International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:8, 2008

Specification of Agent Explicit Knowledge in Cryptographic Protocols

Khair Eddin Sabri, Ridha Khedri, and Jason Jaskolka
Department of Computing and Software
Faculty of Engineering
McMaster University
{sabrike, khedri, jaskolj}@mcmaster.ca

***Abstract*— Cryptographic protocols are widely used in various applications to provide secure communications. They are usually represented as communicating agents that send and receive messages. These agents use their knowledge to exchange information and communicate with other agents involved in the protocol. An agent knowledge can be partitioned into explicit knowledge and procedural knowledge. The explicit knowledge refers to the set of information which is either proper to the agent or directly obtained from other agents through communication. The procedural knowledge relates to the set of mechanisms used to get new information from what is already available to the agent.

In this paper, we propose a mathematical framework which specifies the explicit knowledge of an agent involved in a cryptographic protocol. Modelling this knowledge is crucial for the specification, analysis, and implementation of cryptographic protocols. We also, report on a prototype tool that allows the representation and the manipulation of the explicit knowledge.

Keywords: Information Algebra, Agent Knowledge, Cryptographic Protocols

I. Introduction and Motivation

Cryptographic protocols are widely used in various applications to provide secure communications. They are usually represented as communicating agents that send and receive messages. Different formal methods are used to analyze them such as logical methods [3], [6], [22], process algebras [2], [15], [16], [18], and Petri-Nets [4]. Analyzing cryptographic protocols is usually based on the assumption that an intruder can control the network by intercepting, creating, or modifying messages. The intruder knowledge of the exchanged information and its nature are essential for attacking protocols. The intruder can enrich its knowledge by receiving information as messages and producing new information from the existing one. The way the knowledge is modelled and evolved is critical for a deeper analysis of cryptographic protocols. Mathematical models are essential for capturing agents knowledge and behaviours especially those of an intruder.

An agent knowledge can be seen as composed of *explicit knowledge* and *procedural knowledge*. The *explicit knowledge* is related to the information that an agent possesses such as its own key(s), the cipher(s) used for encryption and decryption, and the identity of other agents. Information in the explicit knowledge can be sent to other agents. Also, the information an agent receives from other agents becomes a part of its explicit knowledge.

The *procedural knowledge* involves a set of mechanisms/functions that enables an agent to obtain new information from its explicit knowledge. For example, if the explicit knowledge of an agent contains an encrypted message as well as the key and the cipher used to decrypt the message, then by using the procedural knowledge, the secret can be obtained. In this paper, we focus only on the explicit knowledge.

Both parts of the agent knowledge are necessary in analyzing cryptographic protocols. When we analyze the literature using this classification of agent knowledge, we find several representations of that knowledge. For example, in the Knowledge Based Logic Inference System [17], four types are used to classify the information in the agent knowledge: agent, nonce, key, and message and each piece of information in the explicit knowledge should belong to a specific type. Explicit knowledge is represented as the axiom $\mathit{Know}(X, M)$ which indicates that the agent X knows the message M. The procedural knowledge is defined as inference rules such as "$\mathit{Know}(X, M)$ and $\mathit{Know}(X, \mathit{Key}\ K)$ then $\mathit{Know}(X, \mathit{Encrypt}\ M\ K)$". In Strand Space [11], an agent explicit knowledge is represented as a set containing the initial information. The procedural knowledge is represented as a set of strands, where each one can be seen as a sequence of possible actions that an agent can perform on the initial information to obtain a new information. In Brutus tool [8], the explicit knowledge is represented as a set of messages. The atomic messages, which are not obtained using the procedural knowledge, are classified into keys, agent names, nonce numbers and data. The procedural knowledge is defined in Brutus as inference rules such as if $m \in M$ and key $k \in A$ then $\{m\}_k \in M$ where M is the set of all messages and A is the set of atomic messages.

In the literature, attention has been given to the procedural knowledge representation. For example, in [23], [25], a framework is developed to specify the algebraic properties of different elements of messages such as secrets, ciphers, and keys. In CSP [16] and NRL analyzer [19], the algebraic properties of cryptographic primitives are taken into consideration in producing new information. In [10], [14], the complexity of extending the intruder capability of deducing information based on algebraic properties is studied. On the other hand, minimum attention is given to the explicit knowledge, even

though an efficient protocol analysis depends on a precise and comprehensive knowledge of the information already exchanged.

The explicit knowledge representation is important for the specification, analysis, and implementation of cryptographic protocols. We summarize below the uses of the explicit knowledge in specifying protocols, specifying properties, generating code, reducing the state space, and generating specific type of attacks. For more information, we refer the readers to [26].

1) Protocols are based on exchanging messages between agents. These messages are constructed from agent explicit knowledge, and therefore it is necessary to specify protocols. Also, explicit knowledge is required to specify agent internal actions such as verifying the freshness of the received key. The explicit knowledge representation becomes more useful in complex protocols. For example, in the registration part of the *Equicrypt protocol* [15], a third party can handle several registrations at a time. Therefore, it should maintain an internal "table" with information on the users that have a registration in progress.
2) Some security properties are based on the explicit knowledge of agents. For example, the secrecy property requires that an intruder should not know a secret information existing in the explicit knowledge of other agents. Also, the classification of information in the agent knowledge allows specifying other properties. For example, long-term keys should not be sent through the network.
3) Even if the protocol is proved to be secure, it could be attacked due to its incorrect implementation. To reduce the risk of incorrect implementation, one can derive the code automatically from the mathematical model of the protocol and prove that the derivation is correct. Having an explicit knowledge representation that allows specifying internal actions such as inserting and extracting information from the knowledge as well as verifying the existence of an information in the knowledge would be necessary to generate the code.
4) State space explosion is one of the main issues in analyzing cryptographic protocols. This problem is caused by the undeterministic behaviour of the intruder. In [20], the authors indicate that an optimized behaviour of an intruder that sends the messages with the potential to be accepted by the receiver helps in reducing the state space. This approach is followed in analyzing an improved Needham-Schroeder public-key protocol using μCRL [21]. The explicit knowledge representation is necessary to build an intruder that sends useful messages only. For example, sending a message encrypted by a key different from that of the receiver is useless if the receiver should decrypt the message. Therefore, the intruder explicit knowledge should associate keys with the identity of the agents that use those keys.
5) The explicit knowledge can be used to generate specific attacks. For example, by specifying the sender of each message in the explicit knowledge, the intruder can replay messages to the sender to generate reflection attack.

In the literature, there are different representations of the explicit knowledge that share some characteristics such as classifying the information. Almost all the methods used in analyzing cryptographic protocols distinguish the key from other information. This classification ensures that only keys are used in the encryption or decryption. Also, this classification decreases the state space when looking for a key. In several cryptographic protocols, information is usually classified as keys, nonce numbers, agent identities, and other data. In the public key encryption, some methods classify keys into public and private. Also, in the literature, different kinds of information in the explicit knowledge are related to each other such as the public and private keys. Other methods [5], [17], [21] associate agent identity with key, card, nonce, and message.

One limitation of the existing methods is that it depends on the protocol to be specified. Therefore, to specify a protocol, different structures (e.g., functions, relations, predicates) need to be defined. Also, several functions are defined to extract a piece of information such as getting a private key corresponding to a specific public key. In order to specify another protocol with different classification of information, different structures and functions need to be introduced. Also, as the protocol becomes more complex and new types are required, additional functions need to be introduced as in the analysis of the merchant registration phase of the SET protocol [17] where six different functions are defined. To reduce the complexity of specifying agent explicit knowledge, it would be essential to have a uniform compact theory with a small number of operators that can be used to specify agent explicit knowledge regardless of the protocol to be specified.

In this paper, we develop a mathematical structure to represent the explicit knowledge and prove that it is an *information algebra* [13]. Proving that our structure is information algebra enables our representation to have similar characteristics to the methods used in specifying the explicit knowledge. Also, it allows our representation to overcome the limitation of the methods reported in the literature by applying our theory to any protocol without introducing a new theory. Finally, proving that our structure is information algebra allows using the established results within the information algebra[1]. Then, we use our structure in specifying protocols and properties, reducing the state space, and generating attacks.

It should be noted that it is necessary to represent agent procedural knowledge in order to analyze cryptographic protocols.

In Section II, we introduce our representation of the explicit knowledge. In Section III, we relate the mathematical structure to cryptographic protocols and illustrate the use of the explicit

[1]An Information algebra is a composite structure formed of a lattice and another composite algebraic structure which includes a commutative semigroup. The literature contains a large number of results in these structures.

knowledge through an example. In Section IV, we present a summary of related work and a discussion. Finally, we conclude and highlight current and future research.

II. Agent Explicit Knowledge

A. Information Algebra

In [13], Kholas and Stark explore connections between different representations of information. They introduce a mathematical structure called *information algebra*. This mathematical structure involves a set of information Φ and a lattice D. They show that relational databases, modules, and constraint systems are information algebras. In the rest of this paper, we denote elements of Φ by small letters of the Greek alphabet such as φ, ψ and χ. Each piece of information is associated with a *frame* (also called domain in [13]), and the lattice D is the set of all frames. Each frame x contains a *unit element* e_x which represents the empty information. Information can be combined or restricted to a specific frame. Combining two pieces of information φ and ψ is represented by $\varphi\psi$. Kholas and Stark [13] assume that the order of combining information does not matter and, therefore, the combining operator is both commutative and associative. Restricting an information φ to a frame x is denoted by $\varphi^{\downarrow x}$ which represents only the part of φ associated with x.

In the following definition and beyond, let $(D, \curlyvee, \curlywedge)$ be a lattice and x and y be elements of D called frames. Let $\preccurlyeq$ be a binary relation between frames such that $x \curlyvee y = y \leftrightarrow x \preccurlyeq y$. Let Φ be a set of information and φ,ψ, χ be elements of Φ. We denote the frame of information $\varphi \in \Phi$ by $d(\varphi)$. Let e_x be the empty information over the frame $x \in D$, the operation $\downarrow$ be a partial mapping $\Phi \times D \rightarrow \Phi$, and $\cdot$ be a binary operator on information. For simplicity, to denote $\varphi \cdot \psi$, we write $\varphi\psi$.

Definition 2.1 (Information Algebra [13]): An information algebra is a system (Φ, D) that satisfies the following axioms:

1) $(\varphi\psi)\chi = \varphi(\psi\chi)$
2) $\varphi\psi = \psi\varphi$
3) $d(\varphi\psi) = d(\varphi) \curlyvee d(\psi)$
4) $x \preccurlyeq y \rightarrow (e_y)^{\downarrow x} = e_x$
5) $d(\varphi) = x \rightarrow \varphi e_x = \varphi$
6) $d(e_x) = x$
7) $x \preccurlyeq d(\varphi) \rightarrow d(\varphi^{\downarrow x}) = x$
8) $x \preccurlyeq y \preccurlyeq d(\varphi) \rightarrow (\varphi^{\downarrow y})^{\downarrow x} = \varphi^{\downarrow x}$
9) $d(\varphi) = x \ \wedge \ d(\psi) = y \rightarrow (\varphi\psi)^{\downarrow x} = \varphi(\psi^{\downarrow x \wedge y})$
10) $x \preccurlyeq d(\varphi) \rightarrow \varphi\varphi^{\downarrow x} = \varphi$

□

The first two axioms indicate that the set of pieces of information together with the combining operator constructs a semi-group. The Axiom 3 states that the frame of combining two pieces of information is the join of their frames. The axioms (4-6) give properties of the empty information e_x. Axioms (7-8) give the properties of focusing an information to a specific frame. The axioms (9-10) give properties that involve combining and focusing of information.

B. Explicit Knowledge Representation

In this section, we present a mathematical structure to specify an agent explicit knowledge which involves a set of pieces of information Φ. Each piece is associated with a frame that belongs to the lattice D. The notion of frame in this paper in different from that of Abadi and Cortier [1]. In [1], frame represents the information available to the intruder. While in this paper, frames are considered as classification of the information in the explicit knowledge. We designate to each agent an information algebra to model its explicit knowledge. We define two operators on frames and prove that the set of frames with the defined operators is a lattice. We also define an operator to represent combining information. Furthermore, we define an operator to focus on a specific part of information such as focusing on keys. We prove that our structure is an information algebra. It should be noted that we do not provide more contributions to the theory of information algebra. Our contribution in this section is developing a mathematical structure to specify the explicit knowledge and prove that the structure is an information algebra which enables us to inherit a vast body of mathematical knowledge related to information algebra.

Definition 2.2 (Agent Information Frame): Let $\{\mathbb{A}_i \mid i \in I\}$ be a family of sets indexed by the set of indices I. An information frame D_I is defined as:

$$D_I \triangleq \prod_{i \in I} \mathcal{P}(\mathbb{A}_i)$$

Which can be equivalently written as

$$D_I \triangleq$$

$$\{f : I \rightarrow \bigcup_{i \in I} \mathcal{P}(\mathbb{A}_i) \mid \forall (i \mid i \in I : f(i) \in \mathcal{P}(\mathbb{A}_i))\}$$

□

Let $J \subseteq I$ and $\mathcal{I}_J \subseteq I \times I$ such that $\mathcal{I}_J = \{(x, x) \mid x \in J\}$ (i.e., $\mathcal{I}_J$ is the identity on J). Given the frame D_I, we can define D_J as $\{g \mid \exists (f \mid f \in D_I : g = \mathcal{I}_J ; f)\}$ where ; denotes relational composition. We call an element φ of D_J an *information* and D_J the *frame* of φ and denote[2] it by $d(\varphi)$ where "d" is called the labelling operator. The information φ is a function which can be written as a set of 2-tuples (i, A) where i is an index and A is a set. Each frame D_J contains a special element called the *empty information* e_{D_J} and is defined as $\{(i, \emptyset) \mid i \in J\}$. Whenever, it is clear from the context, we write e_J instead of e_{D_J}. For $J \subseteq I$, we denote the *set of all frames* D_J by D and the *set of all pieces of information* by Φ.

As an example of our representation of Φ and D, suppose that an agent can handle only keys and ciphers. In this case, $I = \{k, c\}$ and the lattice D is constructed as in Figure 1. The lattice D consists of four frames: $D_\emptyset$ frame that might involve only the empty information $e_\emptyset$ (absence of information), $D_{\{k\}}$ (resp. $D_{\{c\}}$) frame relates to keys (resp. ciphers), and $D_{\{k,c\}}$ frame relates to information that includes both keys and ciphers. To illustrate our representation of Φ, let us assume

[2]The notation $d(\varphi)$ to denote the frame of φ comes from the usage of the term domain in [13] as a synonym for frame. We prefer to use the term frame to avoid any confusion with the domain of a relation.

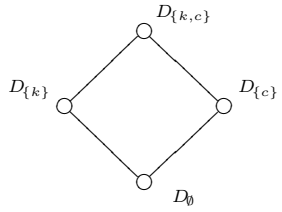

Fig. 1. The lattice D of the frames generated from $I = \{k, c\}$

that an agent knows the keys k_1 and k_2 and the cipher c_1 and c_2. Also, it knows that k_1 is a valid key for c_1 and k_2 is valid for c_2. We represent the information in our structure as: $\Phi = \{\{(k, \{k_1, k_2\})\}, \{(c, \{c_1, c_2\})\}, \{(k, \{k_1\}), (c, \{c_1\})\}, \{(k, \{k_2\}), (c, \{c_2\})\}\}$, where the pieces of information $\{(k, \{k_1, k_2\})\} \in D_{\{k\}}$, $\{(c, \{c_1, c_2\})\} \in D_{\{c\}}$ and both $\{(k, \{k_1\}), (c, \{c_1\})\}$ and $\{(k, \{k_2\}), (c, \{c_2\})\}$ belong to $D_{\{k,c\}}$. Such representation allows ignoring invalid cipher-key combinations such as key k_1 with cipher c_2 and k_2 with cipher c_1.

From the above representations, we have the following axioms that will be used in the proofs given later in this paper.

Axioms 2.1:

1) $\varphi \in D_J \rightarrow d(\varphi) = D_J$
2) $e_J \triangleq \{(i, \emptyset) \mid i \in J\}$

□

From the definition of D_J, it follows that $\varphi \in D_J \rightarrow \forall(i \mid i \in J : \varphi(i) \in \mathcal{P}(\mathbb{A}_i))$. Therefore, $\varphi \in D_J$ can be written as a set of 2-tuples $\{(i, A) \mid i \in J \wedge A \subseteq \mathbb{A}_i\}$. We provide the following propositions that will be needed in subsequent proofs.

Proposition 2.1: For $J, K \subseteq I$ and $\varphi \in \Phi$

1) $\varphi \in D_K \rightarrow \mathcal{I}_J{:}\varphi = \{(i, A) \mid i \in (J \cap K) \wedge A \subseteq \mathbb{A}_i\}$
2) $\mathcal{I}_J{:}\mathcal{I}_K = \mathcal{I}_{J \cap K}$
3) $\varphi \in D_K \rightarrow d(\mathcal{I}_J{:}\varphi) = D_{J \cap K}$

Proof: Detailed proof can be found in [24]

1) The proof uses the definitions of relation composition, φ, and $\mathcal{I}_J$ as well as the trading rule for $\exists$, set intersection axiom, and the distributivity of $\wedge$ over $\exists$.
2) We use the definitions of relation composition, $\mathcal{I}_J$, $\mathcal{I}_K$, and $I_{J \cap K}$ as well as apply the distributivity of $\wedge$ over $\exists$, and set intersection axiom.
3) The proof uses the definition of D_K, definition of $D_{J \cap K}$, Proposition 2.1(2), and Axiom 2.1(1).

■

Let us assume that an intruder obtained the information $\varphi = \{(k, \{k_a, k_b\}), (c, \{c_1\})\}$ from a protocol session. It also obtained the information $\psi = \{(k, \{k_a\}), (c, \{c_2\})\}$ from another protocol session. One needs to express the information that the intruder holds from both sessions as $\varphi \cdot \psi = \{(k, \{k_a, k_b\}), (c, \{c_1, c_2\})\}$. The operator $\cdot$ is defined as follows:

Definition 2.3 (Combining Information): Let Φ be a set of information and φ, ψ be its elements. Let $d(\varphi) = D_J$ and $d(\psi) = D_K$. We define the binary operators $\cdot$ (however, we write $\varphi\psi$ to denote $\varphi \cdot \psi$) on information as:

$$\varphi\psi \triangleq \{(i, A) \mid i \in J \cap K \wedge A = \varphi(i) \cup \psi(i)\} \cup \{(i, A) \mid i \in J - K \wedge A = \varphi(i)\} \cup \{(i, A) \mid i = K - J \wedge A = \psi(i)\}$$

□

We introduce the $*$ operator on pieces of information as $\varphi * \psi \triangleq \{(i, A) \mid i \in J \cap K \wedge A = \varphi(i) \cap \psi(i)\}$. This operator will be used later to define operators on frames. The $*$ operator represents the sharing of two pieces of information. Based on the above operators, we define two operators on frames as:

Definition 2.4: Let D_J and D_K be frames, we define the operators $\curlyvee$ and $\curlywedge$ as:

1) $D_J \curlyvee D_K \triangleq \{\chi \mid \exists(\varphi, \psi \mid \varphi \in D_J \wedge \psi \in D_K : \chi = \varphi\psi)\}$
2) $D_J \curlywedge D_K \triangleq \{\chi \mid \exists(\varphi, \psi \mid \varphi \in D_J \wedge \psi \in D_K : \chi = \varphi * \psi)\}$

□

Note: The detailed proofs of the following propositions can be found in [24]. We provide here the properties used in the proofs.

Proposition 2.2:

1) $D_J \curlyvee D_K = D_{J \cup K}$
2) $D_J \curlywedge D_K = D_{J \cap K}$

Proof:

1) The proof applies the definitions of D_J, D_K, and $D_{J \cup K}$ as well as it applies Definition 2.4(1), distributivity of $\wedge$ over $\exists$, trading rule for $\exists$, nesting axiom, interchange of dummies, Definition 2.3, Proposition 2.1(1), renaming, and range split axiom.
2) We use the definitions of D_J, D_K, and $D_{J \cap K}$ and we apply Definition 2.4(2), distributivity of $\wedge$ over $\exists$, trading rule for $\exists$, nesting axiom, interchange of dummies, definition of $*$, Proposition 2.1(1), renaming, and range split axiom.

■

The operators $\curlyvee$ and $\curlywedge$ satisfy the following properties:

Proposition 2.3:

1) $D_J \curlyvee D_K = D_K \curlyvee D_J$
2) $D_J \curlywedge D_K = D_K \curlywedge D_J$
3) $(D_J \curlyvee D_K) \curlyvee D_L = D_J \curlyvee (D_K \curlyvee D_L)$
4) $(D_J \curlywedge D_K) \curlywedge D_L = D_J \curlywedge (D_K \curlywedge D_L)$
5) $D_J \curlyvee (D_J \curlywedge D_L) = D_J$
6) $D_J \curlywedge (D_J \curlyvee D_L) = D_J$

Proof: We use Proposition 2.2(1), Proposition 2.2(2) and the properties of $\cap$ and $\cup$.

■

Proposition 2.3 states that $(\{D_J\}_{J \subseteq I}, \curlyvee, \curlywedge)$ is a lattice.

For simplicity, we use D to denote the lattice $(\{D_J\}_{J \subseteq I}, \curlyvee, \curlywedge)$. On the lattice D and for D_J and D_K

frames in D, it is known [9, page 39] that we have $D_J \preccurlyeq D_K \leftrightarrow (D_J \curlyvee D_K = D_K) \leftrightarrow (D_J \curlywedge D_K = D_J)$.

As in [13], we define the *more informative* relation on pieces of information as:

Definition 2.5 (More Informative Relation): Let Φ be a set of information and φ, ψ be elements of Φ. Let D be a lattice and D_J and D_K be elements of D. Let $d(\varphi) = D_J$ and $d(\psi) = D_K$. We define the binary relation $\leq$ on information as: $\varphi \leq \psi \leftrightarrow J \subseteq K \ \wedge \ \forall(i \mid i \in J : \varphi(i) \subseteq \psi(i))$

□

Proposition 2.4: The relation $\leq$ is a partial order.

Proof: The proof uses the fact that $\subseteq$ is a partial order.

■

Proposition 2.5:

1) $\forall(J, K \mid J, K \subseteq I : J = K \rightarrow D_J = D_K)$
2) $\forall(J, K \mid J, K \subseteq I : D_J \preccurlyeq D_K \rightarrow J \subseteq K)$

Proof:

1) The proof uses the trading rule for $\forall$ and substitution axiom.
2) The proof uses Proposition 2.5(1) and Proposition 2.2(1).

■

Assume that an agent has the information $\varphi = \{(k, \{k_a, k_b\}), (c, \{c_1, c_2\})\}$ and it is interested only in the set of keys $\{(k, \{k_a, k_b\})\}$. To focus on a part of a piece of information, we define the following operator.

Definition 2.6 (Marginalizing Information): Let D_J be a frame and φ be an information, we define a binary operator $\downarrow: \Phi \times D \rightarrow \Phi$ as $\varphi^{\downarrow D_J} \triangleq \mathcal{I}_J{:}\varphi$.

□

Proposition 2.6: For $J, K \subseteq I$, we have

1) $(\varphi\psi)\chi = \varphi(\psi\chi)$
2) $\varphi\psi = \psi\varphi$
3) $d(\varphi\psi) = d(\varphi) \curlyvee d(\psi)$
4) $d(\varphi) = D_J \rightarrow \varphi e_J = \varphi$
5) $d(e_J) = D_J$
6) $D_J \preccurlyeq D_K \rightarrow (e_K)^{\downarrow D_J} = e_J$
7) $d(\varphi) = D_J \wedge d(\psi) = D_K \rightarrow (\varphi\psi)^{\downarrow D_J} = \varphi(\psi^{\downarrow D_J \wedge D_K})$
8) $D_J \preccurlyeq d(\varphi) \rightarrow d(\varphi^{\downarrow D_J}) = D_J$
9) $D_J \preccurlyeq D_K \preccurlyeq d(\varphi) \rightarrow (\varphi^{\downarrow D_K})^{\downarrow D_J} = \varphi^{\downarrow D_J}$
10) $D_J \preccurlyeq d(\varphi) \rightarrow \varphi\varphi^{\downarrow D_J} = \varphi$

Proof: The full detailed proof can be found in [24].

1) The proof calls for Definition 2.3, commutativity and associativity of $\cup$, and properties of set difference.
2) We use Definition 2.3 and commutativity of $\cap$ and $\cup$.
3) The proof essentially uses Axiom 2.1(1), Propositions 2.2(1), the definition of $D_{J \cup K}$, Proposition 2.1(1), and Definition 2.3.
4) We basically use Definition 2.3, Axiom 2.1(2), idempotency of $\cap$, and empty range axiom.
5) The proof essentially calls for Axiom 2.1(1 and 2), the definition of D_J, and Proposition 2.1(1).
6) The proof uses Definition 2.6, Axiom 2.1(2), Proposition 2.1(1), and Proposition 2.5(2).
7) The proof uses Definition 2.6, Definition 2.3, Proposition 2.1(1), and properties of set difference, $\cup$ and $\cap$.
8) The proof uses Definition 2.6, Proposition 2.1(3), Axiom 2.1(1), and Proposition 2.5(2).
9) We use Definition 2.6, Proposition 2.1(2), and Proposition 2.5(2).
10) The proof calls for Definition 2.6, Proposition 2.1(1), Definition 2.3, Axiom 2.1(1), Proposition 2.5(2), range split axiom, and properties of set difference, $\cup$, and $\cap$.

■

Proposition 2.6 states that the structure (Φ, D) is an information algebra.

As consequence results of proving that (Φ, D) is an information algebra, the following properties hold and the proofs can be found in [13]:

Proposition 2.7:

1) $d(\varphi) = D_J \rightarrow \varphi^{\downarrow e_J} = \varphi$
2) $\varphi\varphi = \varphi$
3) $D_J \preccurlyeq D_K \rightarrow e_J e_K = e_K$
4) $d(\varphi) = D_J \rightarrow (\varphi e_K)^{\downarrow D_J} = \varphi$
5) $D_J \preccurlyeq d(\varphi) \rightarrow (\varphi e_K)^{\downarrow D_J} = \varphi^{\downarrow D_J}$
6) $e_{(J \curlyvee K)} = e_J e_K$

□

III. Relating the Mathematical Structure to Protocols

We represent the explicit knowledge of each agent involved in a protocol as a set of information Φ and a lattice of frames D. Our structure has several interesting features. We can classify information into different frames. The combining, marginalizing, and labelling operators defined in our context on information and frames are essential for handling the explicit knowledge. For example, combining information is used to expand, or to relate different kinds of information. The labelling operator is used to associate each piece of information with a frame so that one can distinguish between different kinds of information such as keys, ciphers, etc. The marginalizing operator can be used to focus on a part of a piece of information. Also, as Φ is a set, different operators on sets can be used to add or remove a piece of information from the knowledge. Each agent can have its own structure by defining its own Φ and D. Finally, our structure and its theory can be applied to any protocol without introducing new theory. This can be considered the main advantage of our representation over the existing ones. For example, we do not need to define explicitly new types or relations to relate two pieces of information or extract part of it as other methods do.

To handle the explicit knowledge, we define several functions. We define functions to insert and remove a piece of information from the knowledge. The signature of these functions is $\mathbb{K} \times \Phi \rightarrow \mathbb{K}$ where $\mathbb{K}$ is a set of knowledge which is an information algebra. We also define a function to update an information with another information in the

knowledge of an agent. The signature of the update function is $\mathbb{K} \times \Phi \times \Phi \rightarrow \mathbb{K}$. Furthermore, we define functions to extract pieces of information from the knowledge and to verify if an information exists in the knowledge of an agent. The *extract* function, which has the signature $\mathbb{K} \times D \times \Phi \rightarrow \mathcal{P}(\Phi)$, is based on the marginalizing, and labelling operators. The function $extract(\mathcal{N}, x, \varphi)$ extracts pieces of information from the knowledge $\mathcal{N} \triangleq (D, \Phi)$ that contain φ, and then restrict them to the frame x. Formally $extract(\mathcal{N}, x, \varphi) \triangleq \{\psi^{\downarrow x} \mid x \in D \wedge \psi \in \Phi \wedge \varphi \leq \psi\}$.

For example, the function $extract(\mathcal{N}, D_{\{key\}}, \{(sender, \{A\})\})$ extracts the keys associated with the *sender* A.

The function $isInKnowledge(\mathcal{N}, x, \varphi)$, which has the signature $\mathbb{K} \times D \times \Phi \rightarrow \mathbb{B}$, verifies if there is an information associated with the frame x and combined with φ exists in the knowledge $\mathcal{N}$. A special case of this function is verifying if the information φ exists in the knowledge $\mathcal{N}$. In this case x should be the frame of φ.

A. Illustrative Example

To assess the adequacy of our definitions, we implement a prototype tool, written in the functional programming language *Haskell*, for the proposed model of information algebra. This prototype tool is used to represent and manipulate explicit knowledge of agents. It allows initializing the lattice of frames D and the set of information Φ for each agent. It implements the functions presented earlier so that the user can insert, remove and update the knowledge of each agent. Also, it allows extracting information from the knowledge and verifying the existence of an information in the knowledge of an agent. The prototype tool is a part of a whole system that is used to analyze cryptographic protocols. The other parts deal with the procedural knowledge and the communication between agents.

In this section, we illustrate the use of our representation of the explicit knowledge to specify protocols, specify properties, reduce the state space, and generate a specific type of attacks using the prototype tool.

As an example, we specify the explicit knowledge of a server and an intruder that we denote by S, and Z, respectively. Specifying the knowledge of each agent requires providing the tool with a set of indices such as I_s or I_z, and the initial set of information. Assume that the server set of frames is indexed by $I_s = \{sessionK, publicK, privateK\}$ that represents session, public, and private keys. Also, assume that the intruder set of frames is indexed by $I_z = \{sender, sessionK, publicK, id, message\}$. The elements of I_z respectively represent the identities of the sender of a message, session keys, public keys, agent identities, and messages. Initializing the indices of agents is implemented in prototype tool in the function *setFrameIndices*. This function takes as parameter an agent name and the set of indices. Then, it builds a lattice and stores it in a file associated with the agent name. The function *makeSet* is used to build the set.

```
-- Initialize the set of indices for the server
-- and the intruder

setFrameIndices "S"
   (makeSet ["sessionK","publicK","privateK"])

setFrameIndices "Z"
   (makeSet ["sender", "sessionK", "publicK",
    "id", "message"])
```

The function *setInitialKnowledge* can be used to set the initial knowledge of each agent. This function takes as input an agent name and a set of pieces of information. Then, it stores them in a file associated with the agent name. For efficiency purposes, we associate each piece of information with its frame when implementing the agent knowledge. This representation facilitates retrieving the frame of a piece of information or the frame produced by combing two pieces of information or focusing of information. Suppose that the initial knowledge of the server contains combined information that relates the public key to its corresponding private key $\{(privateK, \{PrKs\}), (puplicK, \{PKs\})\}$. Also, assume that the initial knowledge of the intruder contains two pieces of information $\{(id, \{A\}), (puplicK, \{PKa\})\}$ and $\{(id, \{S\}), (puplicK, \{PKs\})\}$ that relates agents to their public keys. These two assumptions about the initial knowledge of agents can be represented by using the prototype tool as follows:

```
-- Set the initial knowledge for the server
-- and the intruder

setInitialKnowledge "S"
   [ ([("privateK", ["PrKs"]), ("publicK",
       ["PKs"])],["publicK", "privateK"])]

setInitialKnowledge "Z"
   [([("id",["A"]), ("publicK",["PKa"])],
       ["publicK", "id"]),
   ([("id",["S"]), ("publicK",["PKs"])],
       ["publicK", "id"])]
```

Assume that the server receive a session key $k1$ from other agents. Inserting this information to the server knowledge can be achieved through the *insertInformation* function.

```
-- Insert the following session key into the
-- server knowledge

insertInformation "S"
   ([("sessionK",["k1"])],["sessionK"])
```

Suppose that a protocol specification states that the server receives a *fresh* session key from agent A and forwards it to the agent B. The server internal action that verifies the freshness of the key k (i.e., the key k is not played in previous session) can be specified using our framework as: $\neg\, \exists(\varphi \mid \varphi \in \Phi^S : d(\varphi) = D_{\{sessionK\}} \wedge k \leq \varphi)$. This property can be checked using the tool as follows:

```
-- Verify if the server knowledge contains
-- the session key "k1"

isInKnowledge "S" (["sessionK"])
   ([("sessionK",["k1"])],["sessionK"])
output: True
```

```
-- Verify if the server knowledge contains
-- the session key "k4"

isInKnowledge "S" (["sessionK"])
   ([("sessionK",["k4"])],["sessionK"])
output: False
```

The property that the intruder should not get a session key k can be specified as $\neg\ \exists(\varphi\ \mid\ \varphi \in \Phi^Z\ :\ k \leq \varphi\,)$ equivalently $\forall(\varphi\ \mid\ \varphi \in \Phi^Z\ :\ \neg(k \leq \varphi)\,)$. This property can be checked using the tool through the following command

```
-- Verify if the intruder knowledge contains
-- the session key "k"

isInKnowledge "Z" (["sessionK"])
   ([("sessionK",["k"])],["sessionK"])
output: False
```

One way to reduce the state space is to restrict the behaviour of the intruder to the useful behaviours only. For example, all the messages sent to the server should be encrypted with the server public key if the server should decrypt the message. The server message can be extracted from the knowledge as: $\mathit{extract}(N^Z, D_{\{\mathit{publicK}\}}, \{(\mathit{id}, \{\mathrm{S}\})\})$. This operator can be performed using the tool as follows:

```
extractInformation "Z" (["publicK"])
   ([("id",["S"])],["id"])
output: ([("publicK",["PKs"])],["publicK"])
```

Finally, the intruder can use its explicit knowledge to mount, for example, a reflection attack where it plays a message back to the sender. First, the intruder should combine each received message $\{(\mathit{message}, \{\mathit{msg}\})\}$ with the identity of the sender $\{(\mathit{sender}, \{x\})\}$ where x is the sender of the message *msg*. The function *knowCombine* is used to combine two pieces of information. Inserting information into the intruder knowledge after combing it with the information $\{(\mathit{message}, \{\mathit{msg}\})\}$ is represented using the tool as:

```
insertInformation "Z"
  (knowCombine ([("sender",["A"])],["sender"])
     ([("message",["msg1"])],["message"]))
insertInformation "Z"
  (knowCombine ([("sender",["B"])],["sender"])
     ([("message",["msg2"])],["message"]))
insertInformation "Z"
  (knowCombine ([("sender",["A"])],["sender"])
     ([("message",["msg3"])],["message"]))
```

The set of all messages that can be replayed to an agent A can be extracted from the explicit knowledge as: $\mathit{extract}(\mathcal{N}^Z, D_{\{\mathit{message}\}}, \{(\mathit{sender}, \{A\})\})$. The function extracts the messages from all pieces of information that contain $\{(\mathit{sender}, \{A\})\}$. To do that we proceed as follows:

```
extractInformation "Z" (["message"])
   ([("sender",["A"])],["sender"])
output:
  [([("message",["msg1"])],["message"]),
   ([("message",["msg3"])],["message"])]
```

IV. Related Work and Discussion

In this section, we report on several formal methods used to specify cryptographic protocols and compare them to our approach with respect to the explicit knowledge representation. The surveyed method represents agent explicit knowledge in cryptographic protocols using either sets or predicates. These two structures are related and each structure can be mapped to the other one. LOTOS [15], Brutus tool [8], strand space [11], and inductive approach [22] follow the set-based approach, while the knowledge-based logical system [17] and MSR [7] adopt predicates to capture agent explicit knowledge.

In the literature, different structures are introduced to classify and associate different kinds of information together. For example, LOTOS [15], strand space [11], Brutus [8], and inductive approach [22] associate private key to its corresponding public key. In MSR [7], the shared key is associated with two agent identities that use the key while the public and private keys are associated with one agent identity. In inductive approach [5], several functions are defined to associate agent identity with card, agent identity with key, and card with key. Also, functions are defined to extract pieces of information as in coloured Petri net [4] where a function is defined to return the decryption key of a given key and another function to return the shared key between two agents. The representation of the information that an agent possesses and its relationship are tailored to each protocol.

In our approach, the specifier needs only to define the set of frames of the explicit knowledge which indicates the classification of information. We use a compact number of operators to specify the agent explicit knowledge of any protocol. For example, the knowledge-based logical approach [17] uses about six functions to specify the registration phase of the SET protocol. Four functions are used to map an agent to its public encryption key, private encryption key, public signature key, and private signature keys. Also, a function is introduced to associate two agents with a shared key, and another function to verify if a message is a part of another one. In the framework proposed in this paper, only the pre-defined operators within the framework are required to manipulate the information. There is no need to define new operators. Having a small number of operators would reduce the complexity of specifying cryptographic protocols and verifying them.

Also, the proposed framework enables specifying the internal actions of agents. For example, we can specify the ability of the server to check the freshness of a message while this is not possible in Brutus [8]. The inability of specifying the internal actions would affect the protocol analysis and implementation. Finally, within the proposed framework one can specify a "rational" intruder that sends a message only if it has the potential to be accepted by the receiver. Specifying a rational intruder would reduce the state space generated by analyzing protocols, and thus reduce the complexity of the analysis. The specification of a rational intruder does not require introducing new operators to our framework. While in developing a rational intruder in μCRL [21] for analyzing the Needham-Schroeder public key protocol, two new sets of operators are introduced to store the nonce of the initiator and the nonce of receiver in addition to the original set that stores the nonce numbers. We do not need to specify three frames for nonce numbers. We can have only one frame for nonce

numbers. We should have also a frame for agent identity where the linkage of the two frames exists within our framework.

V. Conclusion and Future Work

In this paper, we present a framework to specify agent explicit knowledge. Our main contribution is developing an algebraic structure to specify the explicit knowledge of cryptographic agents based on information algebra. In our context, we define the combining, marginalizing, and labelling operators. These operators are all what is needed to express operations on information independently of the protocols. The proposed operators involve both information and their frames. We also define a set of frames to be associated with information. Then, we prove that our structure is an information algebra which links our work to a rich heritage of mathematical theories. Our mathematical structure is expressive as it allows combining information for different purposes regardless of their frames, extracting a part of information, or associating information with a frame. Also, our structure allows capturing different kinds of information such as associating keys with ciphers. Developing an expressive structure should be useful in identifying the set of messages that has the potential to generate successful attacks which can be used to build an efficient intruder. At the implementation level, each agent has a knowledge and performs operations such as inserting and extracting information. These operators that are performed as internal actions of an agent are specified within our framework and implemented in Haskell. Therefore, once we have the specification of the internal actions of an agent specified within our framework, the automatic generation of the corresponding code is straightforward.

In the literature of cryptographic protocols, operators are usually defined on information that belongs to a specific type, while the proposed framework enables a uniform and general way to handle information. Also, defining a relation between frames and linking them to the operators applied on information is not addressed in the literature. Furthermore, different protocol-dependent structures should be defined to relate different kinds of information which are not needed in our representation.

Currently, we are using the explicit knowledge to analyze the information flowing between agent. Also, it is used to analyze cryptographic protocols by integrating it with procedural knowledge and global and end-point calculi [24], [25].

References

[1] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1):2–32, 2006.

[2] Martin Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[3] Kamel Adi, Mourad Debbabi, and Mohamed Mejri. A New Logic for Electronic Commerce Protocols. *Theoretical Computer Science*, 291(3):223–283, January 2003.

[4] Issam Al-Azzoni, Douglas G. Down, and Ridha Khedri. Modeling and Verification of Cryptographic Protocols Using Coloured Petri Nets and Design/CPN. *Nordic Journal of Computing*, 12(3):200–228, September 2005.

[5] Giampaolo Bella. Inductive verification of smart card protocols. *Journal of Computer Security*, 11(1):87–132, 2003.

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

[7] Iliano Cervesato. Typed multiset rewriting specifications of security protocols. In A. Seda, editor, *First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology — MFCSIT'00*, pages 1–43, Cork, Ireland, 19–21 July 2000. Elsevier ENTCS 40.

[8] Edmund M. Clarke, Somesh Jha, and Wilfredo Marrero. Verifying Security Protocols with Brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, October 2000.

[9] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge university press, second edition, 2002.

[10] Stéphanie Delaune. Easy Intruder Deduction Problems with Homomorphisms. *Information Processing Letters*, 97(6):213–218, March 2006.

[11] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security*, 7(2–3):191–230, January 1999.

[12] Riccardo Focardi and Roberto Gorrieri. The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, September 1997.

[13] Jürg Kohlas and Robert F. Stärk. Information Algebras and Consequence Operators. *Logica Universalis*, 1(1):139–165, January 2007.

[14] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for the equational theory of Abelian groups with distributive encryption. *Information and Computation*, 205(4):581–623, April 2007.

[15] Guy Leduc and François Germeau. Verification of security protocols using LOTOS-method and application. *Computer Communications*, 23(12):1089–1103, July 2000.

[16] Gavin Lowe and Bill Roscoe. Using CSP to Detect Errors in the TMN Protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, October 1997.

[17] Xiao-Qi Ma and Xiao-Chun Cheng. Formal Verification of Merchant Registration Phase of SET Protocol. *International Journal of Automation and Computing*, 2(2):155–162, December 2005.

[18] Fabio Martinelli. Analysis of Security Protocols as Open Systems. *Theoretical Computer Science*, 290(1):1057–1106, January 2003.

[19] Catherine Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In *IEEE Symposium on Security and Privacy*, pages 216–231. IEEE Computer Society, May 1999.

[20] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murϕ. In *IEEE Symposium on Security and Privacy*, May 1997.

[21] Jun Pang. Analysis of a security protocol in μCRL. Technical Report SEN-R0201, CWI, Amsterdam, 2002.

[22] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, September 1998.

[23] Khair Eddin Sabri and Ridha Khedri. A multi-view approach for the analysis of cryptographic protocols. In *Workshop on Practice and Theory of IT Security (PTITS 2006)*, pages 21–27, Montreal, QC, Canada, 2006.

[24] Khair Eddin Sabri and Ridha Khedri. A mathematical framework to capture agent explicit knowledge in cryptographic protocols. Technical Report CAS-07-04-RK, department of Computing and Software, Faculty of Engineering, McMaster University, 2007. http://www.cas.mcmaster.ca/cas/research/cas_reports06-07.php (accessed on September 15, 2008).

[25] Khair Eddin Sabri and Ridha Khedri. Multi-view framework for the analysis of cryptographic protocols. Technical Report CAS-07-06-RK, department of Computing and Software, Faculty of Engineering, McMaster University, 2007.

[26] Khair Eddin Sabri and Ridha Khedri. Agent explicit knowledge: Survey of the literature and elements of a suitable representation. In *2nd Workshop on Practice and Theory of IT Security (PTITS 2008)*, pages 4–9, Montreal, QC, Canada, 2008.