Verifying the Independence of Security Protocols

Genge Bela
"Petru Maior" University of Tg. Mures
Electrical Engineering Department
16, N. Grigorescu St., Tg. Mures, Romania
bgenge@upm.ro

Iosif Ignat
Technical University of Cluj-Napoca
Computer Science Department
28, Gh. Baritiu St., Cluj-Napoca, Romania
Iosif.Ignat@cs.utcluj.ro

Abstract

Determining if two protocols can be securely run alongside each other requires analyzing the independence of the involved protocols. In this paper we construct a canonical model of security protocols that allows us to conduct a syntactical analysis on the independence of multiple security protocols. By integrating participant knowledge in the model, we are able to detect subtle multi-protocol attacks, where the types of certain message components can not be checked, also known as type-flaw attacks. Of special interest is the construction of messages in the proposed model, which is made by mapping each message component from the regular specification to a type. We provide a theorem for analyzing the independence of security protocols and illustrate its applicability by analyzing two protocols.

1. Introduction

Security protocols are communication protocols in which cryptography is used to give participants the capability to transmit encoded information that can be only decoded by the designated receivers. These protocols have been intensively analyzed throughout the last few decades, resulting in a variety of dedicated formal methods and tools [1, 5, 7, 8, 13, 14, 16]. The majority of these methods consider a Dolev-Yao-like penetrator model [6] to capture the actions available to a penetrator which has complete control over the network. By analyzing each individual protocol in the presence of this penetrator model, the literature has reported numerous types of attacks [3, 8]. However, in practice, there can be multiple protocols running over the same network, thus the penetrator is given new opportunities to construct attacks by combining messages from several protocols, also known as multi-protocol attacks [10].

Multi-protocol attacks raise the fundamental question of determining if two protocols can run *safely* alongside each other in the same system. The answer can be found by verifying if the involved protocols are *independent*, namely that they do not influence each other's security properties. However, this can be rather difficult to verify if we consider that every protocol can have multiple instances. In addition, by simply inspecting the usual protocol specification, we might miss attacks based on limited participant knowledge, also known as type-flaw attacks [5], where the type of certain message components can not be checked.

To simplify the verification process, in this paper we provide a canonical model for analyzing the independence of multiple security protocols. In the first step we enrich the regular specification with participant knowledge. This allows us to clearly separate the components that can be validated by protocol participants from the ones that can not. Based on this representation, in the next step we construct a canonical model, also called a *typed model*, that replaces each message component with its corresponding type, marking out the message structures that directly influence the independence of protocols.

The rest of the paper is structured as follows. In Section 2 we extend the regular specification of security protocols with knowledge. In Sections 3 and 4 we construct the proposed *typed model* based on the extended specification. We provide our independence theorem in Section 5. In Section 6 we relate our work to others found in the literature. We end with a conclusion in Section 7.

2. Extending the regular specification

In the regular strand-based specification [1], protocol participants are modeled using *strands*, where a *strand* denotes a sequence of message transmissions. In this section we extend this specification with participant knowledge, which plays a major role in the construction of the *typed model* from the next sections. From the diversity of specifications, we have chosen to use and extend the strand-based specification mainly because of its flexibility, which has made other authors in the past to extend and apply it in multiple directions [1, 2, 4].

To achieve our goal, first, we specialize the set of terms from the original model using *basic sets* and *function names*. By doing so, we are able to map each term to its corresponding type in the constructions that follow. We continue with the definition of regular *strands*, *knowledge strands* and *penetrator strands*.

2.1. Regular and knowledge strands

Roles (i.e. protocol participants) communicate by exchanging *terms* constructed from elements belonging to the following basic sets: R, denoting the set of role names; N, denoting the set of nonces (i.e. "number once used"); K, denoting the set of cryptographic keys.

To denote the encryption type used to create cryptographic terms, we define the following *function names*:

$$\begin{aligned} FuncName ::= {} & sk && (secret\, key) \\ \mid {} & pk && (public\, key) \\ \mid {} & pvk && (private\, key) \\ \mid {} & h && (hash) \end{aligned}$$

Here, and in the rest of the paper, sk, pk, pvk and h range across secret (i.e. symmetric key-based encryption), public (i.e. asymmetric key-based encryption), private (i.e. private key encryption, also known as "digital signature") and hash function names respectively.

The above-defined basic sets and function names are used in the definition of *terms*, where we also introduce constructors for pairing and encryption:

$$\begin{aligned} \mathcal{T} ::= {} & . \mid \mathsf{R} \mid \mathsf{N} \mid \mathsf{K} \mid (\mathcal{T}, \mathcal{T}) \\ & \mid \{\mathcal{T}\}_{FuncName(\mathcal{T})} \end{aligned}$$

where the '.' symbol is used to denote an empty term.

The terms that can be constructed by roles, based on the definitions above, include the basic components of a wide variety of security protocols. However, these can be further extended if the modeled protocol requires other components.

The composition process of two terms t_1 and t_2 into another term t implies that t has sub-terms. The subterm relation $\sqsubset$ is inductively defined as follows.

Definition 1. *The subterm relation $\sqsubset$ is the smallest relation on terms such that:*

1. $t \sqsubset t$;
2. $t \sqsubset \{t_1\}_{f(t_2)}$ *if* $t \sqsubset t_1 \vee t \sqsubset t_2$;
3. $t \sqsubset (t_1, t_2)$ *if* $t \sqsubset t_1 \vee t \sqsubset t_2$.

Two terms are equal if all their subterms are equal.

Having defined the terms exchanged by participants, we can proceed with the definition of a *strand* and a *strand space* (i.e. a collection of strands). To capture the sending and receiving of terms, the strand model from [1] introduces *signed terms*. The occurrence of a term with a positive sign denotes transmission, while the occurrence of a term with a negative sign denotes reception.

Definition 2. *A* signed term *is a pair $\langle \sigma, t\rangle$ with $t \in \mathcal{T}$ and σ one of the symbols $+, -$. A signed term is written as $-t$ or $+t$. $(\pm\mathcal{T})^*$ is the set of finite sequences of signed terms. A typical element of $(\pm\mathcal{T})^*$ is denoted by $\langle \pm t_1, \pm t_2, \ldots, \pm t_n\rangle$, with $t_i \in \mathcal{T}$.*

Definition 3. *A* strand *is a sequence of term transmissions and receptions, represented as $\langle \pm t_1, \pm t_2, \ldots, \pm t_n\rangle \in (\pm\mathcal{T})^*$. A set of strands is called a* strand space *and is denoted by Σ.*

1. *A* node *is any transmission or reception of a term, written as $n = \langle s, i\rangle$, with $s \in \Sigma$ and i an integer satisfying the condition $1 \leq i \leq length(s)$, where $length(s)$ is a function returning the number of nodes from a strand. We say that the node $n = \langle s, i\rangle$ belongs to the strand s and $strand(n) = s$. The set of all nodes is denoted by $\mathcal{N}$.*
2. *Let $n_1 = \langle s, i\rangle$ and $n_2 = \langle s, i+1\rangle$ be two consecutive nodes from $\mathcal{N}$ on the same strand $s \in \Sigma$. Then, there exists an edge $n_1 \Rightarrow n_2$ in the same strand.*
3. *Let $n_1, n_2 \in \mathcal{N}$. If n_1 is a positive node and n_2 is a negative node and $strand(n_1) \neq strand(n_2)$, then there exists an edge $n_1 \rightarrow n_2$.*
4. *$\mathcal{N}$ together with both sets of edges $n_1 \Rightarrow n_2$ and $n_1 \rightarrow n_2$ is a directed graph $\langle \mathcal{N}, (\Rightarrow \cup \rightarrow)\rangle$.*

Definition 4. *Let $\rightarrow_{\mathcal{C}} \subset \rightarrow$, $\Rightarrow_{\mathcal{C}} \subset \Rightarrow$ and $\mathcal{C} = \langle \mathcal{N}_{\mathcal{C}}, (\Rightarrow_{\mathcal{C}} \cup \rightarrow_{\mathcal{C}})\rangle$ be an acyclic subgraph of $\langle \mathcal{N}, (\Rightarrow \cup \rightarrow)\rangle$. Then $\mathcal{C}$ is a bundle if:*

1. *whenever $n_2 \in \mathcal{N}_{\mathcal{C}}$ receives a term, there exists a unique n_1 such that $n_1 \rightarrow_{\mathcal{C}} n_2$;*
2. *whenever $n_2 \in \mathcal{N}_{\mathcal{C}}$ and $n_1 \Rightarrow n_2$, then $n_1 \Rightarrow_{\mathcal{C}} n_2$*

When running a protocol, roles must act according to the information extracted from the received terms. However, this is only possible if roles can *decide* upon the validity of the term. The decision can be based on a known pattern that must be respected by terms (e.g. a role name has the form someuser@someknownhost.com) or on the fact that the term value is already known to be valid. The information based on which a is called *role knowledge*.

We model role knowledge as a set of terms attached to the strand corresponding to a role. Thus, we extend the regular strands from [1], enriching them with role knowledge, resulting a new type of strand that we call *knowledge strand*.

Definition 5. Role knowledge *is defined as a set $\kappa \in \mathcal{T}^*$. A* k-strand *(knowledge strand) is a tuple $\langle \kappa, s \rangle$, with knowledge $\kappa \in \mathcal{T}^*$ and the strand $s \in \Sigma$. A set of k-strands is called a* k-strand space*. The set of all k-strand spaces is denoted by Σ_κ^*.*

2.2. Honest and penetrator k-strands

The proofs from the following sections consider the existence of multiple k-strand spaces (i.e. multiple security protocols) and a *penetrator k-strand space*, Σ_P. *Penetrator k-strands* model the atomic actions available to the penetrator. If a k-strand does not belong to Σ_P, it is called an *honest k-strand.*

We consider a Dolev-Yao-like penetrator model [6], where the penetrator has full control over the network. It can inject, block, alter messages and it is capable of encrypting and decrypting messages if the right key is in his possession.

We specialize the set K to distinguish between the keys considered to be safe, i.e. $\mathsf{K}_S \subseteq \mathsf{K}$, and the keys known to the penetrator, i.e. $\mathsf{K}_P \subseteq \mathsf{K}$, where K_S and K_P are disjoint.

By eavesdropping over the network or by inspecting the terms belonging to previous sessions that have been compromised, the penetrator can obtain valid terms. The set of terms known to the penetrator is denoted by T_P.

Let $Keys : \Sigma_\kappa^* \rightarrow \mathsf{K}^*$ be a function returning the set of cryptographic keys used in a k-strand space and let $Safe : \Sigma_\kappa^* \rightarrow \mathsf{K}_S^*$ be a function returning the set of safe keys used in k-strand space. Here and in the rest of the paper, K^* denotes the set of all subsets of keys and K_S^* denotes the set of all subsets of safe keys. $Safe(\Sigma_\kappa) \subseteq Keys(\Sigma_\kappa)$, where Σ_κ is a k-strand space.

Definition 6. *A* penetrator k-strand *is one of the following (with $t, t_1, t_2 \in \mathsf{T}_P$, $k \in \mathsf{K}_P$* and *$f \in FuncName$):*

M. *Text message* $\langle \langle\rangle, +t \rangle$
F. *Flushing* $\langle \langle\rangle, \langle -t \rangle \rangle$
T. *Tee* $\langle \langle\rangle, \langle -t, +t, +t \rangle \rangle$
C. *Concatenation* $\langle \langle\rangle, \langle -t_1, -t_2, +(t_1, t_2) \rangle \rangle$
S. *Separation* $\langle \langle\rangle, \langle -(t_1, t_2), +t_1, +t_2 \rangle \rangle$
K. *Key* $\langle \langle\rangle, \langle +k \rangle \rangle$
E. *Encryption* $\langle \langle\rangle, \langle -k, -t, +\{t\}_{f(k)} \rangle \rangle$
D. *Decryption* $\langle \langle\rangle, \langle -k^{-1}, -\{t\}_{f(k)}, +t \rangle \rangle$

As it can be seen, penetrator k-strands do not include any local knowledge. However, there is a global knowledge available to all penetrator k-strands in the form of the two sets, K_P and T_P.

3. Typed strands

In this section we construct a canonical model for security protocols, based on the k-strand space model from the previous section and the *types* defined in this section. The *types* we consider are syntactic constructions used to create other syntactic expressions, such as *typed terms*. This results in a *typed model* that captures the structure of security protocol messages, which plays a major role in the verification process of security protocol independence.

3.1. Typed terms and strands

Typed terms form the basis of the proposed typed model. These are created by applying term forming constructs, defined below, to *basic typed terms* and *function names*. The function names we consider were already defined in the previous section, so we only need to define the *basic typed terms*.

Our notion of a *basic typed term* is formalized using the following grammatical productions:

$$\begin{aligned} BasicTT ::= {} & \mathsf{K}_t && (typed\,keys) \\ \mid {} & r && (role\,type) \\ \mid {} & n && (nonce\,type) \\ \mid {} & u && (unknown\,type) \end{aligned}$$

In the above definition and in the rest of the paper, K_t, r, n and u range over typed keys, role types, nonce types and unknown types respectively.

In the encryption process of the same plaintext, the use of two different keys, K_1 and K_2, will produce two different ciphertexts. This is also true for the decryption process, where the use of two different keys results in two different plaintexts. Because of this, we consider that the type of the encrypted terms after decryption will change too, according to the keys that are used. Thus, we use an indexed key type k_i, such that $k_i \neq k_j$, where $i \neq j$, to distinguish between key types corresponding to different keys. In the definition of $BasicTT$, the set of all typed keys is denoted by K_t.

The *unknown* type corresponds to terms that can not be validated because of limited role knowledge. By including this information in the specification we are able to detect subtle type-flaw attacks using a syntactical comparison of typed terms, that otherwise would require the construction of a state-space that can become rather large if we consider the existence of multiple protocols in the same system.

For the definition of *typed terms* we introduce constructors for pairing and encryption:

$$\begin{aligned} \mathcal{T}_t ::= {} & . \mid BasicTT \mid (\mathcal{T}_t, \mathcal{T}_t) \\ & \mid \{\mathcal{T}_t\}_{FuncName(\mathcal{T}_t)} \end{aligned}$$

where the '.' symbol is used to denote an empty typed term.

We define a t-subterm relation $\sqsubset_t$ on typed terms as follows.

Definition 7. *The t-subterm relation $\sqsubset_t$ is the smallest relation on typed terms such that:*

1. *$t \sqsubset_t t$;*
2. *$t \sqsubset_t \{t_1\}_{f(t_2)}$ if $t \sqsubset_t t_1 \vee t \sqsubset_t t_2$;*
3. *$t \sqsubset_t (t_1, t_2)$ if $t \sqsubset_t t_1 \vee t \sqsubset_t t_2$.*

Two typed terms are equal if all their t-subterms are equal.

In the *typed strand* model, participants exchange typed terms instead of regular terms. In fact, *typed strands* are defined similarly to regular strands.

Definition 8. *A* signed typed term *is a pair $\langle \sigma, t \rangle$ with $t \in \mathcal{T}_t$ and σ one of the symbols $+, -$. A signed typed term is written as $-t$ or $+t$. $(\pm\mathcal{T}_t)^*$ is the set of finite sequences of signed typed terms. A typical element of $(\pm\mathcal{T}_t)^*$ is denoted by $\langle \pm t_1, \pm t_2, \ldots, \pm t_n \rangle$, with $t_i \in \mathcal{T}_t$.*

Definition 9. *A* t-strand *(typed strand) is a sequence of typed term transmissions and receptions, represented as $\langle \pm t_1, \pm t_2, \ldots, \pm t_n \rangle \in (\pm\mathcal{T}_t)^*$. A collection of t-strands is called a* t-strand space *and is denoted by Σ_t. The set of all t-strands is denoted by Σ_t^*.*

1. *A* typed node *is any transmission or reception of a typed term, written as $n_t = \langle s_t, i \rangle$, with $s_t \in \Sigma_t$ and i an integer satisfying the condition $1 \leq i \leq tlength(s_t)$, where $tlength(s_t)$ is a function returning the number of typed nodes from a t-strand. We say that the typed node $n_t = \langle s_t, i \rangle$ belongs to the t-strand s_t and $tstrand(n_t) = s_t$. The set of all typed nodes is denoted by $\mathcal{N}_t$. If the context allows us, a typed node is simply called a node.*
2. *Let $n_{t1} = \langle s_t, i \rangle$ and $n_{t2} = \langle s_t, i+1 \rangle$ be two consecutive nodes from $\mathcal{N}_t$ on the same strand $s_t \in \Sigma_t$. Then, there exists an edge $n_{t1} \Rightarrow n_{t2}$ on the same t-strand.*
3. *Let $n_{t1}, n_{t2} \in \mathcal{N}_t$. If n_{t1} is a positive node and n_{t2} is a negative node and $tstrand(n_{t1}) \neq tstrand(n_{t2})$, then there exists an edge $n_{t1} \rightarrow n_{t2}$.*
4. *$\mathcal{N}_t$ together with both sets of edges $n_{t1} \Rightarrow n_{t2}$ and $n_{t1} \rightarrow n_{t2}$ is a directed graph $\langle \mathcal{N}_t, (\Rightarrow \cup \rightarrow) \rangle$.*

Definition 10. *Let $\rightarrow_{\mathcal{C}_t} \subset \rightarrow$, $\Rightarrow_{\mathcal{C}_t} \subset \Rightarrow$ and $\mathcal{C} = \langle \mathcal{N}_{\mathcal{C}_t}, (\Rightarrow_{\mathcal{C}_t} \cup \rightarrow_{\mathcal{C}_t}) \rangle$ be an acyclic subgraph of $\langle \mathcal{N}_t, (\Rightarrow \cup \rightarrow) \rangle$. Then $\mathcal{C}_t$ is a t-bundle if:*

1. *whenever $n_2 \in \mathcal{N}_{\mathcal{C}_t}$ receives a term, there exists a unique n_1 such that $n_1 \rightarrow_{\mathcal{C}_t} n_2$;*
2. *whenever $n_2 \in \mathcal{N}_{\mathcal{C}_t}$ and $n_1 \Rightarrow n_2$, then $n_1 \Rightarrow_{\mathcal{C}_t} n_2$*

Example. Consider Lowe's modified version of the BAN concrete Andrew Secure RPC [3]:

$$\begin{aligned} A \rightarrow B &: A, N_a \\ B \rightarrow A &: \{N_a, K, B\}_{K_{AB}} \\ A \rightarrow B &: \{N_a\}_K \\ B \rightarrow A &: N_b \end{aligned}$$

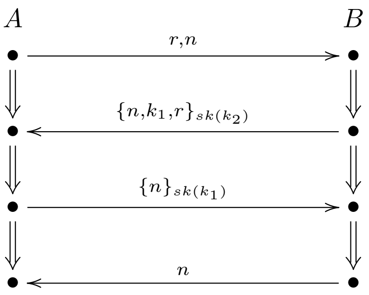

Figure 1: Lowe's BAN concrete Andrew Secure RPC protocol representation in the proposed typed strand model.

By running the protocol, the two parties, A and B, establish a fresh session key K. The nonce N_a ensures freshness of the newly generated key and the nonce N_b is sent by B to be used in future sessions.

The specification of Lowe's protocol in the typed strand model is presented in Figure 1. Role A is modeled as the t-strand $\langle +(r,n), -(\{n, k_1, r\}_{sk(k_2)}),\ +(\{n\}_{sk(k_1)}), -(n) \rangle$ and role B is modeled as the t-strand $\langle -(r,n), +(\{n, k_1, r\}_{sk(k_2)}), -(\{n\}_{sk(k_1)}), +(n) \rangle$.

This typed representation has been constructed intuitively, by replacing each component with it's corresponding type. However, in Section 4 we present several function mappings used to transform a k-strand specification into a t-strand specification.

3.2. Typed honest and penetrator strands

The atomic actions available to the penetrator in the typed model are identical to the ones from the k-strand model. The only difference is that instead of dealing with terms, the penetrator now uses typed terms to construct security protocol attacks.

The set of t-strands belonging to the penetrator is called the *penetrator t-strand space* and is denoted by Σ_{tP}. If a typed strand does not belong to Σ_{tP}, it is called a *typed honest strand*.

We further specialize the set of typed keys K_t with two disjoint sets, $\mathsf{K}_{tS} \subseteq \mathsf{K}_t$, denoting the set of safe typed keys and $\mathsf{K}_{tP} \subseteq \mathsf{K}_t$, denoting the set of typed keys known by the penetrator. Additionally, a penetrator may possess typed terms, denoted by the set T_{tP}.

Let $TKeys : \Sigma_t^* \rightarrow \mathsf{K}_t^*$ be a function returning the set of all typed keys and let $TSafe : \Sigma_t^* \rightarrow \mathsf{K}_{tS}^*$ be a function returning the set of typed keys considered to be safe in a typed strand space. Here and in the rest of the paper, K_t^* denotes the set of all subsets of typed keys and K_{tS}^* denotes the set of all subsets of typed safe keys.

Similarly to the k-strand space model, we have $TSafe(\Sigma_t) \subseteq TKeys(\Sigma_t)$, where Σ_t is a typed strand space.

4. Strand mappings and virtual strands

As seen in the previous sections, the typed model of a security protocol can only be constructed by relating each term to it's corresponding type. To ensure a proper transformation, we define several mapping functions from the k-strand model to the typed strand model of a protocol.

In the transformation process of a knowledge strand to a typed strand, term components that are not part of the knowledge are replaced with an unknown type u and terms that are part of the knowledge, are transformed according to the TTr (Term Transformation) function, $TTr : \mathcal{T} \to \mathcal{T}_t$, defined as:

$$TTr(t) = \begin{cases} r, & \text{if } t \in \mathsf{R}\ , \\ n, & \text{if } t \in \mathsf{N}, \\ k_i, & \text{if } t \in \mathsf{K}, \\ (TTr(t_1), TTr(t_2)), & \text{if } t = (t_1, t_2), \\ \{TTr(t_1)\}_{f(TTr(t_2))}, & \text{if } t = \{t_1\}_{f(t_2)}. \end{cases}$$

where $t, t_1, t_2 \in \mathcal{T}$ and $f \in FuncName$.

To transform the terms transmitted or received by a k-strand into typed terms, we use the $KTTr$ (Knowledge Term Transformation) function, $KTTr : \mathcal{T}^* \times \mathcal{T} \to \mathcal{T}_t$, defined as:

$$KTTr(\kappa, t) = \begin{cases} (KTTr(\kappa, t_1), KTTr(\kappa, t_2)), & \text{if } t = (t_1, t_2), \\ \{KTTr(\kappa, t_1)\}_{f(KTTr(\kappa,t_2))}, & \text{if } t = \{t_1\}_{f(t_2)}, \\ TTr(t), & \text{if } t \in \kappa, \\ u, & \text{otherwise.} \end{cases}$$

where κ denotes the knowledge corresponding to a strand and t is the transformed term.

When applying the $KTTr$ transformation function on each term found in a collection of strands, if the knowledge of interacting strands is different, the resulting typed strands will not satisfy Clause 3 from the definition of typed strands. To solve this problem we introduce the concept of *virtual typed strands*, such that for each typed strand denoting a protocol participant, there is a corresponding virtual typed strand that acts as a transformation layer for all the received and transmitted typed terms.

Virtual typed strands (for simplicity also called *virtual strands*) and typed strands are constructed from knowledge strands, using transformation functions. Let $s_\kappa = \langle \kappa, s \rangle$ be a knowledge strand, with $s = \langle \pm t_1, \pm t_2, \ldots, \pm t_n \rangle$, $t_i \in \mathcal{T}, s \in (\pm\mathcal{T})^*$ and $n = length(s)$. Then the two functions, $KSTr, VSTr : \mathcal{T}^* \times (\pm\mathcal{T})^* \to (\pm\mathcal{T}_t)^*$, denoting a transformation function from a knowledge strand to a typed strand and a transformation function from a knowledge strand to a virtual strand respectively, are defined as:

$$KSTr(s_\kappa) = \langle \pm KTTr(\kappa, t_1), \pm KTTr(\kappa, t_2), \ldots, \pm KTTr(\kappa, t_n) \rangle$$

$$VSTr(s_\kappa) = \begin{cases} \langle VSTr(\kappa, \pm t_1), \ldots, VSTr(\kappa, \pm t_n) \rangle, & \text{if } s_\kappa = \langle \kappa, \langle \pm t_i \rangle \rangle, \\ \langle -t, +KTTr(\kappa, t) \rangle, & \text{if } s_\kappa = \langle \kappa, -t \rangle, \\ \langle -t, +t \rangle, & \text{if } s_\kappa = \langle \kappa, +t \rangle. \end{cases}$$

The following two propositions relate to the process of key transformation from one model to another. The first one states that if a key is safe, then the transformed key will also be safe. The second one states that two different keys from the k-strand model have different correspondents in the t-strand model.

Proposition 11. *Let K_S be the set of safe keys in the k-strand model and let K_{tP} be the set of keys known by the penetrator in the t-strand model. If $K \in \mathsf{K}_S$, then $TTr(K) \notin \mathsf{K}_{tP}$.*

Proposition 12. *Let Σ_κ and Σ'_κ be two k-strand spaces, with Σ_t and Σ'_t being their corresponding typed strand spaces. If $K_i \in Keys(\Sigma)$ and $K_j \in Keys(\Sigma')$, with $K_i \neq K_j$, then $TTr(K_i) \neq TTr(K_j)$.*

5. Protocol independence

In this section we prove that if certain conditions are met, protocols can be run in parallel without affecting each other's security properties. First, we provide a protocol independence theorem and then exemplify the applicability of the theorem accompanied by the proposed typed model by analyzing the independence of two security protocols.

5.1. Protocol independence theorem

The process of determining if two protocols are independent, starts with the examination of the security properties that each protocol must meet when run in isolation. The security properties that protocols must meet fall under one or multiple basic properties: *authentication, secrecy, integrity* and *non-repudiation* [12]. Intuitively, *authentication* denotes the process by which the communicating parties are ensured that they are talking to the right participant. The *secrecy* property guarantees participants that a value used in a protocol is not revealed to others. *Integrity* and *non-repudiation* are guaranteed by cryptographic operations such as hash functions or digital signatures.

By inspecting the mentioned security properties, we formulate two conditions that must be met so that two security protocols are considered independent (we later discuss each one of them separately):

- The secret values from one protocol must not be revealed by the other protocol, thus preserving the secrecy properties;
- The encrypted messages from the two protocols must be independent of each other, which leads to the non-interference of the other three properties (i.e. authentication, integrity and non-repudiation).

The first condition that must be satisfied by independent protocols states that secret values from one protocol must not be disclosed by the other protocol in question. This condition is satisfied by security protocols in general because session keys and nonces - that usually form the secrets of a protocol - are generated for every protocol run, separately. This condition is in fact similar to the "disjoint encryption" condition from [4]. *Secrecy independence* is formally defined as follows.

Definition 13. *Let Σ_κ and Σ'_κ be two k-strand spaces, with S being the set of secret terms belonging to Σ_κ. Let $SendEncr : \Sigma^*_\kappa \rightarrow \mathcal{T}^*$ be a function returning the set of all encrypted terms transmitted by all k-strands belonging to a k-strand space.*

Σ_κ is secrecy independent *of Σ'_κ if the terms belonging to S are not sent out in clear by the k-strands $s'_\kappa \in \Sigma'_\kappa$ and if $t \sqsubset \{t_1\}_{f(K)}$, then $K \in Safe(\Sigma'_\kappa)$, where $\{t_1\}_{f(K)} \in SendEncr(\Sigma'_\kappa)$.*

Σ_κ is key secrecy independent *of Σ'_κ if Σ_κ is secrecy independent of Σ'_κ and $(Keys(\Sigma'_\kappa) \setminus Safe(\Sigma'_\kappa)) \cap Keys(\Sigma_\kappa) = \phi$, where ϕ denotes an empty set.*

The second condition that must be satisfied by independent protocols states that the penetrator should not be able to construct attacks on one protocol based on encrypted messages originating from a different protocol. The condition includes only encrypted terms because the penetrator can create unencrypted terms using the **M**, **T**, **C** and **S** penetrator strands. We verify this condition by using the typed model, which effectively captures the message structural similarities, including terms that can not be validated by roles.

To formally define the second condition we first need to introduce the $CT : \mathcal{T}_t \times \mathcal{T}_t$ (Constructible) predicate to express if a typed term t can be constructed from another typed term t'. By defining this predicate we are able to reason about terms that can be accepted as valid in a protocol, regardless of where (i.e. other protocol) the terms are originating from.

$$CT(t',t) = \begin{cases} \text{True}, & \text{if } (t = u \wedge t' \in BasicTT) \vee \\ & \quad t = t' \vee (t' = u \wedge t \in BasicTT) \\ CT(t'_1, t_1) \wedge CT(t'_2, t_2), & \\ & \text{if } (t' = (t'_1, t'_2) \wedge t = (t_1, t_2)) \vee \\ & \quad (t' = \{t'_1\}_{f(t'_2)} \wedge t = \{t_1\}_{f(t_2)}) \end{cases}$$

Definition 14. *Let Σ_t and Σ'_t be two typed strand spaces. Let $TSendEncr : \Sigma^*_t \rightarrow \mathcal{T}^*_t$ be a function returning the set of all encrypted terms transmitted by non-virtual typed strands and let $TRecvEncr : \Sigma^*_t \rightarrow \mathcal{T}^*_t$ be a function returning the set of all encrypted terms received by non-virtual typed strands.*

Σ_t is message independent *of Σ'_t if for all $t' \in TSendEncr(\Sigma'_t)$ and for all $t \in TRecvEncr(\Sigma_t)$, the predicate $CT(t',t)$ does not hold.*

Next, we provide the protocol independence theorem through the form of a definition and a proposition.

Definition 15. *Let Σ_t and Σ'_t be two typed strand spaces. Σ_t is* independent *of Σ'_t if there is no bundle $\mathcal{C}_t$ such that $n_P \rightarrow_{\mathcal{C}_t} n_t$ and $n'_t \rightarrow_{\mathcal{C}_t} n'_P$, where $n_t, n'_t, n_P, n'_P \in \mathcal{C}_t$, $tstrand(n_t) \in \Sigma_t$, $tstrand(n'_t) \in \Sigma'_t$ and $tstrand(n_P), tstrand(n'_P) \in \Sigma_{tP}$.*

Proposition 16. *Let Σ_κ, Σ'_κ be two k-strand spaces and Σ_t, Σ'_t their corresponding t-strand spaces. If Σ_κ is key secrecy independent of Σ'_κ and Σ_t is message independent of Σ'_t, then Σ_t is independent of Σ'_t.*

Proof. We want to show that, given the conditions, there is no bundle with a starting node in Σ'_t and an ending node in Σ_t. In other words, we show that the penetrator can not create valid encrypted terms in Σ_t using any terms captured from Σ'_t or replay valid encrypted terms from Σ'_t into Σ_t.

By Definition 13, we may assume that $Keys(\Sigma'_\kappa)$ are either safe or they are not used in cryptographic operations in Σ_κ, i.e. $Keys(\Sigma_\kappa) \cap (Keys(\Sigma'_\kappa) \setminus Safe(\Sigma'_\kappa)) = \phi$. By Propositions 11 and 12 we have that if $K \in Safe(\Sigma'_\kappa)$ then $TTr(K) \in TSafe(\Sigma'_t)$ and that if $K \in Keys(\Sigma'_\kappa)$ such that $K \notin Keys(\Sigma_\kappa)$ then the corresponding typed key $TTr(K) \in TKeys(\Sigma'_t)$ is not used in cryptographic operations in Σ_t, i.e. $TTr(K) \notin TKeys(\Sigma_t)$.

Because of these, $(Keys(\Sigma_t) \setminus Safe(\Sigma_t)) \cap (Keys(\Sigma'_t) \setminus Safe(\Sigma'_t)) = \phi$, thus the penetrator can not get a valid cryptographic key from Σ'_t to create cryptographic terms to be injected in Σ_t. In other words, the typed strands available to the penetrator (**K**, **E** and **D**) can not be used to create cryptographic terms that are valid in Σ_t, based on typed terms captured from Σ'_t.

If the penetrator can not create encrypted typed terms that does not mean it can not generate new terms, separate, concatenate and replay typed terms from one protocol to another using penetrator strands **M**, **F**, **T**, **C** and **S**. However, because Σ_t and Σ'_t are message independent, $TRecvEncr(\Sigma_t)$ does not contain any typed encrypted terms that can be constructed from any typed encrypted terms of $TSendEncr(\Sigma'_t)$. □

Proposition 17. *Let Σ_κ and Σ'_κ be two k-strand spaces and Σ_t, Σ'_t their corresponding t-strand spaces. If Σ_t is independent of Σ'_t, then Σ_κ is independent of Σ'_κ.*

Proof. By Definition 13 we have that the penetrator cannot obtain cryptographic keys from Σ'_{κ} to be used in Σ_{κ}. Thus, the penetrator strands **K**, **E** and **D** can not be used to create valid encrypted terms in Σ_{κ} from any terms leaked from Σ'_{κ}.

Because Σ_t is message independent of Σ'_t, we have that $\forall t_t \in TRecvEncr(\Sigma_t)$ and $\forall t'_t \in TSendEncr(\Sigma'_t)$, $CT(t'_t, t_t)$ does not hold. By following the inverse logic of construction, for each typed term t_t and t'_t there is a set of terms generated by replacing each typed term with all possible corresponding values, resulting t and t', respectively. Because CT does not hold, the typed terms t'_t are not accepted as valid in Σ_t, thus, by construction, the corresponding t' terms will not be accepted as valid in Σ_{κ}. Thus, the remaining penetrator strands **M**, **F**, **T**, **C** and **S** can not be used to replay valid terms into Σ_{κ} using terms captured from Σ'_{κ}. □

5.2. Applying the independence theorem

Because of space limitations we illustrate the applicability of the protocol independence theorem by using simplified versions of the Yahalom-Lowe (Y-L) and Kao-Chow (K-C) protocols (both chosen from the SPORE library [11]), modeled in the k-strand space as Σ_{YL} and Σ_{KC} respectively. A graphical representation of these protocols can be seen in Figure 2, where the encryption function is the same for all cryptographic terms ($f = sk$) and has been omitted from the representation for simplicity.

The terms considered to be secret are N_b, K_{AB} for the Y-L protocol and K'_{AB} for the K-C protocol. The set of all keys is $\mathsf{K} = \{K'_{AB}, K_{AB}, K_{AS}, K_{BS}\}$ and the set of safe keys is $\mathsf{K}_S = \{K_{AS}, K_{BS}\}$. Because the keys used in cryptographic operations by both protocols belong to K_S and no secrets from Y-L are transmitted unencrypted by K-C, Σ_{YL} is *key secrecy independent* of Σ_{KC}.

To determine if the *message independency* property is satisfied we must construct the typed representation of the two protocols. The knowledge of each role consists of all the role names, the permanent keys shared with role S and the nonces or keys generated by each role. After applying the $KSTr$ function, the resulting typed model can be seen in Figure 3, where we used interrupted lines to symbolize the existence of virtual layers. The typed models of Σ_{YL} and Σ_{KC} are denoted by Σ_{tYL} and Σ_{tKC} respectively.

By applying the CT predicate to all encrypted typed terms emitted by t-strands in Σ_{tKC} and all typed terms received by t-strands in Σ_{tYL}, we find two typed terms for which CT holds, namely: $\{r, r, u, k'\}_{k_1}$ emitted in Σ_{tKC} and $\{r, u, n, u\}_{k_1}$ received in Σ_{tYL}. Because of this, Σ_{tYL} is not message independent of Σ_{tKC}. By allowing the two protocols to run alongside each other, the intruder can construct a type-flaw attack based on these two typed terms. To correct the problem, we extend the typed term from the Y-L protocol with a role type, so that the term $\{r, u, n, u\}_{k_1}$ becomes $\{r, r, u, n, u\}_{k_1}$. Thus, the predicate CT will not hold for any encrypted typed term pair, resulting, by the independence theorem, that the two protocols (in the corrected form) are independent.

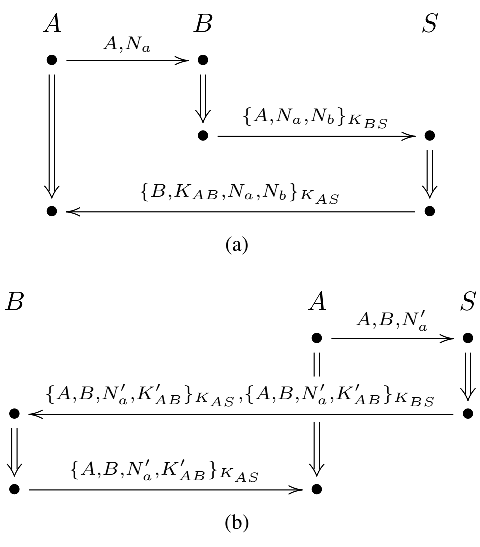
A
B
S
A, N_a
$\{A, N_a, N_b\}_{K_{BS}}$
$\{B, K_{AB}, N_a, N_b\}_{K_{AS}}$
(a)
B
A
S
A, B, N'_a
$\{A, B, N'_a, K'_{AB}\}_{K_{AS}}, \{A, B, N'_a, K'_{AB}\}_{K_{BS}}$
$\{A, B, N'_a, K'_{AB}\}_{K_{AS}}$
(b)

Figure 2: Representing two simplified protocols in the k-strand model: (a) Yahalom-Lowe (b) Kao-Chow.

6. Related work

In the literature we find many formal methods developed for the analysis of security protocols. However, only a few of these can be applied to the analysis of multiple concurrent protocols [13, 15, 16]. One of these approaches was proposed by Cremers and Maw in [13], based on which a tool has been developed for the automatic verification of security protocols [14]. The analysis process is based on a semantic enriched with local role knowledge. This specification is used by the automatic tool to construct the state space and determine if attacks are possible. The analysis of multiple security protocols is made intuitively, by including in the specification multiple protocols. In contrast, our method does not need a construction of the state space. However, because we developed a method for verifying the independence of security protocols, our approach can not be used for the analysis of a protocol running in isolation. The other approaches either need additional constructions to include multiple protocols in the representation [15] or they limit themselves to a representation from which results are rather difficult to interpret [16].

The method proposed by Guttman and Fabrega in [4] is similar to ours in the sense that it uses the strand-based representation accompanied by a theorem to verify the inde-

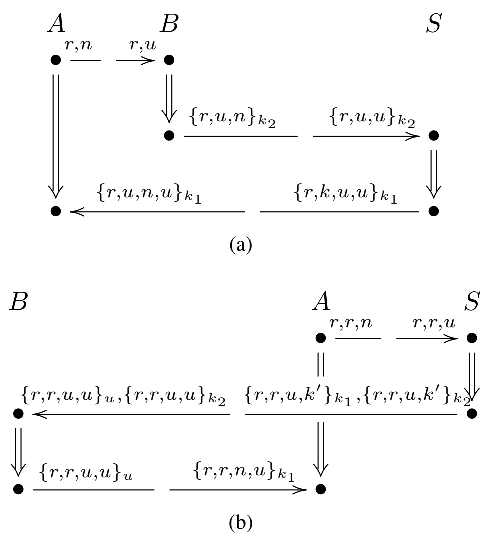

Figure 3: Protocol representation in the t-strand model: (a) Yahalom-Lowe (b) Kao-Chow.

pendence of security protocols. The approach requires the identification of message similarities in the modeled protocols that can be used by the penetrator to construct attacks. This is, however, rather difficult to achieve mainly because of the multiple protocol instantiations that can simultaneously exist. Our method simplifies the identification process by including in the representation only the types of components. This allows a syntactical analysis of the involved protocols that eliminates the state space explosion problem.

Lately, there has been a growing interest towards protocol design by composition [2, 9]. The involved subprotocols must not only be independent, but the resulting protocol must present an accumulative set of properties. These approaches, however, use predefined protocols which contain sufficient information so that the composition will not lead to the violation of security properties. Because of this, these methods can not be applied on existing protocols.

7. Conclusion

We have developed a procedure for verifying the independence of security protocols. The proposed model allows us to conduct a syntactical analysis on security protocol messages, thus eliminating the state space explosion problem due to multiple protocol instances. By including role knowledge in the model, we are able to represent different participant views on protocol messages. We also provided several mapping functions to allow a correct transformation from the regular strand-based specification to the proposed typed model.

For verifying security protocol independence, we constructed a theorem and proved that if two protocols satisfy a secrecy and a message independence condition, they are independent not only in the typed model, but also in the regular model.

Because of the abstraction level that includes only message component types, the approach from this paper may identify message similarities that do not necessarily lead to attacks. These situations should, however, be resolved because there may exist an instantiation of the involved messages that can be manipulated by the intruder to construct effective computational DoS (i.e. Denial of Service) attacks.

References

[1] F.J.T. Fabrega, J.C. Herzog, J.D. Guttman, “Strand Spaces: Proving security protocols correct”, Journal of Computer Science, Vol. 7, 1999, pp. 191–230.

[2] J.D. Guttman, “Security protocol design via authentication tests”, In Proc. of the 15th IEEE Computer Security Foundations Workshop, IEEE CS Press, 2002.

[3] G. Lowe, “Some new attacks upon security protocols”, In Proc. of the Computer Security Foundations Workshop VIII, 1996.

[4] J.D. Guttman, F.J.T. Fabrega, “Protocol Independence through Disjoint Encryption”, In the Proc. of the 13th CSFW, 2000, pp. 24–34.

[5] C. Meadows, “A Procedure for Verifying Security Against Type Confusion Attacks”, In the Proc. of the 16th CSFW, 2003, p. 62.

[6] D. Dolev, A. Yao, “On the security of public-key protocols”, IEEE Transactions on Information Theory, Vol. 29, 1983, pp. 198–208.

[7] A.D. Gordon, A. Jeffrey, “Authenticity by Typing for Security Protocols”, Journal of Computer Security, Vol. 4, 2003, pp. 451–520.

[8] C. Weidenbach, “Towards an automatic analysis of security protocols”, In the Proc. of the 16th International Conference on Automated Deduction, 1999, pp. 378-382.

[9] Hyun-Jin Choi, “Security protocol design by composition”, Cambridge University, UK, Technical report Nr. 657, UCAM-CL-TR-657, ISSN 1476-2986, 2006.

[10] C.J.F. Cremers, “Feasability of Multi-Protocol Attacks”, In the Proc. of the first ARAS conference, 2006.

[11] Security Protocol Open Repository (SPORE), Available at: http://www.lsv.ens-cachan.fr/spore.

[12] B. Schneier, “Applied Cryptography”, John Wiley & Sons, 1996.

[13] C.J.F. Cremers, S. Mauw, “Operational semantics of security protocol”, LNCS, Vol. 3466, Springer, 2005.

[14] C.J.F. Cremers, Scyther documentation, 2004.

[15] M. Abadi, A. Gordon, “A calculus for cryptographic protocols: The spi calculus”, Inf. Comput., 1999, pp. 1–70.

[16] G. Lowe, “Casper: A compiler for the analysis of security protocols”, In the Proc. of the 10th Computer Security Foundations Workshop, IEEE, 1997, pp. 18–30.