Access Control Mechanisms for Fraglets

Fabio Martinelli
IIT-CNR
Pisa, Italy
fabio.martinelli@iit.cnr.it

Marinella Petrocchi
IIT-CNR
Pisa, Italy
marinella.petrocchi@iit.cnr.it

ABSTRACT

The paper describes a revised version of Crypto-fraglets, and an enhancement of the framework with access control mechanisms. Fraglets is an interesting computing model that has connections with communication protocols, formal rewriting systems and biological systems. Indeed, fraglets basic mechanisms resemble bio-chemical reactions.

1. INTRODUCTION

In [14, 15, 16, 17], an execution model for communication protocols that resembles the chemical reactions in living organisms has been introduced and applied to protocol resilience and genetic programming experiments. The goal was to propose a framework for making automatic the whole process of protocol development, involving the various phases of design, implementation and deployment. Trying to make automatic as much as possible operations and functionality of a network, the $fraglets$ execution model has been thought for its strong connections to themes like resilience and evolution. The underlying idea is indeed that nature has by itself "mechanism both for continuous operations as well as evolution using the processing of macromolecules", [14].

The main concepts underlying fraglets are active networking [13] (a mechanism for handling mobile code), chemical pathways (the way through which a cell metabolism is described), and automatic protocol synthesis. In particular, this last field has produced first results within information security. As an example, in [9, 12], the authors present theory and tools for specifying a security protocol that satisfies a given set of properties.

Motivation for working on fraglets within computer security is twofold. On the one hand, the programming language defined for fraglets can be encoded by the MultiSet Rewriting formal rules [3], as shown in [10]. This serves as a first brick to give to fraglets a formal semantics, the starting point for developing verification tools.

On the other hand, the fraglets programming language has been extended with basic cryptographic primitives [10], with the intent to start modeling and verifying security properties of network protocols.

Here, the goal is to refine that model and to extend it with access control mechanisms and primitives dealing with trust management.

Such features are necessary to enable the use of these computation fragments also in a security-sensitive context, and, in particular, to *tune* the security levels of the computations according to, *e.g.*, the trust levels of the nodes. Similarly, this will allow a fine-grained management of the fraglets model by making it closer to a real execution framework. Such a model is exploited in the BIONETS project [1] and the security and trust extensions are necessary to make it a running framework.

The original contributions of the paper are the following:

- We define a threat model for a fraglet network, and we extend Crypto-fraglets to be compliant with this model.
- We introduce also access control mechanisms to control how fraglets can migrate between nodes and how trust relationships may be created among them.

The paper is organized as follows. Section 2 recalls the work by Tschudin [14], it gives some basic instructions for processing fraglets, and it explains how a fraglet is processed within the network. Section 3 defines the threat model and gives a refined version of Crypto-fraglets. Section 4 discusses some access control mechanisms that could enhance the security and robustness of such framework. Finally, Section 5 gives some final remarks.

2. FRAGLETS

The theme underlying fraglets is that nature has developed a mechanism for continuous operations as well as evolution by processing macromolecules. Fraglets have been introduced as the equivalent of such molecules, and they represent fragments of a distributed computation.

A fraglet is denoted as $[s_1 : s_2 : \ldots tail]$, where s_i is a symbol and $tail$ is a (possibly empty) sequence of symbols.

Nodes of a communication network may process fraglets as follows. Each node maintains a fraglet store to which incoming fraglets are added. Fraglets may be processed within a store, except for the operation that transfers a fraglet from a source store to a destination store.

Fraglets processing is through a simple prefix programming language. Each operation may involve a single fraglet, or two fraglets. Here, we recall the instructions that serve in this paper. The interested reader can find the whole set of instructions in the tutorial available online on the fraglets website [4].

Rule $Send$ is responsible for transferring a fraglet from a context (store) A to another context (store) B. The rule takes as input the fraglet $_A[send : B : tail]$, located at store A, and returns the fraglet $_B[tail]$, located at store B. The name of the second store is given by the second symbol in the original fraglet. The $Send$ operation is unreliable, *i.e.*, it is not certain the $tail$ reaches the destination store.

Rule $Match$ concatenates two fraglets with matching tags. With the following two fraglets, $[match : s : \ldots tail_1]$ and $[s : tail_2]$, the rule $match$ returns the fraglet: $[tail_1 : tail_2]$, and the matching tag is s.

3. CRYPTO-FRAGLETS REVISED

Here, we revise Crypto-fraglets, with respect to [10], and according to the threat model defined as follows:

- Stores: at deployment, stores are classified into two categories, trusted and untrusted. An untrusted store is a store where fraglets may be maliciously processed.
- Adversary can:
  - eavesdrop during transmission;
  - process fraglets within an untrusted store, by means of all usual fraglets instructions;
  - maliciously deviate fraglets into untrusted stores, governed by the adversary itself.
- The adversary could not:
  - guess private information, unless it eavesdrops this (*e.g.*, private keys, secret keys);
  - have capability of processing fraglets within trusted stores, *e.g.*, by not entering trusted stores.

Finally, due to the unreliability of the $Send$ operation, a fraglet can move to an untrusted store by chance.

Against eavesdropping, misrouting and unreliability, it is correct to cipher any sensitive data. By doing so, the adversary cannot access the data, either when fraglets are in transit, or when fraglets are deviating, intentionally or by chance, onto untrusted stores.

Thus, we slightly change the rules for encryption and decryption, with respect to [10].

We consider symmetric cryptography, using the same key both for encryption and decryption. Given the clear text $tail$, one can apply a key K by obtaining the cyphertext $tail_K$. The original message is retrieved by means of the inverse operation, *i.e.*, by applying key K to cyphertext $tail_K$, the result is the clear text $tail$.

One can apply to a fraglet an encryption fraglet, with tag enc, to produce a third fraglet that represents the cyphertext. The same holds for decryption.

The rules are as follows:

$Operation$	$Input$	$Output$
Enc	${}_A[enc : s : K]$ ${}_A[s : tail]$	${}_A[s : tail_K]$
Dec	${}_B[dec : t : tail_K]$ ${}_B[t : K]$	${}_B[t : tail]$

An example program is the following:

${}_A[enc : s : K]$
ANY FRAGLET ${}_A[s : tail]$ IS PROCESSED THROUGH Enc;
${}_A[match : s : send : B : dec : t]$
ANY FRAGLET ${}_A[s : tail_K]$ IS PROCESSED THROUGH $match$;
${}_B[dec : t : tail_K]$
Dec APPLIED WITH OPPORTUNE FRAGLET ${}_B[t : K]$

and its execution steps are:

${}_A[enc : s : K]$	${}_A[s : tail]$
${}_A[match : s : send : B : dec : t]$	${}_A[s : tail_K]$
${}_A[send : B : dec : t : tail_K]$	
${}_B[dec : t : tail_K]$	${}_B[t : K]$
${}_B[t : tail]$	

4. ACCESS CONTROL

One of our global assumption is that, at deployment, there exist stores that are trusted by construction. These stores contain fraglets representing right encryption and decryption keys K.

In order to avoid that, when executing a certain program, malicious fraglets can enter trusted stores, we propose an *access control* mechanism. This will allow to execute $send$ instructions only if *guarded*. Currently, ${}_A[send : B : tail]$ is executed with no guard, *i.e.*, apart from unreliability, $tail$ may enter B with no control by B itself.

We enrich each store with *antechambers*, defined as a sort of check points, through which each fraglet must pass before really entering the store. AC_B stands for antechamber of B and notation ${}_{AC_B}[.. : .. : tail]$ means that the corresponding fraglet is at the antechamber of store B.

We assume that, at deployment, a set of session keys have been given to trusted fraglets.

Let us suppose to be at the antechamber of store B. Also, let us suppose to have a generic incoming fraglet ${}_A[send : AC_B : dec : s : tail_{K^t}]$ from store A, where K^t is the session key (to be used only once). The access to B will be granted iff $tail_{K^t}$ is something encrypted with one among the session keys not already used at store B.

All fraglets entering an antechamber must be encrypted (see $tail_{K^t}$ above). To do this, we must change the semantics of the $Send$ operation, with respect to the standard one recalled in Section 2, by distinguishing between a $Send*$ operation, that has an antechamber as a destination, and the classical $Send$, that transfers a fraglet from the antechamber to the actual store.

Op	$Input$
$Send*$	${}_A[send* : AC_B : dec : s : tail_{K^t}]$
$Send$	${}_{AC_B}[send : B : tail]$

Op	$Output$	
$Send*$	${}_{AC_B}[dec : s : tail_{K^t}]$	*tail must be encrypted*
$Send$	${}_B[tail]$	*tail can be a cleartext*

The execution steps for the access control mechanism are:

[....]	*any operation at store A leading to:*
${}_A[send* : AC_B : dec : s : tail_{K^t}]$	
${}_{AC_B}[dec : s : tail_{K^t}]$	${}_{AC_B}[s : K^t]$
${}_{AC_B}[s : tail]$	${}_{AC_B}[match : s : send : B]$
${}_{AC_B}[send : B : tail]$	
${}_B[tail]$	
	...we are now at store B...

We give some informal considerations about the security analysis of this construction.

The keys must be temporary. Indeed, suppose that the incoming fraglet is a malicious one. The first time it tries to enter store B, the access is denied, since it does not know a legitimate K^t, and decryption will not succeed. On the other hand, it will have the possibility to operate on part of the code devoted to control the access, *i.e.*, ${}_{AC_B}[s : K^t]$, and possibly discover the key. Thus, the key must be used only once.

Also, the model expects all the fraglets entering an antechamber to be encrypted. By requiring this, we avoid a very simple attack according to which the adversary can inject in the antechamber both a corrupted, encrypted fraglet ${}_{AC_B}[dec : s : tail^x_{K^x}]$ and the fraglet ${}_{AC_B}[s : K^x]$, representing the corresponding corrupted key K^x. Given the fraglets nature, these two will react, by passing the access control test. On the contrary, we force the injection of encrypted fraglets. By doing so, the only way to introduce $[s : K^x]$ at the antechamber is to encrypt K^x with one of the legitimate session keys.

However, our model is not robust enough against the following event. Suppose that a fraglet knows the right session key. It can correctly behave by encrypting $tail$ with that key, but it can also behave maliciously, since $tail$ can be of whatever nature, even a malicious $tail^x$.

Also, the model assumes to store at the antechamber as many session keys, as the number of the $Send*$ operations required in the life-cycle of the antechamber. Finally, an exhaustive search on the valid session key is required at the antechamber.

4.1 An alternative model

The previous access control mechanism maintains the same nature of asynchronous communication that is inherent to the fraglets approach. Indeed, given an encrypted $tail$, any incoming fraglet enters the antechamber with no control. Here, we present an alternative that makes the communication mechanism synchronous.

It is worth noticing that fraglets have something in common with formal languages for mobility such as Mobile Ambients, [5], proposed by Cardelli and Gordon for reasoning about properties of mobile processes. In mobile ambients, processes run inside an ambient, and processes within the same ambient may exchange messages. There are three primitives for movement: *in*, through which an ambient enters another ambient; *out*, through which an ambient exits another ambient; *open*, that dissolves an ambient boundary. From a security point of view, when processes are executed in parallel, there could be an interference leading to a sort of non determinism. As an example, during a parallel execution, the same ambient may be opened, but may also jump into another ambient. This resembles the unguarded nature of the $Send$ operation in fraglets.

To avoid these interferences, Levi and Sangiorgi introduce Mobile Safe Ambients, [6]. In particular, they introduce so called co-capabilities, according to which *in*, *out* and *open* are executed only if both the source and the destination agree.

The introduction of co-capabilities also for fraglets represents an interesting alternative for implementing the access control mechanism.

The idea is to let trusted stores allow a fraglet to enter if and only if a *guarded synchronization* occurs.

Broadly speaking, a synchronization occurs when two complementary instructions are executed, *e.g.*, send/ receive instructions. Then, a guarded synchronization occurs when two complementary actions are executed, depending on the outcome of some operation.

Within the fraglets world, one may think to allow the reception of $tail$ from A iff $tail$ satisfies some kind of policy P, for some fraglet populating store B. At conceptual level:

$$\begin{array}{ll} {}_A[send : B : tail] & \\ {}_B[receive : A : P(tail)] & {}_B[tail] \end{array}$$

An example policy is the following: access may be granted iff, on receiving from A something encrypted, B has the right decryption key K, *e.g.*, :

$$\begin{array}{ll} {}_A[send : B : dec : t : tail_K] & \\ {}_B[receive : A : K] & {}_B[dec : t : tail_K] \end{array}$$

Outcome ${}_B[dec : t : tail_K]$ is subject to an implicit capability of performing decryption with a certain key.

4.2 Trust management

The presented model can be further extended by allowing more complex reasoning. We could consider to enrich the model by allowing a credential-based access control, by encoding rules defining and combining credentials as instructions of the fraglet programming language.

Here, we consider the Role-based Trust Management Language (RTML) [7], as a model for trust management. In this language, credentials carry information on policies to define attributes of principals by starting from assertions of other principals. RTML inherits concepts of Role-based Access Control (RBAC) [11], by inheriting the notion of role, interposed in the assignment of permissions to users, and of trust management [2], by inheriting principles for managing distributed authority through credentials.

Thus, a central concept is the notion of role. A role is formed by a principal and a role term. If principals are denoted as $A, B, C...$ (usually the corresponding public key is used to identify such principals) and role terms are denoted as $r, r_1, r_2...$, then $A.r$ is role term r defined by principal A. A role may define a set of principals who are members of this role, and each principal A defines who are the members of each role of the form $A.r$. Also, roles can be seen as attributes, *i.e.*, a principal is a member of a role if and only if it has the attribute identified by the role, [7].

The basic statements in RTML are:

- $A.r \leftarrow D$ (simple member)
  A and D are (possibly the same) principals. Through this statement A says that D has role $A.r$ or, equivalently, attribute r.

- $A.r \leftarrow A.r_1.r_2$ (linking containment)
  This statement defines a linked role. If B has role $A.r_1$ and D has role $B.r_2$, then D has role $A.r$.

- $A.r \leftarrow A_1.r_1 \cap A_2.r_2$ (intersection containment)
  This statement defines that if D has both roles $A_1.r_1$ and $A_2.r_2$, then D has role $A.r$.

These three credentials are encoded in a fraglets style as follows.

SIMPLE	${}_A[cred_s : (A, r, D) : tail]$
LINKING	${}_A[cred_l : (A, r, A, r_1, r_2) : tail]$
INTERSECTION	${}_A[cred_i : (A, r, A_1, r_1, A_2, r_2) : tail]$

The rule for the linking operation is as follows. It takes as premises three credentials fraglets and it returns as output the fraglet consisting of the inferred credential.

$$\begin{array}{ll} In & Out \\ {}_A[cred_s : (A, r_1, B) : tail] & \\ {}_A[cred_s : (B, r_2, D) : tail] & {}_A[cred_s : (A, r, D) : tail] \\ {}_A[cred_l : (A, r, A, r_1, r_2) : tail] & \end{array}$$

Similarly, the rule for the intersection operation is:

$$\begin{array}{l} In \\ {}_{A}[cred_s : (A_1, r_1, D) : tail] \\ {}_{A}[cred_s : (A_2, r_2, D) : tail] \\ {}_{A}[cred_i : (A, r, A_1, r_1, A_2, r_2) : tail] \\ \\ \quad Out \\ \quad {}_{A}[cred_s : (A, r, D) : tail] \end{array}$$

These rules may be used to infer attributes. In particular, one could use them to pilot the receive action. The following sequence of rules allows the stores to receive fraglets from any source, provided that the source is trusted, denoted by the fact that the encryption key is trusted. Here, we make the over simplification that both the encryption key and the decryption one are the same. In a real implementation, we should use public key cryptography. This can be easily modeled within Crypto-fraglets. The generic store is denoted as !.

${}_{B}[cred_s : (B, receive, K) : tail]$
B trusts incoming fraglets encrypted by K.
This pilots a receive action:
${}_{!}[send : B : dec : t : tail_K]$
${}_{B}[receive :! : K]$
leading to:
${}_{B}[dec : t : tail_K]$

Then this fraglet may be decrypted as usual to get cleartext $tail$.

The model can be also extended with weights, as we did in [8] by obtaining quantitative notions of trust and reputation. We do not further develop this concept, due to the short nature of this paper.

5. CONCLUSIONS

In this paper, we have reasoned about fraglets and security. We have considered a threat model that leads us to reformulate Crypto-fraglets, with respect to the original model proposed in [10]. Also, we have added access control mechanisms to fraglets, by significantly enhancing their practical usage in a real context.

We plan to implement this framework within the BIONETS project, by also adding public key cryptography capabilities.

Acknowledgments

This work is partly supported by the FP6-027748 EU project BIONETS (*BIOlogically-inspired autonomic NETworks and Services*).

6. REFERENCES

[1] BIONETS website. http://www.bionets.eu/.

[2] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[3] I. Cervesato, N. Durgin, P. D. L. J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proc. CSFW-12*, pages 55–69. IEEE, 1999.

[4] FRAGLETS website. http://www.fraglets.net.

[5] L.Cardelli and A.D.Gordon. Mobile ambients. In *Proc. FOSSACS*, pages 140–155, 1998.

[6] F. Levi and D. Sangiorgi. Mobile safe ambients. *ACM Trans. Program. Lang. Syst.*, 25(1):1–69, 2003.

[7] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 1(11):35–86, 2003.

[8] F. Martinelli and M. Petrocchi. On relating and integrating two trust management frameworks. In *Proc. VODCA'06, ENTCS 168*, pages 191–205. Elsevier, 2007.

[9] A. Perrig and D. Song. Looking for diamonds in the desert - Extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proc. CSFW'00*, pages 64–76. IEEE, 2000.

[10] M. Petrocchi. Crypto-fraglets. In *BIONETICS*. IEEE, 2006.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, pages 38–47, 1996.

[12] D. Song, A. Perrig, and D. Phan. AGVI - automatic generation, verification, and implementation of security protocols. In *Proc. CAV'01, LNCS 2102*, pages 241–245. Springer, 2001.

[13] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

[14] C. Tschudin. Fraglets - a metabolistic execution model for communication protocols. In *Proc. AINS'03*, 2003.

[15] C. Tschudin and L. Yamamoto. A metabolic approach to protocol resilience. In *Proc. WAC'04, LNCS 3457*, pages 191–206. Springer, 2004.

[16] L. Yamamoto and C. Tschudin. Experiments on the automatic evolution of protocols using genetic programming. In *Proc. WAC'05, LNCS 3854*, pages 13–28. Springer, 2005.

[17] L. Yamamoto and C. Tschudin. Genetic evolution of protocol implementations and configurations. In *Proc. SelfMan'05*, 2005.