algorithm in Pseudo-Haskell code uses functions

- `head l` and `tail l` that return the first element and the rest list `l`, respectively. They return error when `l=[]` (the empty list).
- `take n l` takes the first `n` elements of list `l`, similarly `drop n l` removes the first `n` elements. Both return an error when the list has less than `n` elements.
- `takeWhile p l` returns the longest prefix of `l` such that each element satisfies `p`, and `dropWhile` is the counter-part.
- `toNum s` gives the integer represented by string `s`.
- `elemX` checks whether a element belongs to set $\mathbb{X}$.
- We write `[x|->t]` and the like for the substitutions created. Finally, we assume that in building substitutions we check conformity with the length, i.e., that $|t| \subseteq |x|$.

Theorem 1: Let $\mathsf{form}_i(x_1, \ldots, x_n) = \mathit{fld}_1 \cdot \ldots \cdot \mathit{fld}_m$ be a format with clear boundaries. Given a string s, there is at most one string substitution θ as which s can be parsed for form form_i. If such a θ exists, then $\mathit{parse}\ [\mathit{fld}_1, \ldots, \mathit{fld}_m]\ s$ will return θ and produce an error otherwise.

Proof: It is straightforward that the above algorithm is sound. Also it is obvious that for all cases that involve fields with fixed lengths and offset construction, there is no other choice to parse the string. If it is any other variable, then by clear boundaries it must be the last field of the format, so again there is just one choice to parse, namely setting the variable to the remaining string. The only other case is an encoded field $\mathsf{enc}(x)$. By clear boundaries, the next field (if it exists) starts with a letter that is not n $\mathbb{X}$. Thus, we parse until we hit the first element that is not in $\mathbb{X}$; this yields the longest prefix of s that we can possibly parse as $\mathsf{enc}(x)$. It is also the shortest, because otherwise the remainder of s starts with a letter in $\mathbb{X}$ that cannot be parsed as the next field in the recursive call. ∎

B. Disjoint Formats

Another aspect that is later relevant for abstracting is whether two formats are actually sufficiently different so that confusions are excluded. We want to exclude that one honest agent produces a message $\mathsf{form}_1(t_1, \ldots, t_n)$ and another who receives this message accidentally parses it as $\mathsf{form}_2(t'_1, \ldots, t'_m)$. Note that unencrypted forms are of course not protected against manipulations by the intruder (e.g., he may replace tags) but one given message should have an "un-ambiguous meaning" in the sense that it can be parsed in only one way. Formally, we say two formats $\mathsf{form}_i(x_1, \ldots, x_n)$ and $\mathsf{form}_j(y_1, \ldots, y_m)$ are *disjoint* iff $[\![\mathsf{form}_i(x_1, \ldots, x_n)]\!] \cap [\![\mathsf{form}_j(y_1, \ldots, y_m)]\!] = \emptyset$. (Recall that in case of variables, the semantics considers the set of byte strings that are allowed by the declared variable lengths.)

In general, deciding whether two formats are disjoint is difficult, for instance through the offsets we get a complicated relation between lengths of one message and the content of another. However, in practice it is often quite easy, for instance if the formats start with distinct tags we are already done. We give here a simple procedure for checking disjointness that uses over-approximation in difficult cases. For this over-approximation the procedure is a sufficient but not necessary for disjointness: when the procedure answers `True`, then the given formats are indeed disjoint.

The algorithm $\mathit{disjoint}(F_1, F_2)$ in Fig. 2 receives two formats as arguments, which are again given as lists of fields. We make again several simplifications to the presentation:

- In the call $\mathit{disjoint}(F_1, F_2)$ we make a case-distinction on the shape of F_1 and of F_2. By symmetry of disjointness, we would have to write many almost identical cases that differ only in the order of the F_i. We then write only one case.
- Again, we assume that patterns are linear, so we do not need to compare the value of different fields.
- We assume literal constants are only of length 1 (as longer constants can be broken down).
- Since $[\![\mathsf{off}_k(x) \cdot x]\!] \subseteq \mathbb{B}^*$, we can simplify the procedure by over-approximating all offset constructions with variables of unbounded length. Similarly, we over-approximate all variables that have more than one length with arbitrary length. Thus we have only variables of fixed length l, which we denote as $x[l]$ in the algorithm, and variables of unbounded length which we denote $x[*]$.

It may seem strange that after all the care we have spent on the details of modeling variables with an offset con-

```
parse [] s = if s=="" then [] else error
parse (t:ts) s =
  case t of
  c -> -- constant (for simplicity assume constant of length one)
       if (head s)==c then parse ts (tail s) else error
  x -> if (length x)==[l]  -- fixed length l
       then [ x |-> take l s]++(parse ts (drop l s))
       else [ x |-> s ] -- in this case ts is empty by def. clear boundaries
  off_k(x) -> -- by construction (head ts)==x
                let n=toNum (take k s) -- offset
                    s'=drop k s -- rest of string
                in [x|->take n s']++(parse (tail ts) (drop n s'))
  enc(x) -> [x|-> dec (takeWhile elemX s)]++(parse ts (dropWhile elemX s))
```

Figure 1. Parser for formats with clear boundaries.

struction, we here over-approximate them by arbitrary-length variables. In fact the precise length are most relevant for parsing, but typically not for disjointness: in "good" designs, the distinction between two formats will not result from the precise lengths of some fields, but by some identifying constant in a fixed position. We come back to this issue in the TLS case study in Section VIII.

Theorem 2: If `disjoint(F_1,F_2)=True`, then $[\![F_1]\!] \cap [\![F_2]\!] = \emptyset$.

Proof: The proof is by structural induction, one induction step for each case of the algorithm: for each case the property holds if it holds for each recursive call of the algorithm.

As all cases are very similar, we give only one case: $(x[l], y[*])$. The induction hypothesis in this case (see the recursive calls of this case): $[\![F1]\!] \cap [\![y[*] : F2]\!] = \emptyset$ and $[\![x[*] : F1]\!] \cap [\![F2]\!] = \emptyset$. To show is that this implies $[\![x[l] : F1]\!] \cap [\![y[*] : F2]\!] = \emptyset$.

Suppose this were not true, i.e., there is a string $s \in [\![x[l] : F1]\!]$ and $s \in [\![y[*] : F2]\!]$. Thus s can be written as $s = s_1 \cdot s_1'$ for some s_1 with $|s_1| = l$ and some $s_1' \in [\![F1]\!]$; and also s can be written as $s = s_2 \cdot s_2'$ with some $s_2 \in \mathbb{B}^*$ and $s_2' \in [\![F_2]\!]$. Consider two cases: First, if $|s_1| \leq |s_2|$ then $s_2 = s_1 \cdot s_3$ for some string s_3. Thus $s_3 \cdot s_2' = s_1' \in [\![F_1]\!]$. Also $s_3 \cdot s_2' \in [\![y[*] : F2]\!]$ so, $[\![F_1]\!] \cap [\![y[*] : F2]\!]$ cannot be empty, contradicting the induction hypothesis.

Second, if $|s_1| > |s_2|$ then $s_1 = s_2 \cdot s_4$ for some s_4. Thus $s_4 \cdot s_1' = s_2' \in [\![F_2]\!]$ and also $s_4 \cdot s_1' \in [\![x[*] : F1]\!]$, so $[\![x[*] : F1]\!] \cap [\![F_2]\!] \neq \emptyset$, contradicting the induction hypothesis. ■

IV. The Concrete Algebra

We have so far defined a signature and semantics for terms as strings. One can see this as an algebra $\mathcal{C}$, where the universe is $\mathbb{P}(\mathbb{B}^*)$ and every function symbol f is interpreted by $f^{\mathcal{C}}$, thus the interpretation of a term t in $\mathcal{C}$ is $[\![t]\!]$.

For automated verification, this algebra is problematic, since we are using actual cryptography. For instance, hash functions necessarily have collisions in real cryptography, so there are terms t_1 and t_2 where $[\![t_1]\!] \neq [\![t_2]\!]$ but $[\![h(t_1)]\!] = [\![h(t_2)]\!]$. Of course, it should be difficult for an attacker to find collisions in reality (at least for good hash functions). The same holds for randomly chosen nonces or collisions between different constructions. For instance there may be terms t, k, and m such that $[\![h(t)]\!] = [\![\mathsf{crypt}(k, m)]\!]$, but it should be hard, given t, to find such k and m. One could say that these kinds of collisions are "accidents" that happen with a low probability and it is practically impossible for an attacker to exploit them. Therefore most formal verification approaches implicitly exclude these collisions by modeling messages as terms where in a free algebra, i.e., two terms are equal in the algebra iff they are syntactically equal.

We want to model all the details of the real world as far as *structuring/formatting* of messages is concerned, but still abstract from the cryptography. The rationale is of course that the clear-text operations are much easier to manipulate than cryptography, for instance it is not a problem, given a string $s \in \mathbb{X}^*$ to find another string $s' \in \mathbb{B}^*$ such that $s = \mathsf{enc}^{\mathcal{C}}(s')$.

We now define a new algebra that picks the best of both worlds, abstracting away all "accidental" collisions caused by the cryptography and preserving all details of the non-crypto operators. A key insight is that the unwanted collisions are due to the concrete choice of $\mathcal{C}$, i.e., how the symbols are interpreted. Recall that in the definition of a cryptographic algebra $\mathcal{C}$ we have made several requirements on the interpretation of the non-cryptographic function symbols (and on the universe and lengths), while we deliberately have not specified how the cryptographic functions work.

The idea is now to define an abstract algebra in which two ground terms t_1 and t_2 are equal iff they are equal in *every* cryptographic algebra $\mathcal{C}$. In other words, we want t_1 and t_2 to be interpreted equally only when this is a consequence of our requirements on the interpretation of the non-cryptographic functions:

Definition 4: We define $\approx$ as the least relation on ground

```
disjoint [] [] = False
disjoint [] (c:F2) = True
disjoint [] (x[*]:F2) = disjoint [] F2
disjoint [] (x[l]:F2) = if l>0 then True else disjoint [] F2
disjoint [] (enc(x[*]):F2) = disjoint [] F2
disjoint (f1:F1) (f2:F2) =
  case (f1,f2) of
  (c,c') -> c != c' || disjoint F1 F2
  -- if we start with different constants (both length 1) we are done
  -- otherwise compare the remaining fields
  (c,x[l]) -> if l==0 then disjoint (c:F1) F2 -- x is empty, discard
              else disjoint F1 (x[l-1]:F2)    -- the first letter of x consumes c
  (c,x[*]) -> disjoint (c:F1) F2              -- x may be empty (discard)
              && disjoint F1 (f2:F2)          -- otherwise x consumes c
  (c,enc(x[*])) -> (not (elemX c) || disjoint F1 (f2:F2))
                -- x can consume the c if c is in the alphabet
                && disjoint (c:F1) F2         -- otherwise x could be empty
  (x[l],y[m]) -> if (l<=m) then disjoint F1 (y[m-l]:F2)
                 -- if x is longer, then it is consumed by y
                 else disjoint (x[l-m]:F1) F2
  (x[l],y[*]) or (x[l],enc(y[*]))
        -> disjoint F1 (f2:F2)                -- x could be completely consumed by y
           && disjoint (x[*]:F1) (F2)      -- or y could be completely consumed by x,
           -- for simplicity remainder of x set to length *
  all other -> disjoint F1 (f2:F2) && disjoint (f1:F1) F2) -- one consumes the other
```

Figure 2. Sufficient check for format disjointness (omitting symmetric cases).

terms such that $t_1 \approx t_2$ iff in every cryptographic algebra $(\mathcal{C}, |\cdot|_{\mathcal{C}})$ it holds that $t_1^{\mathcal{C}} = t_2^{\mathcal{C}}$. Note that $\approx$ is a congruence relation. We extend it to non-ground terms $t_1 \approx t_2$ iff $\sigma(t_1) \approx \sigma(t_2)$ for every substitution σ that maps all variables to ground terms.

We define the *detailed term model* $\mathbb{D}$ as the quotient algebra $\mathcal{T}_\Sigma/\approx$. (The quotient algebra is an algebra that interprets two terms as equal iff they are equal modulo $\approx$. Formally, the universe of the quotient algebra is the set $\{[t]_\approx \mid t \in \mathcal{T}_\Sigma\}$ of equivalence classes $[t]_\approx = \{t' \mid t \approx t'\}$. The interpretation of function symbols in $\mathbb{D}$ is: $f^{\mathbb{D}}([t_1]_\approx, \ldots, [t_n]_\approx) = [f(t_1, \ldots, t_n)]_\approx$. Note that this definition of $f^{\mathbb{D}}$ chooses representatives $t_1, \ldots, t_n$ from the respective equivalence classes, but since $\approx$ is a congruence relation, the result does not depend on this choice.)

We give a few examples to illustrate this new algebra $\mathbb{D}$. First note that we have a very precise representation of the non-cryptographic operators. For instance

- cat is associative $\mathsf{cat}(t, \mathsf{cat}(u, v)) \approx \mathsf{cat}(\mathsf{cat}(t, u), v)$ and ϵ is the neutral element $s \cdot \epsilon \approx \epsilon \cdot s \approx s$.
- Length relates to concatenation as expected: $\mathsf{off}_k(s \cdot t) \approx \mathsf{off}_k(t \cdot s)$,
- When using hexadecimal encoding, we have $\mathsf{enc}(s \cdot t) \approx \mathsf{enc}(s) \cdot \mathsf{enc}(t)$. We have this property for Base64 only if $|s|$ is divisible by 3. (Note that this level of precision is beyond what one can axiomatize with algebraic equations.)

On the other hand, concerning cryptography, the algebra $\mathbb{D}$ behaves as standard term-algebraic message models:

- For any two different constants c and d we have $c \not\approx d$.
- Hash functions are collision free in $\mathbb{D}$: $h(s) \approx h(t)$ iff $s \approx t$ for any ground terms s and t.
- Similarly we do not have any clashes between distinct cryptographic elements: $\mathsf{crypt}(s, t) \not\approx h(u)$.

Due to the "semantic" definition of $\approx$ as equalities that hold in all crypto algebras, reasoning in $\mathbb{D}$ is not trivial. In fact, the key contribution of this work is that we can safely abstract from this and move to a free algebra. The first step towards this is the following generalization of the previous examples:

Lemma 1: Let $t = f(t_1, \ldots, t_n)$ and $t' = g(t'_1, \ldots, t'_m)$ be terms in which ϵ does not occur and $f \in \{\mathsf{crypt}, \mathsf{scrypt}, \mathsf{sign}, \mathsf{h}, \mathsf{mac}\}$. Then $t \approx t'$ implies $f = g$ and $t_i \approx t'_i$ for all $1 \leq i \leq n$.

Proof: The difficulty of this proof lies in the semantic definition of $\approx$ as those equations that hold in all crypto algebras. We proceed indirectly and assume terms t and t' as in the statement and where $f \neq g$ or $t_i \not\approx t'_i$ for some $1 \leq i \leq n$; we show that this implies $t \not\approx t'$. To prove $t \not\approx t'$, we construct a special crypto algebra $\mathcal{C}$ in which $t^{\mathcal{C}} \neq t'^{\mathcal{C}}$.

This $\mathcal{C}$ is a bit "absurd" in the sense that it does not reflect a useful implementation of the crypto operators; its mere purpose is to easily find an interpretation in which t and t' are unequal.

Let $\mathbb{D} = \mathbb{B}^* \setminus (\mathbb{X}^* \cup \mathbb{L})$. Note that this set is countably infinite. Thus there exists an injective function $\mathsf{fi} : (\mathbb{B}^*)^4 \to \mathbb{D}$. We may also assume that $|\mathsf{fi}(s_1, s_2, s_3, s_4)| \geq |s_1| + |s_2| + |s_3| + |s_4|$. We define the *injective crypto algebra* $\mathcal{C}$ as follows. We define $\mathsf{crypt}^C(s_1, s_2, s_3) = \mathsf{fi}(\mathtt{crypt}, s_1, s_2, s_3)$ where we literally introduce the string `crypt` as the first argument. Similarly we design an interpretation for all other uninterpreted symbols (all except $\{\mathsf{cat}, \mathsf{enc}, \mathsf{off}\} \cup \mathbb{L}$); when the arity is smaller than 3, we fill the respective s_i with ϵ.

We can easily conclude the proof for the cases $f = g$, $g \in \{\mathsf{crypt}, \mathsf{scrypt}, \mathsf{sign}, \mathsf{h}, \mathsf{mac}\}$, and $g \in \mathbb{U}$, since the injectivity of $\mathcal{C}$ ensures that t and t' are interpreted differently. Similarly, we already conclude for the cases when $g \in \mathbb{L}$ or $g = \mathsf{enc}$, since $\mathbb{D}$ is disjoint from $\mathbb{X}^*$ and $\mathbb{L}$.

The case $g = \mathsf{off}_k(\cdot)$ can be handled with a different crypto algebra: we set the particular interpretation of t to a string s with $|s| \neq k$. The case $g = \mathsf{form}_i$ is handled by replacing it with the corresponding format definition.

Finally for the case $g = \mathsf{cat}$, observe that in the injective crypto algebra $\mathcal{C}$, all crypto-operators produce a string that is at least as long as the sum of the lengths of the arguments, and this holds also for $\mathsf{enc}(\cdot)$ and $\mathsf{cat}(\cdot, \cdot)$; the only operator that may produce a shorter string is $\mathsf{off}_k(\cdot)$. This allows us to exclude that t' somehow contains t as a subterm: Suppose for some proper subterm t'' of t' we have $t''^{\mathcal{C}} = t'^{\mathcal{C}}$ and this subterm is not under an $\mathsf{off}_k(\cdot)$. Since ϵ cannot occur (i.e., we cannot have $t' = \mathsf{cat}(\epsilon, t)$ and the like), we have that $|t'^{\mathcal{C}}| > |t^{\mathcal{C}}|$ which excludes $t \approx t'$.

So we now can assume that no subterm of t' is $\mathcal{C}$-equivalent to t except maybe under some $\mathsf{off}_k(\cdot)$. Therefore, we either have $t^{\mathcal{C}} \neq t'^{\mathcal{C}}$ (and are thus already done) or, if $t^{\mathcal{C}} = t'^{\mathcal{C}}$ we can produce a modified crypto algebra $\mathcal{C}'$ that is identical to $\mathcal{C}$ except that we change the interpretation of f for the particular argument $t_1^{\mathcal{C}}, \ldots, t_n^{\mathcal{C}}$ to some different result of the same length (so that $\mathsf{off}_k(t)$ will not change). This will not change the interpretation of t' but of t, so $t'^{\mathcal{C}'} = t'^{\mathcal{C}} = t^{\mathcal{C}} \neq t^{\mathcal{C}'}$ and thus $t \not\approx t'$. ∎

A. Freedom of Forms

We have shown that the cryptographic operators in our algebra $\mathbb{D}$ behave like free functions. The non-cryptographic operators do not, in fact they have rather complicated properties. The crucial next step is that the form-operators indeed also behave like free function symbols in $\mathbb{D}$—if all forms are un-ambiguous and pairwise disjoint:

Lemma 2: Given form_i is an un-ambiguous format. Then $\mathsf{form}_i(t_1, \ldots, t_n) \approx \mathsf{form}_i(t'_1, \ldots, t'_n)$ implies $t_1 \approx t'_1, \ldots, t_n \approx t'_n$.

Given two disjoint formats form_i and form_j (thus $i \neq j$), then $\mathsf{form}_i(t_1, \ldots, t_n) \not\approx \mathsf{form}_j(t'_1, \ldots, t'_m)$.

Proof: For the first part, let $\mathsf{form}_i(x_1, \ldots, x_n) = \mathit{fld}_1, \ldots, \mathit{fld}_m$ be an un-ambiguous format definition. Assume $\mathsf{form}_i(t_1, \ldots, t_n) \approx \mathsf{form}_i(t'_1, \ldots, t'_n)$ while $t_i \not\approx t'_i$ for some i. Thus there is a crypto algebra $\mathcal{C}$ in which $t_k^{\mathcal{C}} \neq t_k'^{\mathcal{C}}$; fix such a crypto algebra $\mathcal{C}$ and let $s = \mathsf{form}_i(t'_1, \ldots, t'_n)^{\mathcal{C}}$ and $s_i = t_i'^{\mathcal{C}}$ for each i.

We use our parser and by Theorem 1, s can only be parsed for form_i by $\theta = [x_1 \mapsto s_1, \ldots, x_n \mapsto s_n]$ (in algebra $\mathcal{C}$). The unambiguity implies that s *cannot* be parsed as $\theta' = [x_1 \mapsto t_1^{\mathcal{C}}, \ldots, x_n \mapsto t_n^{\mathcal{C}}]$ since $t_k^C \neq t_k'^{\mathcal{C}} = s_k$ and thus $\theta' \neq \theta$. Thus, $\mathsf{form}_i(t_1, \ldots, t_n)^{\mathcal{C}} \neq \mathsf{form}_i(t'_1, \ldots, t'_n)^{\mathcal{C}}$ and thus $\mathsf{form}_i(t_1, \ldots, t_n) \not\approx \mathsf{form}_i(t'_1, \ldots, t'_n)$, contradicting the assumption.

For the second part, let form_i and form_j be disjoint formats. Suppose $\mathsf{form}_i(t_1, \ldots, t_n) \approx \mathsf{form}_j(t'_1, \ldots, t'_m)$. Then $\mathsf{form}_i(t_1, \ldots, t_n)^{\mathcal{C}} = \mathsf{form}_j(t'_1, \ldots, t'_m)^{\mathcal{C}}$ in any cryptographic algebra $\mathcal{C}$, and thus $[\![\mathsf{form}_i(x_1, \ldots, x_n)]\!]_{\mathcal{C}} \cap [\![\mathsf{form}_j(y_1, \ldots, y_m)]\!]_{\mathcal{C}} \neq \emptyset$, contradicting the disjointness of the forms. ∎

Now suppose we are considering only terms that do not directly use the "low-level string operations" $\mathsf{cat}, \mathsf{off}, \mathsf{enc}, \epsilon$, but instead use formats. Thus we are considering the signature $\Sigma' = \Sigma \setminus \{\mathsf{cat}, \mathsf{off}, \mathsf{enc}, \epsilon\}$. Putting Lemmata 1 and 2 together we have that the resulting sub-algebra of $\mathbb{D}$ over Σ' is isomorphic to the free term algebra over Σ':

Theorem 3: Let $\Sigma' = \Sigma \setminus \{\mathsf{cat}, \mathsf{off}, \mathsf{enc}, \epsilon\}$ and suppose all formats are un-ambiguous and pairwise disjoint. Then for all $s, t \in \mathcal{T}_{\Sigma'}$ holds $s \approx t$ iff $s = t$. Moreover for terms with variables we can use free algebra unification: let $s, t \in \mathcal{T}_{\Sigma'}(\mathcal{V})$ and τ be any unifier with $\tau(s) \approx \tau(t)$ and over full Σ. Then s, t have a most general unifier σ in the free algebra over $\mathcal{T}_{\Sigma'}$ and τ is an instance of σ modulo $\approx$.

Proof: The statement $s \approx t$ iff $s = t$ for ground terms follows from Lemmata 1 and 2. For unification assume we are given a set of pairs of terms over $\mathcal{T}_{\Sigma'}(\mathcal{V})$. We follow the steps of the free algebra unification algorithm, obtaining the most general unifier σ if one exists, and show for every step that any unifier τ (over Σ and $\approx$) is an instance of σ.

First, if we have a pair (x, t) of a variable x to unify with some term t (or the symmetric case (t, x)); then we check if x occurs as a proper subterm in t. In this case there is no unifier modulo $\approx$, because $t = f(\ldots)$ (for x to be a proper subterm of t) and f cannot be cat or $\mathsf{off_k}$ by assumption, preventing that t could "collapse" to one of its subterms. Otherwise, if x does not occur in t, $[x \mapsto t]$ is the valid most general unifier for this pair (and any unifier τ must support this); so we apply it to all pairs and continue.

If we have a pair (t, t') with $t = f(t_1, \ldots, t_n)$ and $g(t'_1, \ldots, t'_m)$, then the free algebra unification algorithm checks that $f = g$ (and fails otherwise) and replaces the pair (t, t') with $U = \{(t_1, t'_1), \ldots, (t_n, t'_n)\}$. We have to show that every solution τ with $\tau(t) \approx \tau(t')$ is also a solution for U. Suppose τ is a grounding solution, i.e., $\tau(t)$ and $\tau(t')$ are ground, then by Lemmata 1 and 2, $f = g$ and $\tau(t_i) \approx \tau(t'_i)$

for every $1 \leq i \leq n$. Also the non-grounding τ support U since all their grounding instances do. ■

V. Intruder Model

We have defined an algebra $\mathbb{D}$ to represent terms and defining when two terms are equal. Now we finally do something with these terms and define a (Dolev-Yao-style) deduction relation $K \vdash t$ where K is a finite set of terms (messages) and t is a term. We say *the intruder can derive term t from knowledge K*. We define $\vdash$ to be the least relation closed under the rules in Fig. 3 that we explain in the following.

The $(Axiom)$ rule says that every term in K is derivable. For composing terms, we first define a subset $\Sigma_p \subseteq \Sigma$ that describes the *public* symbols: we assume that functions like encryption are not themselves secret (only the keys may be), so the intruder is able to apply them to any terms he knows. For simplicity we assume that all symbols except uninterpreted constants $\mathbb{U}$ are public.[3] This in particular means that all literals in $\mathbb{L}$ are public. One reason for this is that the intruder should always be able to create formats himself (replacing the parameters x_i with values he knows). Thus, rule $(Comp)$ expresses that the intruder can apply functions in Σ_p to any known terms.

For decomposition, the intruder can decrypt messages if he has the necessary decryption key. This is formalized in the rules $(Dscrypt)$, $(Dcrypt)$, and $(Open)$. (We model here a signature primitive that reveals the signed text even without knowing the verification key; but other models are also compatible with our approach.) Similarly, for cat and enc and all the forms, the intruder can obtain the "contents", as formalized by $(Split)$, (Dec), and $(Parse)$. For hashes and Macs as well as $\mathsf{off_k}$, we have no rules since the intruder cannot extract the original message in these cases.

Finally, we have the rule (Eq) that says that for every term, the intruder can derive also every $\approx$-equivalent variant. Note that the rule $(Parse)$ is redundant in the presence of (Eq) since the intruder can first rewrite the respective form into its definition as a concatenation (with encodings) and then apply $(Split)$ and (Dec) as necessary. However, we want to show below that under certain conditions, we can safely omit the rule (Eq) without excluding attacks.

A. The Lazy Intruder

A popular verification technique is the constraint-based approach, that we refer to as the *lazy intruder* [6]–[8]. The basic idea is that many security problems of communicating processes can be reduced to satisfiability of a number of constraint satisfaction problems. (In fact, this number of constraint satisfaction problems is in general infinite, but it is finite if all processes except the intruder can perform only finitely many transitions.)

[3] As a variant, some verification approaches may use "private" functions like $\mathsf{shk}(A, B)$ to denote a secret shared key of agents A and B, these will then of course be excluded from Σ_p.

$$\frac{}{K \vdash t}\ (Axiom)\ t \in K$$

$$\frac{K \vdash t_1 \quad \ldots \quad K \vdash t_n}{K \vdash f(t_1, \ldots, t_n)}\ (Comp)\ f \in \Sigma_P$$

$$\frac{K \vdash \mathsf{scrypt}(k, m) \quad K \vdash k}{K \vdash m}\ (Dscrypt)$$

$$\frac{K \vdash \mathsf{crypt}(pub(k), m) \quad K \vdash priv(k)}{K \vdash m}\ (Dcrypt)$$

$$\frac{K \vdash \mathsf{sign}(priv(k), m)}{K \vdash m}\ (Open)$$

$$\frac{K \vdash \mathsf{cat}(m_1, m_2)}{K \vdash m_i}\ (Split) \quad \frac{K \vdash \mathsf{enc}(m)}{K \vdash m}\ (Dec)$$

$$\frac{K \vdash \mathsf{form}_i(m_1, \ldots, m_n)}{K \vdash m_j}\ (Parse)\ j \in \{m_1, \ldots, m_n\}$$

$$\frac{K \vdash t}{K \vdash t'}\ (Eq)\ t \approx t'$$

Figure 3. Intruder deduction relation.

Each constraint satisfaction problem is a finite conjunction $\bigwedge_{i=1}^{n} K_i \vdash t_i$. We only give an intuition why this reflects a security problem; for a formal account see for instance [8]. The intuition is that we represent a symbolic state transition system where each state is parametrized over some variables (that are placeholders for arbitrary ground terms) along with some constraints that decide which values for the variables are possible and a current knowledge K of the intruder. Whenever the intruder receives a message, we add it his knowledge K, and whenever the intruder sends a message, we create a new variable x and add the new constraint $K \vdash x$. We work with x as the message sent and accordingly substitute it for a more concrete term if the receiver has requirements on x. Thus, roughly speaking, for constraints of the form $K \vdash t$ the messages in K are received by the intruder from honest agents, and t is the pattern of message that an honest agent is willing to receive.

We have so far not formally defined $K \vdash t$ for terms with variables. An *interpretation* $\mathcal{I}$ is a mapping from $\mathcal{V}$ to $\mathcal{T}_\Sigma$, and we extend it to a function on terms and sets of terms as expected. The define that $\mathcal{I} \models K \vdash t$ iff $\mathcal{I}(K) \vdash \mathcal{I}(t)$. For conjunctions we have of course $\mathcal{I} \models \phi \wedge \psi$ iff both $\mathcal{I} \models \phi$ and $\mathcal{I} \models \psi$. We say that a constraint ϕ is *satisfiable* iff there is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \phi$.

VI. Main Result

We now use the lazy intruder approach as a convenient way to represent and reason about attacks. We emphasize, however, that our result is independent from a particular

verification technique like the lazy intruder. Also, while the lazy intruder provides a decision procedure only when limiting the steps of all non-intruder processes, our result does not rely on such bounds. We do only assume that one considers a security problem that can be reduced to a number of lazy intruder constraint reduction problems (possibly infinitely many).

Our main result is now: when using formats, a lazy intruder constraint is satisfiable iff it is satisfiable in the free algebra. Here, *using formats* intuitively means that the honest agents never use the low-level functions $\mathsf{cat}, \mathsf{enc}, \mathsf{off}, \epsilon$ directly but instead always use one of the form_i to structure messages. Moreover all used forms have to be un-ambiguous and pairwise disjoint. Remember that modulo $\approx$, each form is equivalent to some low-level term, and the intruder is always able to use low-level terms. The point is that if all honest participants use un-ambiguous and pair-wise disjoint formats, all attempts to manipulate messages on the low-level is pointless for the intruder.

We formalize this notion of "using formats" as conditions on the terms in lazy intruder constraints: recall that in $K \vdash t$, all K and t terms are message patterns sent and received by honest agents, respectively.

Definition 5: A lazy intruder constraint $\bigwedge_{i=0}^{n} K_i \vdash t_i$ is called *well-formatted* iff the following holds:

- $K_0 \subseteq K_1 \ldots \subseteq K_n$
- Each variable occurring in K_i also occurs in $\{t_1, \ldots, t_{i-1}\}$. [4]
- The symbols $\{\mathsf{cat}, \mathsf{enc}, \mathsf{off}, \epsilon\}$ do not occur in any K_i or t_i.
- All form_i of Σ are un-ambiguous and pairwise disjoint.

Theorem 4: Given a well-formatted, satisfiable intruder constraint ϕ, then exists an interpretation $\mathcal{I}$ that satisfies ϕ in the *free algebra*, i.e., no deduction proof $\mathcal{I}(K) \vdash \mathcal{I}(t)$ requires the use of the (Eq) rule, and $\mathcal{I}$ maps every variable to a ground term in which $\mathsf{cat}, \mathsf{enc}, \mathsf{off}_\mathsf{k}, \epsilon$ do not occur.

Proof: Given a well-formatted constraint ϕ that is satisfiable, and let $\mathcal{I}$ be a satisfying interpretation. Thus for every constraint $K_i \vdash t_i$, we can label t_i with a ground proof tree for $\mathcal{I}(K_i) \vdash \mathcal{I}(t_i)$ (formed with instances of the rules in Fig. 3). We show that there is an interpretation $\mathcal{I}'$ that satisfies ϕ and that does not use any low-level symbols and that is a solution in the free algebra (i.e., that works without the (Eq) rule of Fig. 3). To that end, we step by step transform the constraint, where these transformations are sound in the sense that the constraint has either the same set of models or fewer ones.

No top-level forms: First observe that a constraint of shape $K \cup \{\mathsf{form}_i(t_1, \ldots, t_n)\} \vdash t$ is equivalent to $K \cup \{t_1, \ldots, t_n\} \vdash t$. To see that, we can adapt the proof tree for $\mathcal{I}(K) \vdash \mathcal{I}(t)$ accordingly: whenever $\mathsf{form}_i(t_1, \ldots, t_n)$ is needed, we can compose it from the t_i using $(Comp)$. Similarly the constraint $K \vdash \mathsf{form}_i(t_1, \ldots, t_n)$ is equivalent to $K \vdash t_1 \wedge \ldots \wedge K \vdash t_n$. Thus, whenever we have form_i on the left-hand or right-hand side of a constraint, we can replace it by its subterms without changing the set of models. We can thus silently assume through the rest of this procedure that we do not have any terms with root symbol form_i. (Note form_i terms may well occur as proper subterms and thus "come to the surface" during our constraint reduction procedure.)

Simple Constraints: If the constraint has only variables on the right-hand side, i.e., all $t_i \in \mathcal{V}$, then we are done, because the intruder can always construct *some* term from his knowledge.

Non-Simple Constraints: Thus in the following we only need to care about constraints that have at least one $t_i \notin \mathcal{V}$. Choose the one with the lowest index i, and consider the proof tree for $\mathcal{I}(K_i) \vdash \mathcal{I}(t_i)$. We consider different cases based on the root node in this proof tree.

- For every (Eq) node we consider also the next deeper node in the tree; if it is again (Eq), we can merge the two proof steps to one (since $\approx$ is transitive). Otherwise, if it is one of the $(Axiom)$, $(Comp)$, or destructors, we consider the (Eq) together with the respective rule. Thus in the following we will consider $(Axiom)$, $(Comp)$, and destructors, modulo $\approx$.
- If it is the $(Axiom)$ rule, then there is a term $t' \in K_i$ such that $\mathcal{I}(t') \approx \mathcal{I}(t_i)$. Note that t_i is not a variable, but t' may be. If $t' = x$ is a variable, then by well-formattedness there is an earlier t_j, $j < i$ in which x occurs. Since t_i is the first term that is not a variable, $t_j = x$. Thus we have $K_j \vdash x$, and we can simply use instead the derivation tree for $\mathcal{I}(K_j) \vdash \mathcal{I}(x)$. (This kind of reduction is well-founded since there is a smallest t_i.)

  If t' is not a variable, then t' and t'_i have by Theorem 3 a unique most general unifier σ that is identical with the most general unifier in the free-algebra and that does not map any variable to a term with low-level symbols. Thus $\mathcal{I}$ is an instance of σ (i.e., $\mathcal{I}(x) = \mathcal{I}(\sigma(x))$ for all variables). We can thus apply σ to the entire constraint without loosing the model $\mathcal{I}$ (and without introducing new models). After this substitution, we have $\sigma(K_i) \vdash \sigma(t_i)$ which is tautological as $\sigma(t_i) \in \sigma(K_i)$ and we can thus delete this conjunct.
- If it is a $(Comp)$, i.e., we have a proof tree of the form

$$\frac{\mathcal{I}(K_i) \vdash u_1 \quad \ldots \mathcal{I}(K_i) \vdash u_k}{\mathcal{I}(K_i) \vdash \mathcal{I}(t_i)}$$

  where $\mathcal{I}(t_i) \approx f(u_1, \ldots, u_k)$ for some $f \in \Sigma_p$. Note that $t_i = f'(u'_1, \ldots, u'_l)$ where $f' \notin \{\mathsf{cat}, \mathsf{enc}, \mathsf{off}, \mathsf{form}_\cdot(\cdot), \epsilon\}$ (recall that in case of top-level $\mathsf{form}_\cdot(\cdot)$ we move to equivalent constraints with-

[4] This and the previous condition are often called *well-formedness* and are standard in lazy intruder approaches: the intruder knowledge monotonically grows and all variables that occur in messages from other participants actually result from choice of the intruder in earlier messages.

out top-level $\mathsf{form}_\cdot(\cdot)$). Again by Theorem 3, we have thus $f = f'$ and $u_j \approx \mathcal{I}(u'_j)$ for all j. Thus, we can replace conjunct $K_i \vdash t_i$ with $K_i \vdash u'_1 \wedge \ldots \wedge K_i \vdash u'_l$. The modified constraint still supports model $\mathcal{I}$ and does not introduce new models.

- The analysis steps are a bit tricky, since the proof trees may contain further analysis or composition steps. The general principle is here to first go to one of the "inner-most" analysis steps, i.e., so that no sub-tree of the proof has further analysis steps. We will first handle that analysis step. To make this easier to read, let us suppose this is an $(Dscrypt)$ step; the cases $(Dcrypt)$, $(Open)$, and $(Parse)$ are similar, the cases $(Split)$ and (Dec) will be handled separately.
  So we have a step of the form

$$\frac{\dfrac{\Pi_1}{\mathcal{I}(K_i) \vdash \mathsf{scrypt}(k, m)} \quad \dfrac{\Pi_2}{\mathcal{I}(K_i) \vdash k}}{\mathcal{I}(K_i) \vdash m}$$

  with two sub-trees Π_1 and Π_2. If Π_1 is a composition step, then the intruder has just symmetrically encrypted a message and then decrypted it again. (That this composition could be by any other operator than $\mathsf{scrypt}(\cdot,\cdot)$ is excluded here again by Theorem 3.) We can then simplify the proof tree by replacing the derivation for m with the corresponding subtree of Π_1. Otherwise, it can only be a leaf. Then there is a term $t' \in K_i$ such that $\mathcal{I}(t') \approx \mathsf{scrypt}(k, m)$.
  If t' is actually a variable, say $t' = x$, then x must be introduced in an earlier constraint $T_j \vdash t_j = x$, and we replace the proof tree Π_1 with the proof tree for t_j, and continue with the next applicable case.
  Finally if t' is not a variable, then $t' = \mathsf{scrypt}(k', m')$ for some k' and m' with $\mathcal{I}(k') \approx k$ and $\mathcal{I}(m') \approx m$. We can then indeed realize the respective decryption step on the constraint. To that end, note that $\mathcal{I} \models K_i \vdash k'$ (as proved by Π_2), so we do not loose the model $\mathcal{I}$ if we add $K_i \vdash k'$ to our constraint. Then we can safely add the decrypted message m' to all K_j with $j \geq i$. This is possible since m' can be derived in every $K_j \supseteq K_i$ and it is necessary to add to all these $K_j \supseteq K_i$ to ensure that they remain supersets (or else we would destroy well-formattedness).
- If the inner-most analysis step is (Dec) (and the case $(Split)$ is similar). We then have a proof-subtree of the form:

$$\frac{\dfrac{\Pi}{\mathcal{I}(K_i) \vdash \mathsf{enc}(m)}}{\mathcal{I}(K_i) \vdash m}$$

  where Π is a proof tree for $\mathsf{enc}(m)$. If Π is a leaf node, then consider the corresponding term $m' \in K_i$ with $\mathcal{I}(m') \approx \mathsf{enc}(m)$. If it is a variable $m' = x$, again we replace the proof tree Π with the proof tree that x has in the constraint where x was introduced.
  The remaining case is that Π consists of $(Comp)$ steps and axioms. We claim that in this case we can find a composition for m directly, without constructing a more complex term $\mathsf{enc}(m)$ (or $\mathsf{cat}(m,\cdot)$ or $\mathsf{cat}(\cdot,m)$) that has to be decomposed. This final piece of the proof is given in the next paragraph.

Elimination of enc and pair: It remains to show a property for proofs that use only composition except for a final step (Dec) or $(Split)$: we show that these can be done as pure composition proofs without (Dec) or $(Split)$. More precisely, given a set K of ground terms where every term has a top-level symbol in $\{\mathsf{crypt}, \ldots, \mathsf{sign}, \mathsf{h}, \mathsf{mac}\}$ as well as public constants. We say a *comp-proof* is a $K \vdash t$ proof that uses only (Eq), $(Comp)$ and $(Axiom)$. We now show: If there is a comp-proof $K \vdash \mathsf{enc}(m)$ then there is also a comp-proof for $K \vdash m$. (Similarly, one can prove: if there is a comp-proof for $K \vdash \mathsf{cat}(m_1, m_2)$ then there are comp-proofs for $K \vdash m_1$ and $K \vdash m_2$.)

Since $K \vdash \mathsf{enc}(m)$, we can syntactically construct a term t using only elements of K and public symbols, such that $t \approx \mathsf{enc}(m)$. Note that m may contain a subterm of the form $\mathsf{off}_k(m_0)$ where m_0 can not necessarily be constructed from K. Let m' be a modification of m where each such $\mathsf{off}_k(m_0)$ is replaced this with an $\approx$-equivalent $\mathsf{off}_k(m'_0)$ where m'_0 has the same length as m_0 and can be constructed from K and public symbols. Note that $m' \approx m$, and thus, if we can prove $K \vdash m'$ with only $(Comp)$, so we can prove $K \vdash m$ with only $(Comp)$. To that end it suffices to prove that m' can be syntactically constructed using only elements in K and public symbols.

Suppose this were not the case, i.e., m' contains some subterm m_1 that cannot be composed with elements from K and public symbols (and this subterm cannot be underneath an $\mathsf{off}_k(\cdot)$). Let $\mathcal{C}$ be again the injective crypto algebra from the proof of Lemma 1 that is injective on the cryptographic functions and uninterpreted constants in $\mathbb{U}$. Recall that $\mathsf{enc}(m') \approx \mathsf{enc}(m) \approx t$, thus $\mathsf{enc}(m')^{\mathcal{C}} = t^{\mathcal{C}}$. Since m_1 is a term that occurs only in m' and not in t, we can create a modified crypto algebra $\mathcal{C}'$ in which ${m'}^{\mathcal{C}'} \neq {m'}^{\mathcal{C}}$ but $t^{\mathcal{C}'} = t^{C}$. Then we have $\mathsf{enc}(m')^{\mathcal{C}'} \neq t^{C'}$ and thus $\mathsf{enc}(m') \not\approx t$, contradicting the assumption.

Termination: One may wonder if our transformation of the constraints can run into an infinite loop. In fact most steps reduce the problem in some sense as they work off nodes of the proof trees. However, there are substitutions which can increase the actual size of the constraint store again, since they replace variables with more complex terms. However, since all unifiers are in the free algebra, the number of variables in the constraints decreases with every substitution (except the identity), and so no sequence of steps can involve an infinite number of non-identity substitutions, and all other steps decrease the constraint size. ■

We conclude this section with the remark that in some applications also arise some inequality constraints $s \not\approx t$ along with the intruder deduction constraints. As for instance shown in [8], this extension can be reduced to some $\approx$-unification problem. This unification problem is not hard: since s and t are produced by honest participants, s and t are again terms without low-level symbols and by Theorem 3 we can thus use free-algebra unification.

VII. Abstraction of the Scanning Problem

We have so far considered forms without whitespaces. However formats like XML allow whitespaces between tokens. For instance in the example $\circ\langle\circ\,\mathsf{nonce}\,\circ\rangle\circ\,\mathsf{enc}(x)\circ\langle/\circ\,\mathsf{nonce}\,\circ\rangle\circ$ we have marked with $\circ$ all positions where XML would allow whitespaces (we like nonce to be parsed as a token name, so it cannot be broken by whitespaces); additionally, one may insert whitespaces anywhere into the string produced by $\mathsf{enc}(x)$. An implementation that allows for whitespaces should thus accept any string for $\mathsf{enc}(x)$ that consists only of symbols of $\mathbb{X}$ and whitespaces, and simply filter the whitespaces. Vice-versa, if we think of implementations generating $\mathsf{enc}(x)$, the implementation is free to insert whitespaces, adding non-determinism to the model. In order to keep this manageable, we assume a modified function $\mathsf{enc}(t, r)$ where t is the term to encode and r is a randomness string, specifying in some way where and how to insert white spaces. As is already the case for encryption, we may simply omit the r in the notation when not interesting.

For all other whitespaces, we can use special variables $w_1, w_2, \ldots$ to insert into the formats.[5] We have usually $|w_i| = \mathbb{N}$ and we may have $|w_i| = \mathbb{N} \setminus \{0\}$ for mandatory whitespaces. We thus have an alphabet $\mathbb{W} \subset \mathbb{B}$ of whitespaces, $\mathbb{W} \cap \mathbb{X} = \emptyset$, and string substitution must substitute whitespace variables only with strings from $\mathbb{W}^*$. We require $\mathbb{W} \subseteq \Sigma_p$. These variables are additional parameters of the formats, and again we treat them as "silent" arguments (i.e., we do not denote them when no confusion arises).

This extension requires some updates on the results. First, for ambiguity of formats, we require that whitespace variables cannot be followed by a variable x (while $\mathsf{enc}(x)$ is fine) and it cannot be followed by $\mathsf{off}_k(x)$. This requirement is necessary since for both x and $\mathsf{off}_k(x)$, the concrete strings in $[\![\cdot]\!]$ can start with a byte in $\mathbb{W}$, so the boundary may be unclear. Under this additional restriction, the parser is however easy to extend and we have again un-ambiguity. For disjointness, the checking procedure is updated also in a straightforward way: whitespace variables are treated like normal (arbitrary-size) variables, except that constants that are not in $\mathbb{W}$ are disjoint from them.

The results of Theorem 3 and 4 hold under this extension: the additional silent arguments to $\mathsf{enc}(\cdot)$ and form_i do never hurt, and the whitespace variables can be treated as normal variables. Note when we have a constraint $K \vdash w$ for a whitespace variable w, the intruder can also solve this correctly, since we have set the whitespace alphabet $\mathbb{W}$ to be public symbols.

[5] From a users point of view, a more convenient notation is to use a special "scanner" declaration, defining keywords/tags, whitespaces and comments.

VIII. Case Study: TLS

As a practical case study, we examine the message formats used in TLS [9]. Given the complexity of TLS with many optional choices, it is not surprising that some of its features are at the border of what the result of this paper supports—and some are beyond. This indicates potential future extensions that would be helpful in practice, but also demonstrates how much our current method already covers. Note that we do not verify TLS itself here, but we formalize all message formats of TLS and show that they are pairwise disjoint and unambiguous, so that it is sound to abstract the formats into free function symbols as is standard in formal models like [10].

Note also that typical models as in [10], [11] make further simplifications, e.g., cryptographic, while our format abstraction can also be used in models TLS.

Structure of TLS

TLS has two layers. At the top layer we have four subprotocols: *Handshake* for negotiating session keys, *Change Cipher* for setting these keys into use, *Alert* for warning and error messages, and *Application Data* for transmitting the actual payload data. (We discuss the later addition of the *Heartbeat* protocol below.) At the bottom, we have the record protocol that acts as a kind of envelope for the top layer protocols and can be described by a format as follows:

$$\textsc{record}(\mathrm{sub}, \mathrm{data}) = \mathrm{sub}{\cdot}\mathsf{byte}(3){\cdot}\mathsf{byte}(3){\cdot}\mathsf{off}_2(\mathrm{data}){\cdot}\mathrm{data}$$

where $\mathsf{byte}(n)$ denotes literal one-byte constants (e.g., $\mathsf{byte}(3) \cdot \mathsf{byte}(3)$ is the (fixed) version number, actually referred to as "TLS 1.2"). Variable sub is a one-byte tag that specifies to which subprotocol data belongs to:

- $\mathsf{byte}(20)$ for the change cipher spec protocol,
- $\mathsf{byte}(21)$ for the alert protocol,
- $\mathsf{byte}(22)$ for the handshake protocol, and
- $\mathsf{byte}(23)$ for the application data protocol.

We have set the format RECORD in small capitals to indicate that this is actually a "meta-format": as *concrete* formats we consider said four instantiations of sub and these are obviously pairwise disjoint.

The data is however problematic, since by the 2-byte offset, data needs to be less than 2^{16} bytes long (and the standard restricts it even further for technical reasons). Thus, longer messages need to be fragmented into several record packets by the sender, and joined again by the recipient. This is beyond our current soundness result, and we can thus cover here TLS only for data packets up to that length. However, it seems intuitively clear that the fragmenting

mechanism does not induce further vulnerabilities: an attacker can of course re-order these packets, but he could do the same in a variant of the protocol where data is transmitted as a single chunk (with a larger size bound or unbounded)—and for this variant our result applies.

Handshake

We illustrate some key points at hand of a few formats of the Handshake protocol. The full formalization can be found in Fig. 4. Here, we abbreviate the notation $\mathsf{off}_k(m) \cdot m$ with $[m]_k$. Also, in Fig. 5 we show an example of how a typical exchange looks like in Alice-and-Bob notation using a subset of the formats. Note that in this version the client is not authenticated and we leave out all complications such as error handling or re-opening of sessions—as is often done in simple protocol models.

Again HANDSHAKE is a meta-format where tag and data are instantiated with concrete values to define formats like $\mathrm{server\_hello}$. (Variables like random that are used without offset generally have some fixed length.) One complication here is that the client can ask for some extended functionalities and the server should tell in the server hello message which of these functionalities that are actually available. In this case we have an extended variant of the format:

$$\begin{aligned}&\mathrm{server\_helloE}(\mathrm{random}, \mathrm{session\_id}, \mathrm{cipher}, \mathrm{compr}, \mathrm{exts})\\ &= \textsc{handshake}(\mathsf{byte}(2), \ldots \cdot \mathrm{compr} \cdot \mathsf{off}_2(\mathrm{exts}) \cdot \mathrm{exts})\end{aligned}$$

Note that $\mathrm{server\_hello}$ and $\mathrm{server\_helloE}$ are disjoint as they both have clear boundaries and all variables that are not presented with offsets have fixed lengths. Also note that there is no flag that says whether the extension is present—this is decided by whether or not after parsing compr further bytes are available. The server hello is an example of a complication that is still supported by our approach, however a more convenient notation for optional parameters/constructions is desirable.

Forms $\mathrm{client\_key\_ex}$ and $\mathrm{PMS\_form}$ are an example of messages that involve a cryptographic operations. This will in fact be used as

$$\mathrm{client\_key\_ex}(\mathsf{crypt}(\mathsf{pub}(\mathrm{k}), \mathrm{PMS\_form}(\mathrm{PMS})))$$

where $\mathsf{pub}(k)$ is the public key of the server and PMS is the so-called pre-master-secret randomly generated by the client. The point is that whenever we are not performing an encryption of "raw" data like PMS, we should define a format into which the information is embedded—in this case, the $\mathrm{PMS\_form}$ includes the version number.

As a last example, consider the $\mathrm{cert\_verify}$ message. Here, the $\mathrm{signed\_handshake}$ should actually contain a signed hash of all previous handshake messages of the session, i.e., $\mathrm{h}(\mathrm{m}_1 \cdot \ldots \cdot \mathrm{m}_\mathrm{k})$. Here we thus concatenate a sequence of messages without any further formatting constructs. Note however that each of the m_i is an instance of one of the formats, say $\mathrm{form}_\mathrm{i}(\mathrm{ts}_\mathrm{i})$ for some list of parameters ts_i. We could thus define an "all messages format" as follows (removing duplicates in the arguments):

$$\mathrm{all}(\mathrm{ts}_1, \ldots, \mathrm{ts}_\mathrm{n}) = \mathrm{form}_1(\mathrm{ts}_1) \cdot \ldots \cdot \mathrm{form}_\mathrm{n}(\mathrm{ts}_\mathrm{n})$$

As such, the all format is again unambiguous and disjoint from all other formats. This follows from the fact that for every string that can be parsed as one TLS format, no proper prefix of that string can be parsed as the same format.

There is however a subtle problem: as mentioned some formats like $\mathrm{server\_hello}$ have an optional extension which simply consists of further bytes trailing the mandatory part. When both variants are in use, also the concatenations occur in the different variants, e.g.,

$$\begin{aligned}\mathrm{all}_1(\ldots) &= \ldots \cdot \mathrm{server\_hello} \cdot \mathrm{server\_cert} \ldots\\ \mathrm{all}_2(\ldots) &= \ldots \cdot \mathrm{server\_helloE} \cdot \mathrm{server\_cert} \ldots\end{aligned}$$

and these variants are in general not disjoint, if for instance the extension is not disjoint from the beginning of $\mathrm{server\_cert}$. The danger is that an intruder can get away with deleting or inserting information like available/requested extensions. In the concrete cases this seems practically infeasible (due to signed certificates). However this indicates a weakness of the "trailing optionals" approach of TLS, which, as the standard states, is an outdated approach kept for compatibility. It seems reasonable to change this, spending an extra byte for flagging whether an extension is present or not.

During the publication process of this paper the so-called *Heartbleed attack* was discovered in an implementation of the *Heartbeat extension* of TLS [12]. One may in fact argue that this is a problem of parsing messages: in particular the vulnerable implementation suffers from a buffer overflow. While the format (i.e., the protocol standard) is arguably not to blame for the flaw, one may thus wonder if an implementation that uses our format abstraction could have prevented the problem. The idea is that our format descriptions can be *automatically* translated into an API of parsers and pretty printers, so that the implementation does not need to directly handle byte strings anymore but rather always uses the format API, similar to the use of crypto-APIs that is already standard today. This could help to minimize the chance of such accidents in implementations.

IX. Conclusions and Related Work

This paper presents a novel kind of term algebraic model that models all details of the non-cryptographic operators for structuring messages. At the same time it abstracts from all details of the cryptography. It does so by taking the real byte-string message algebra and defining a new algebra $\mathbb{D}$ based on congruence relation $\approx$ that includes exactly those equalities that hold regardless of the cryptographic algorithms. While this algebra is semantically defined and thus hard to handle directly in reasoning, we show that when all formats are un-ambiguous and pairwise disjoint,

$\text{HANDSHAKE}(\text{tag}, \text{data}) = \text{tag} \cdot [\text{data}]_3$
$\text{hello\_req}() = \text{HANDSHAKE}(\mathsf{byte}(0), \epsilon)$
$\text{client\_hello}(\text{time}, \text{random}, \text{session\_id}, \text{cipher\_suites}, \text{comp\_methods})$
$\quad = \text{HANDSHAKE}(\mathsf{byte}(1), \mathsf{byte}(3) \cdot \mathsf{byte}(3) \cdot \text{time} \cdot \text{random} \cdot [\text{session\_id}]_1 \cdot [\text{cipher\_suites}]_2 \cdot [\text{comp\_methods}]_1)$
$\text{server\_hello}(\text{time}, \text{random}, \text{session\_id}, \text{chosen\_cipher}, \text{chosen\_comp})$
$\quad = \text{HANDSHAKE}(\mathsf{byte}(2), \mathsf{byte}(3) \cdot \mathsf{byte}(3) \cdot \text{time} \cdot \text{random} \cdot [\text{session\_id}]_1, \text{chosen\_cipher}, \text{chosen\_comp})$
$\text{server\_cert}(\text{certificate\_tls\_vec}) = \text{HANDSHAKE}(\mathsf{byte}(11), [\text{certificate\_tls\_vec}]_3)$
$\text{server\_key\_exchange}(\text{cipher\_config}) = \text{HANDSHAKE}(\mathsf{byte}(12), \text{cipher\_config})$
$\text{cert\_request}(\text{cert\_type}, \text{supp\_alg}, \text{cert\_auths}) = \text{HANDSHAKE}(\mathsf{byte}(13), [\text{cert\_type}]_1 \cdot [\text{supp\_alg}]_2 \cdot [\text{cert\_auths}]_2)$
$\text{server\_hello\_done}() = \text{HANDSHAKE}(\mathsf{byte}(14), \epsilon)$
$\text{cert\_verify}(\text{signed\_handshake}) = \text{HANDSHAKE}(\mathsf{byte}(15), \text{signed\_handshake})$
$\text{client\_key\_ex}(\text{EncrPreMasterSecret}) = \text{HANDSHAKE}(\mathsf{byte}(16), \text{EncrPreMasterSecret})$
$\text{finished}(\text{encr\_finished}) = \text{HANDSHAKE}(\mathsf{byte}(20), \text{encr\_finished})$
$\text{PMS\_form}(\text{secret}) = \mathsf{byte}(3) \cdot \mathsf{byte}(3) \cdot \text{secret}$
$\text{master\_form}(\text{PMS}, \text{R}) = \text{PMS} \cdot \texttt{"master secret"} \cdot \text{R}$
$\text{client\_finished}(\text{MS}, \text{R}, \text{H}) = \text{MS} \cdot \texttt{"client finished"} \cdot \text{R} \cdot \text{H}$
$\text{server\_finished}(\text{MS}, \text{R}, \text{H}) = \text{MS} \cdot \texttt{"server finished"} \cdot \text{R} \cdot \text{H}$
$\text{key\_block}(\text{MS}, \text{R}) = \text{MS} \cdot \texttt{"key expansion"} \cdot \text{R}$
$\text{alert}(\text{level}, \text{descr}) = \text{level} \cdot \text{descr}$
$\text{change\_cipher}() = \mathsf{byte}(1)$

Figure 4. Formats of TLS (omitting variants with optional extensions).

the sub-algebra $\mathbb{D}'$ that does not use directly low-level operators for message structuring (but uses the abstract formats) is isomorphic to the free algebra. We then show that if we consider a communicating system where all honest agents are using abstract formats (that are un-ambiguous and pairwise disjoint), and assuming attacks can be reduced to satisfiability of $K \vdash t$ intruder constraints, we can recast any attack in $\mathbb{D}$ to one in $\mathbb{D}'$. It is thus sound to verify protocols in a free algebra with abstract formats and without considering low-level structuring primitives—we can thus use existing protocol verification tools without modification.

Closely related to our result are some soundness results for typed models [8], [14]–[16]: these results essentially prove that under certain conditions it is sound to bound the depth of terms that the intruder can create. This ensures termination in ProVerif and improves the efficiency of tools like SATMC. These approaches are ideally suited for the combination with our result: they assume, on the Dolev-Yao level, the disjointness of certain message terms; our model similarly requires disjointness of formats on the implementation level and proves the soundness of a Dolev-Yao-style abstraction where the different formats are represented as disjoint, free symbols form_i. An interesting point is here that for typing results we cannot allow unstructured terms under cryptographic operators as in $\mathsf{crypt}(k, x)$ because an agent receiving this message would accept any term for x. Rather, the protocol would have to use some format in place of x to ensure a unique interpretation of this information. In a similar way, compositional reasoning combines well with our results as also here disjointness of message formats is at the core of the assumptions [17]–[20].

There are several works that use Dolev-Yao-style models with a term algebra modulo some algebraic properties of cryptographic operators, e.g., in modular exponentiation [21]. In principle it seems possible to extend our results with some properties of cryptographic operators, but it seems to require changes to some of the semantic proofs and we leave this question for future work.

Several works have used the integration of more algebraic properties into Dolev-Yao-style models in order to shrink the gap between formal model and implementation with regard to the structure of messages. CL-AtSe is to our knowledge the only verification tool in this field that fully supports an associative concatenation operator [22]. Also, the use of associate-commutative operators has been used to model and automatically verify protocols with XML-style messages [23]. A problem with these approaches is that the models still have no notion of message lengths or format encodings. This leads to a large number of false positives, since when applied to abstract terms, the structure often allows mis-association that would not work in a real implementation (due to message lengths or encodings). Moreover,

Client A generates R_A

$A \to B:$ $\text{RECORD}(\mathsf{byte}(22), \text{client\_hello}(time, R_A, \epsilon, cipher, compr)$

Server B generates R_B and ID

$B \to A:$ $\text{RECORD}(\mathsf{byte}(22), \text{server\_hello}(time', R_B, ID, cipher, compr),$
$\text{RECORD}(\mathsf{byte}(22), \text{server\_cert}(\mathsf{sign}(\mathsf{priv}(key(ca)), x509(B, \mathsf{pub}(key(B))))),$
$\text{RECORD}(\mathsf{byte}(22), \text{server\_hello\_done}())$

A checks the certificate (assuming here ca is a trusted certificate authority that A knows the public key of)

A extracts the public key of B and generates the pre-master secret PMS.

Compute master secret $MS = PRF(\text{master\_form}(PMS, RA + RB))$

$A \to B:$ $\text{RECORD}(\mathsf{byte}(22), \text{client\_key\_ex}(\mathsf{crypt}(\mathsf{pub}(key(B)), \text{PMS\_form}(PMS))))$
$\text{RECORD}(\mathsf{byte}(20), \text{change\_cipher}())$
$\text{RECORD}(\mathsf{byte}(22), \text{finished}(PRF(\text{client\_finished}(MS, RA + RB, h(prev\_msgs)))))$

$B \to A:$ $\text{RECORD}(\mathsf{byte}(20), \text{change\_cipher}())$
$\text{RECORD}(\mathsf{byte}(22), \text{finished}(PRF(\text{server\_finished}(MS, RA + RB, h(prev\_msgs)))))$

A and B compute the keys $clientK = \text{extractCK}(\text{key\_block}(MS, RA + RB))$

and $serverK = \text{extractCK}(\text{key\_block}(MS, RA + RB))$

A and B exchange payload messages as follows:

$A \to B:$ $\text{RECORD}(\mathsf{byte}(23), \mathsf{scrypt}(clientK, PAYLOAD))$

$B \to A:$ $\text{RECORD}(\mathsf{byte}(23), \mathsf{scrypt}(serverK, PAYLOAD))$

Figure 5. Typical run of TLS in Alice and Bob notation.

one cannot really be sure that all properties that are relevant for an attack are actually captured. This is the very reason why our approach starts with an algebra $\mathbb{D}$—without giving any concern to automation at this point—that models exactly the equalities of terms that the real implementation has, except the ones that depend on the cryptography. Also, our soundness result allows us to keep the verification process free from all the complications of algebraic reasoning (e.g., that unification modulo an associative operator is infinitary).

The idea of abstract formats is not new. Most prominently, the TulaFale tool uses a connection between XML-formats of the concrete protocol and abstract symbols for that format in the model used in ProVerif [24]. There is no proof of soundness, probably since in a limited set of XML-formats, it seems intuitively clear that nothing can go wrong. Our result proves this formally and in a more general setting (i.e., including also data-structure-style message formats).

In the "more cryptographic" literature, we find many proofs of security protocols that consider no abstract Dolev-Yao model but rather actual cryptographic algorithms and reduce security to some (with high certainty) intractable problems. Many of these works do not put much focus on the non-cryptographic aspects, since these are problems that can *somehow* be solved. When the details are considered, we often have the limitation that the proofs are hard to generalize [25]. When an implementation detail is changed, it may have repercussions throughout the proof. In contrast, our approach is modular: when the implementation of a format is changed without changing the abstract form_i (i.e., changing parameters) then all we need to check is that the new implementation is still un-ambiguous and disjoint from the other formats; the abstract verification on the Dolev-Yao level does not need to be repeated.

In general, the verification on the cryptographic/implementation level is much harder than on the abstract Dolev-Yao level which is much easier for automated and interactive verification, compositional reasoning and refinement. In this spirit, several works give computational soundness results linking the abstract Dolev-Yao models with cryptographic models [1], [2]. One may see our work as the missing piece in this area, systematically studying the structure of messages for large classes of implementations. Since we abstracted from cryptography, an interesting question for future work would be whether one can combine our soundness result with the computational soundness results. Our notion of forms is in fact close to the notion of transparent functions in [26] and a combination could obtain soundness for collections of very diverse protocol formats.

As seen with the optional extensions in TLS, the real world formats are sometimes beyond what can be conveniently expressed. Also, the formats we consider in this paper are composed from a fixed number of elements. There are however several applications for messages of an

unbounded number of elements, e.g., for certification one may supply a chain of certificates (in a single message) and its length should not be bounded. Most tools do not support such *open-ended* messages, but first results exists [27]. More generally, using dependent types to formalize more complex formats with options, repetitions, or cardinality constraints are interesting question for future work.

ACKNOWLEDGMENT

This work was partially supported by the EU FP7 Project no. 318424, "FutureID: Shaping the Future of Electronic Identity" (futureid.eu). We thank Omar Almousa and Luca Viganò for inspiring discussions and comments.

REFERENCES

[1] V. Cortier, S. Kremer, and B. Warinschi, "A survey of symbolic methods in computational analysis of cryptographic systems," *J. Autom. Reasoning*, vol. 46, no. 3-4, pp. 225–259, 2011.

[2] M. Backes, A. Malik, and D. Unruh, "Computational soundness without protocol restrictions," in *ACM Conference on Computer and Communications Security*, 2012, pp. 699–711.

[3] C. J. F. Cremers, "Feasibility of multi-protocol attacks," in *ARES*, 2006, pp. 287–294.

[4] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *CSFW*, 2001, pp. 82–96.

[5] A. Armando, W. Arsac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuéllar, G. Erzse, S. Frau, M. Minea, S. Mödersheim, D. von Oheimb, G. Pellegrino, S. E. Ponta, M. Rocchetto, M. Rusinowitch, M. T. Dashti, M. Turuani, and L. Viganò, "The avantssar platform for the automated validation of trust and security of service-oriented architectures," in *TACAS*, 2012, pp. 267–282.

[6] M. Rusinowitch and M. Turuani, "Protocol insecurity with finite number of sessions is np-complete," in *CSFW*, 2001.

[7] J. K. Millen and V. Shmatikov, "Constraint solving for bounded-process cryptographic protocol analysis," in *Proceedings of CCS'01*. ACM Press, 2001, pp. 166–175.

[8] S. Mödersheim, "Deciding security for a fragment of aslan," in *ESORICS*, 2012, pp. 127–144.

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

[10] L. C. Paulson, "Inductive Analysis of the Internet Protocol TLS," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 3, pp. 332–351, 1999.

[11] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron, "The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications," in *Proceedings of CAV'05*, ser. LNCS 3576. Springer, 2005, pp. 281–285. [Online]. Available: http://www.avispa-project.org

[12] R. Seggelmann, M. Tuexen, and M. Williams, "RFC6520: Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension," 2012. [Online]. Available: http://tools.ietf.org/rfc/rfc6520.txt

[13] D. Eastlake, "RFC6066: Transport Layer Security (TLS) Extensions: Extension Definitions," 2011. [Online]. Available: http://tools.ietf.org/rfc/rfc6066.txt

[14] J. Heather, G. Lowe, and S. Schneider, "How to prevent type flaw attacks on security protocols," in *Proceedings of CSFW'00*. IEEE Computer Society Press, 2000.

[15] M. Arapinis and M. Duflot, "Bounding messages for free in security protocols," in *FSTTCS*, 2007, pp. 376–387.

[16] B. Blanchet and A. Podelski, "Verification of cryptographic protocols: tagging enforces termination," *Theor. Comput. Sci.*, vol. 333, no. 1-2, pp. 67–90, 2005.

[17] S. Ciobâcă and V. Cortier, "Protocol composition for arbitrary primitives," in *Proceedings of CSF'10*, 2010.

[18] J. D. Guttman, "Authentication tests and disjoint encryption: a design method for security protocols," *Journal of Computer Security*, vol. 3–4, no. 12, pp. 409–433, 2004.

[19] S. Mödersheim and L. Viganò, "Secure pseudonymous channels," in *Proceedings of ESORICS 14*, ser. LNCS 5789. Springer, 2009, pp. 337–354.

[20] T. Groß and S. Mödersheim, "Vertical protocol composition," in *CSF*, 2011, pp. 235–250.

[21] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani, "Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents," in *FST TCS'03*, ser. LNCS 2914, 2003, pp. 124–135.

[22] M. Turuani, "The CL-Atse Protocol Analyser," in *Proceedings of RTA*, ser. LNCS 4098. Springer-Verlag, 2006, pp. 277–286.

[23] Y. Chevalier, D. Lugiez, and M. Rusinowitch, "Towards an automatic analysis of web service security," in *FroCoS*, 2007, pp. 133–147.

[24] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella, "Tulafale: A security tool for web services," in *FMCO*, 2003, pp. 197–222.

[25] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner, "Cryptographically Sound Theorem Proving," in *Proceedings of CSFW 19*. IEEE Computer Society Press, 2006, pp. 153–166.

[26] F. Böhl, V. Cortier, and B. Warinschi, "Deduction soundness: prove one, get five for free," in *ACM Conference on Computer and Communications Security*, 2013, pp. 1261–1272.

[27] B. Blanchet and M. Paiola, "Automatic verification of protocols with lists of unbounded length," in *ACM Conference on Computer and Communications Security*, 2013, pp. 573–584.