Tolérance aux fautes et sécurité par fragmentation-redondance-dissémination

Jean-Charles Fabre — Yves Deswarte — Laurent Blain

LAAS-CNRS & INRIA
7, avenue du Colonel Roche
31077 Toulouse cedex

RESUME : Cet article décrit une approche assurant des traitements fiables même en présence de fautes accidentelles ou intentionnelles. Cette approche, appelée Fragmentation-Redondance-Dissémination (FRD), fait l'objet de recherches au LAAS depuis plusieurs années. Elle consiste à utiliser la répartition pour tolérer les deux classes de fautes en utilisant les techniques de redondance, tout en préservant la confidentialité des informations traitées et stockées. La technique de FRD a d'abord été appliquée au stockage des fichiers persistants et utilisée pour implémenter des fonctions réparties de gestion de la sécurité (authentification et autorisation). A partir des leçons apprises lors de ces premières expériences, elle a ensuite été appliquée aux traitements manipulant des informations sensibles, en utilisant une approche de conception par objet. Cet article présente une synthèse de ces travaux.

ABSTRACT: This paper describes an approach to reliable computing in the presence of accidental or intentional faults. This approach, called Fragmentation-Redundancy-Scattering (FRS), has now been investigated at LAAS for several years. It consists of taking advantage of distribution to tolerate both classes of faults using redundancy techniques while preserving the confidentiality of the information being stored and processed. The FRS technique has been first applied to store persistent files and to implement distributed security management functions (authentication and authorisation). Taking into account lessons learnt from these first experiments, it has been applied to the processing of sensitive information by using an object-oriented design approach. This paper gives a survey of this work.

MOTS CLES : sûreté de fonctionnement, tolérance aux fautes, sécurité, confidentialité, systèmes répartis, développement par objets, serveur de fichiers, serveur de sécurité

KEYWORDS: dependability, fault-tolerance, security, confidentiality, distributed computing systems, object-oriented development, file server, security server

1. Introduction

La plupart des systèmes répartis tolérants aux fautes (tels qu'Isis [BIR 85], Argus [LIS 85], Newcastle Connection [BEN 85]) ont été conçus pour prendre en considération une classe limitée de fautes, à savoir les fautes accidentelles physiques qui surviennent durant le fonctionnement du système (voire même le plus souvent une classe restreinte de fautes telles que les "défaillances par arrêt"). Cependant,

d'autres classes de fautes peuvent empêcher le fonctionnement correct des systèmes répartis : actuellement une classe prépondérante est certainement celle des fautes intentionnelles humaines d'interaction, les intrusions. Celles-ci peuvent être définies comme des tentatives délibérées de transgresser la politique de sécurité du système. Elles peuvent avoir pour origine des intrus externes au système, des utilisateurs enregistrés essayant d'outrepasser leurs privilèges, ou encore des utilisateurs privilégiés tels que les administrateurs, les opérateurs, les responsables de la sécurité, etc., qui abusent de leurs privilèges pour réaliser des actions malveillantes.

Les intrusions et les fautes accidentelles peuvent avoir les mêmes effets : la modification ou destruction incorrecte d'informations sensibles, ou la révélation d'informations confidentielles. L'utilisateur percevra ces effets comme une défaillance du système : le service dévie de l'accomplissement de la fonction du système[1] [LAP 95]. Considérons un système réparti composé de stations de travail personnelles[2] et de serveurs partagés. Chaque utilisateur peut généralement avoir confiance dans sa propre station de travail en considérant qu'il la contrôle complètement, alors qu'il peut avoir des raisons de se méfier des serveurs partagés et des autres stations de travail car il ne sait pas directement si ces serveurs et ces stations de travail sont défaillants ou ont été pénétrés par un intrus. La confiance peut être améliorée si le système tolère les fautes, c'est-à-dire si la défaillance d'un serveur ou d'une station de travail n'a pas de conséquence, quelle que soit la cause de cette défaillance, une faute physique accidentelle ou une intrusion.

Si les techniques classiques de tolérance aux fautes telles que la réplication des données ou des traitements permettent de tolérer les fautes accidentelles, elles réduisent généralement la confidentialité en ne prenant pas en compte les intrusions. En effet, si les intrusions doivent être prises en compte et si la confidentialité des informations sensibles doit être maintenue, alors une simple réplication diminuera la confiance que l'on peut avoir dans le système puisque plusieurs répliques de l'information confidentielle peuvent être prises pour cible par une intrusion. Une technique a été développée au LAAS pour tolérer les fautes tout en préservant la confidentialité, à savoir la technique de Fragmentation-Redondance-Dissémination (FRD) [DES 91a]. La fragmentation consiste à découper toute information sensible en fragments de telle manière qu'un fragment isolé quelconque ne contienne pas d'information significative. Les fragments sont ensuite disséminés sur différents sites du système réparti non dignes de confiance, de telle manière que toute intrusion sur une partie du système donne seulement accès à des fragments sans rapport entre eux. De la redondance est ajoutée aux fragments (par réplication ou utilisation d'un code correcteur d'erreur) dans le but de tolérer les destructions et les altérations délibérées ou accidentelles de fragments. L'information complète ne peut être ré-assemblée que sur un site digne de confiance du système réparti, c'est-à-dire sur la station de travail de l'utilisateur.

Cet article est divisé en trois sections principales. Dans la section 2, nous décrivons une approche de conception d'applications par FRD et les services tolérants aux fautes fournis par le système, en prenant en compte des contraintes de

---

[1] Les exigences de performance, de fiabilité et de sécurité font partie des spécifications du système. Lorsque ces exigences ne sont plus satisfaites, le système est défaillant.

[2] Une station de travail personnelle peut soit être dédiée à un utilisateur, soit faire partie d'un pool de machines pouvant être utilisées par n'importe qui.

confidentialité. L'application de la FRD au stockage de fichiers persistants est développée dans la section 3. Les aspects fiabilité et confidentialité de la gestion de la sécurité sont présentés dans la section 4 et le serveur de sécurité qui a été conçu et implémenté est présenté en détail.

2. Fragmentation-Redondance-Dissémination

L'approche FRD fournit un cadre général de conception pour développer des applications fiables traitant des informations confidentielles en mettant à profit les propriétés de la distribution.

2.1. Architecture du système, hypothèses et problématique

L'architecture du système réparti que nous considérons (voir figure 1) dans cet article est composée d'un ensemble de stations de travail et de serveurs. Aucune de ces machines, prise individuellement, n'est considérée comme digne de confiance.

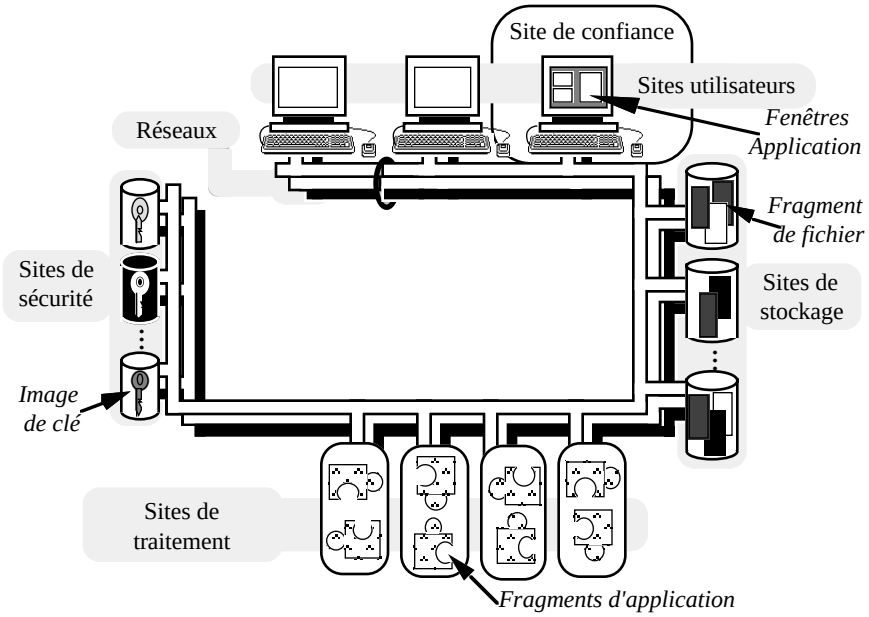

Figure 1. *Architecture du système réparti*

Cependant, pendant une session, l'utilisateur peut avoir confiance dans la station de travail qu'il utilise. En effet, chaque station peut être configurée de façon à empêcher l'accès à cette station autrement que depuis la console, et des moyens de protection peuvent être mis en œuvre pour en garantir l'intégrité physique, rendant impossible la modification du matériel et du logiciel de base. Les informations confidentielles et les logiciels d'application peuvent être chargés sur la station de travail pendant une session d'utilisation, mais ces informations sont détruites sur la

station à la fin de la session. Pendant la session, la station constitue donc une zone de confiance pour son utilisateur.

Des serveurs assurent le stockage des informations persistantes, ainsi que les traitements demandant des ressources trop coûteuses pour être présentes sur chaque station de travail. Ces serveurs sont partagés par l'ensemble des utilisateurs et répartis. Ils sont susceptibles d'être atteints par des fautes accidentelles et des intrusions, sans que les utilisateurs ne puissent s'en rendre compte directement. Pour que les utilisateurs aient confiance dans ce système réparti, il faut que ces serveurs tolèrent les fautes accidentelles et les intrusions. C'est précisément cette problématique que se propose de résoudre la FRD.

Sur la figure 1, nous présentons l'implémentation des services et des applications utilisant la FRD. Les " sites de sécurité " sont utilisés pour la gestion de la sécurité et sont responsables de l'authentification et du contrôle d'accès des utilisateurs. D'autres sites appelés " sites de stockage " sont responsables du stockage de fichiers persistants, c'est-à-dire en dehors des sessions utilisateurs. Enfin, les " sites de traitement " sont utilisés pour l'exécution d'applications qui sont conçues par la FRD pour traiter de manière fiable des données confidentielles. D'un point de vue pratique, un utilisateur est tout d'abord authentifié par les sites de sécurité, et il reçoit alors une information d'authentification nécessaire pour accéder aux services de traitement et de stockage.

En ce qui concerne les services de la FRD, nos hypothèses de fautes couvrent les fautes accidentelles, les fautes physiques qui affecteraient les sites partagés, mais aussi n'importe quel type d'intrusion qui toucherait les sites partagés ou les réseaux. Les actions malveillantes passives et actives sont prises en compte : par exemple celles qui consistent à essayer d'obtenir ou de modifier des informations statiques ou dynamiques sur n'importe quel site partagé (en incluant les segments de mémoire, les fichiers temporaires, les fichiers systèmes, le noyau du système d'exploitation), y compris par des administrateurs malveillants de ces sites. Ces attaques sont aussi considérées pendant le traitement des informations confidentielles.

Bien que nous admettions la possibilité d'intrusions sur les sites partagés, nous supposons néanmoins que de telles intrusions ne sont pas particulièrement aisées à réaliser, et que l'effort nécessaire pour un intrus afin de réussir des intrusions séparément sur plusieurs sites est proportionnel au nombre de sites impliqués. (Clairement, les mécanismes décrits dans ce papier ont pour but d'assurer que des intrusions réussies sur un seul ou un petit nombre de sites non dignes de confiance, les sites serveurs, ne donnent pas de moyen d'accéder ou de modifier des données ou des traitements qui sont sous la responsabilité de n'importe quel autre site.)

Enfin, cette approche ne présuppose pas un type particulier de politique de sécurité : différents types de politiques peuvent être mises en œuvre par les sites de sécurité qui contrôlent l'accès aux serveurs de traitement et de stockage. Ceci n'exclut pas d'implanter des mécanismes complémentaires à la FRD : ainsi, pour empêcher des flux d'informations qui ne seraient pas contrôlés par les sites de sécurité, il peut être nécessaire que chaque station de travail soit raccordée au réseau par des unités de contrôle de confiance, comme dans les systèmes Blacker [WEI 92], DSS [RUS 83] ou DS^3 [DAU 94].

2.2. *Conception d'applications FRD*

Le but premier de la technique de FRD est de fournir une méthode générale pour le traitement fiable de données confidentielles sur des sites non dignes de confiance. On s'intéresse ici uniquement à la confidentialité des informations manipulées par un programme d'application lors de son exécution (les données), et non pas à la confidentialité du programme lui-même. La confidentialité repose sur la sémantique des données et elle doit être prise en compte durant le processus de conception. Cette prise en compte doit être basée sur les hypothèses faites sur l'architecture du système, tant du point de vue de la confiance que l'on attribue à ses différents composants, que de celui des fautes accidentelles qui peuvent les affecter. Complémentairement aux techniques mises en œuvre, la nature répartie du système peut alors être mise à profit lors du processus de conception comme moyen de séparation, d'isolation, et de réplication de l'information ainsi que son traitement à des fins de tolérance aux fautes intentionnelles et accidentelles.

2.2.1. Principe du traitement fiable d'informations confidentielles

Pour toute application ou service du système, l'utilisation de la FRD consiste à concevoir le logiciel sous une forme fragmentée selon plusieurs règles de base :

- l'application (code et données inclus) est divisée en fragments d'applications de telle manière que la coopération des fragments satisfasse les spécifications initiales de l'application complète ;
- un fragment d'application quelconque ne doit fournir aucune information confidentielle à un intrus potentiel sur le site où le fragment réside ;
- tous les fragments d'application doivent être disséminés sur les sites d'un système réparti (séparation) de telle manière que des groupes de fragments stockés sur un site donné ne fournissent aucune information significative à un intrus ;
- une redondance appropriée doit être introduite, soit durant la fragmentation, soit durant la dissémination ;
- autant que possible, un intrus ne doit pas être capable d'identifier des fragments appartenant au même processus d'application ou au même objet ; pour obtenir ceci, les fragments d'application doivent être identifiés à partir du site utilisateur par des références chiffrées générés par une fonction à sens unique[3].

L'opération de fragmentation est réalisée par le concepteur de l'application. Nous considérons que l'application utilisateur peut être perçue à un niveau d'abstraction donné comme une collection d'entités fonctionnelles. Pendant l'exécution de l'application, toute entité de base (c'est–à–dire un fragment d'application) peut être projetée sur une unité d'exécution autonome du système d'exploitation réparti[4] ; chaque fragment traite des informations qui peuvent être des données persistantes (c'est–à–dire des informations locales privées) ou des données externes reçues par des

[3] Une fonction à sens unique est une fonction cryptographique basée sur des algorithmes non inversibles. Cela signifie qu'à partir d'une clé secrète et d'une référence en clair, il est facile de générer la référence chiffrée, mais qu'à partir de la référence chiffrée, il est impossible de retrouver la référence en clair.

[4] La projection d'entités de conception (blocs, modules, objets, ...) sur des entités d'exécution dépend de l'interface de l'environnement d'exécution sous-jacent (processus, tâches, acteurs, objets, ...). La transparence de la distribution est intéressante pour faciliter l'interaction entre les fragments, mais la visibilité de la localisation est nécessaire pour la réplication et la dissémination des fragments.

paramètres d'entrées. Un fragment d'application reçoit des informations d'entrée, les traite en fonction de son état local persistant et délivre des résultats à d'autres fragments d'application par messages. L'objectif de l'opération de fragmentation est de concevoir l'application de telle manière que les fragments d'application résultants ne conservent, ni ne collectent d'information confidentielle durant l'exécution. Les fragments qui manipulent des informations confidentielles durant l'exécution doivent résider sur le site digne de confiance, c'est–à–dire la station de travail de l'utilisateur d'où l'application a été lancée.

Notre approche est basée sur un processus de conception récursif qui opère sur une représentation hiérarchique de l'application pour définir des fragments d'application. Chaque pas consiste à raffiner la conception des entités confidentielles pour obtenir des entités plus élémentaires ; la récursion se termine dès que, sur chaque branche de l'arbre de conception, une entité qui ne traite plus aucune information confidentielle est rencontrée. Cette entité est un fragment puisque, même non chiffrée, elle ne représente pas une information significative. Les fragments d'application sont ensuite disséminés sur l'architecture répartie et communiquent par messages. La fragmentation peut, par elle-même, introduire de la redondance : par analogie avec les codes correcteurs d'erreurs, si l'information confidentielle peut être restaurée à partir d'un sous–ensemble de fragments, la modification ou la destruction de quelques fragments n'empêche en rien un traitement correct (voir section 2.2.2). Au contraire, si le processus de fragmentation ne fournit aucune redondance, alors les fragments sont répliqués lors de la dissémination.

2.2.2. *La confidentialité de l'information*

L'identification d'informations confidentielles dans une application est basée sur les spécifications qui précisent la sémantique de l'information vis-à-vis de la confidentialité. Ces spécifications sont appelées *contraintes de confidentialité* et sont utilisées comme entrées du processus de conception. Une information peut-être confidentielle par sa valeur même, par exemple une variable chaîne de caractères telle que " *Le salaire de mon patron est supérieur à 1.000.000 F par mois!* ". Mais en général, la sémantique de l'information ne peut être interprétée que selon la structuration des données en termes de sa représentation interne, les opérations sous-jacentes (le type des données) et la valeur de l'information elle-même. Cette sémantique ne peut pas être interprétée sans une connaissance de sa structure ou de son type. Par exemple, une chaîne de bits correspondant à une variable de salaire dans le segment des données d'un programme doit être interprétée par un intrus à partir de sa représentation réelle dans la machine. Cependant ceci n'est pas suffisant puisqu'une information confidentielle est en fait un ensemble d'éléments dont seule la combinaison peut révéler l'information à un intrus potentiel. Ce dernier peut obtenir l'information sur le salaire si et seulement si il est capable d'associer plusieurs informations telles que le nom de la personne, le montant du salaire, la période sur laquelle porte le salaire et la monnaie. Cette idée peut être illustrée par un autre exemple très simple : supposons qu'une réunion d'un groupe de personnes soit une information confidentielle. L'information réunion est composée d'un ensemble d'éléments, une *liste de participants*, un *sujet* donné, un *lieu*, *l'heure et la date*. Un participant est défini par son identité personnelle qui peut être considérée comme une information publique : la même hypothèse peut être faite pour les autres éléments tels que le lieu. En fait, l'information à propos de la réunion est confidentielle à cause du sujet discuté et à cause de la liste des participants. Une

solution possible entraîne la dissémination de la liste des participants. Les opérations sur les informations relatives aux participants sont réalisées sur différents sites du réseau.

Ces exemples simples montrent qu'*une information confidentielle est souvent constituée d'un ensemble d'éléments qui, pris isolément, sont non-confidentiels. Un nombre donné d'éléments est nécessaire pour reconstruire l'information confidentielle*.

Cependant, ceci n'est pas toujours vrai et quelques informations ne peuvent être décomposées en éléments plus petits. Dans ce cas, d'autres techniques, comme le chiffrement ou les schémas à seuil définis par Shamir [SHAM 79], doivent être appliquées à des informations non-structurées, telles que des chaînes de caractères, des clés cryptographiques, des fichiers, etc. (par exemple, le sujet confidentiel de la réunion dans l'exemple indiqué ci-dessus devrait être chiffré). La technique des schémas à seuil consiste à rassembler un nombre d'éléments (les *images*) pour reconstruire le secret : *N* images sont calculées à l'aide d'une fonction polynomiale dans un corps fini ; si *T-1* est le degré du polynôme, il suffit de *T* images pour calculer le secret (N>T). Dans ce cas, la fragmentation assure la redondance : le coût de la redondance est proportionnel à *N-T*. Cette technique, assez proche dans son principe de la FRD, peut être utilisée pour des clés cryptographiques, mais aussi pour toute information de petite taille.

L'application de la FRD à des applications traitant de l'information confidentielle consiste (i) à définir une structuration appropriée afin que chaque information confidentielle soit découpée en éléments non-confidentiels, et (ii) à sélectionner la technique appropriée pour les objets confidentiels non-structurés.

2.2.3. *Une vue " objet " de la FRD*

Différentes techniques de fragmentation basées sur une conception structurelle des applications ont été identifiées et expérimentées : décomposition de blocs, parallélisation, traitement en tranches de bits [TRO 91a, TRO 91b, TRO 91c]. Un problème majeur avec ces premières techniques de FRD fut celui des communications entre fragments de code. Mais c'est le problème des variables globales qui fut une des premières motivations pour l'utilisation de techniques à objets [FAB 92, FAB 94]. Le modèle objet utilisé ici n'est pas spécifique à un langage particulier de programmation par objet : nous supposons simplement que les *objets* sont dérivés de *classes* et encapsulent des structures de données qui peuvent être seulement manipulées par un ensemble de fonctions (*méthodes*) ; les *objets* peuvent être décomposés en *sous–objets* qui sont identifiés par des *références*. L'approche FRD par objet permet de concevoir une application selon les règles de base énoncées au paragraphe 2.2.1.

Le principal intérêt du modèle objet, sur lequel repose notre approche, est qu'il permet de concevoir une application en une collection d'objets de telle manière que la plupart des objets, pris individuellement, ne traitent pas d'information confidentielle. La notion d'objet est en fait intéressante pour plusieurs raisons. Habituellement, un objet informatique modélise un objet réel dont on connaît la sémantique, et auquel on peut facilement attribuer un caractère confidentiel en fonction des spécifications. A un niveau d'abstraction donné, un objet confidentiel peut être substitué ou décomposé en un ensemble d'objets ou de sous-objets, avec pour objectif de définir des objets (sous-objets) non-confidentiels. Lors de la décomposition, certains des objets obtenus restent confidentiels : il s'agit d'objets qui possèdent des références

sur des objets dont l'agrégation est confidentielle, mais aussi d'objet qui sont confidentiels compte tenu de leur sémantique intrinsèque. Ces derniers sont alors analysés selon la même démarche. Ainsi, l'approche de conception par objet permet d'analyser à différents niveaux d'abstraction, la confidentialité d'une application en termes d'objets. Selon cette approche de conception récursive prenant en considération la confidentialité des objets, on obtient une organisation de l'application sous la forme d'une collection d'objets dans laquelle les objets confidentiels sont parfaitement identifiés. La récursion se termine dès lors que les objets confidentiels ne sont pas décomposables : ce sont des objets non structurés, ou bien de petite taille et sur lesquels les méthodes correspondent à des opérations élémentaires. Des techniques complémentaires de chiffrement ou de schéma à seuil peuvent être alors envisagées. Enfin, les objets possédant des références sur des objets dont l'agrégation est confidentielle seront conservés sur le site de confiance. Les objets non-confidentiels pourront être disséminés et exécutés sur des sites non dignes de confiance. Dès lors que les objets confidentiels, y compris ceux qui possèdent des liens sur des objets non-confidentiels, sont exécutés sur un site de confiance, la sécurité de l'application est assurée.

```
pour tout <objet > dans l'ensemble d'objets courant
faire
  si <objet> confidentiel alors
    si <objet> décomposable alors
          décomposition en sous-objets (fragmentation)
    sinon
        soit application de techniques de chiffrement ou de schémas à seuil
        soit marquage d'<objet> pour allocation sur le site de confiance
    fin_si
  fin_si
fin_pour
```

Figure 2. *Conception d'applications par fragmentation d'objets confidentiels*

La figure 2 résume de façon très synthétique la démarche générale d'une *itération de conception* à un niveau d'abstraction donné ; celle-ci se répète à chaque niveau d'abstraction dès lors qu'il existe des objets confidentiels décomposables.

En fait, le modèle objet est lié aux deux aspects de l'utilisation de la FRD : la fragmentation des informations confidentielles qui sont traitées et les opérations (les méthodes) réalisées sur celles-ci. Enfin, tout objet produit par la démarche de conception peut facilement être projeté sur une unité d'exécution autonome (notion d'objet actif).

La réplication peut être introduite soit pendant la phase de configuration, c'est-à-dire lors du lancement des exécutables, sur un système réparti tolérant les fautes (comme le système Delta–4 [POW 91]), ou plus tôt durant la conception en utilisant *l'héritage* de classes systèmes (comme dans le système Arjuna [SHR 91]), ou enfin en utilisant des techniques à objets plus avancées telle que la *réflexivité* [MAE 87, STR 92], cette dernière faisant actuellement l'objet de recherches qui ont déjà donné des résultats significatifs [FAB 95a]. En fait, cette propriété propre à quelques langages de programmation par objets permet de contrôler à un méta-niveau de programmation le comportement des objets applicatifs, en redéfinissant par

exemple certaines opérations de base, telles que la " création d'objet "[5] ou " l'invocation de méthode "[6]. Ce concept apporte une transparence complète de la FRD du point de vue de la programmation fonctionnelle. En effet, tout objet réflexif est associé à un méta-objet (instance d'une classe de méta-niveau) qui gère la tolérance aux fautes, en particulier lorsqu'il existe plusieurs exemplaires, indépendamment de la programmation de l'objet d'application lui-même. L'héritage appliqué aux méta–classes est alors utilisé pour définir et raffiner le comportement approprié de n'importe quel objet de l'application.

La démarche complète de conception et de mise en œuvre d'une application par l'approche FRD est résumée sur la figure 3.

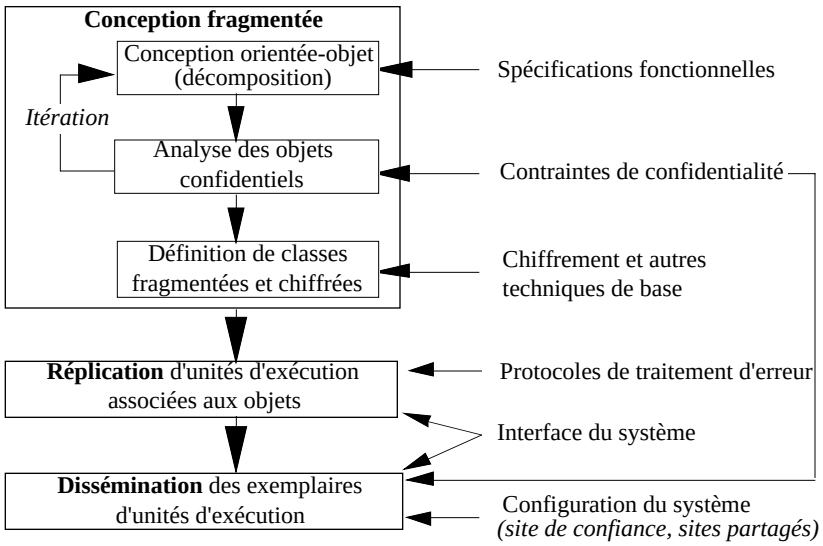

Figure 3. *Démarche FRD de conception et de mise en œuvre*

L'approche FRD peut être utilisée de différentes manières et pour divers types d'applications. Par exemple, dans les applications transactionnelles, une grande quantité d'information confidentielle peut être conservée dans des objets persistants, mais dans ce cas, la quantité de traitement peut être relativement limitée (système de gestion de dossiers médicaux, par exemple). A l'opposé, d'autres applications, tels que les calculs numériques, conduisent à des traitements très intensifs, mais les objets sont principalement temporaires parce qu'il n'y a pas d'état persistant et que tous les paramètres d'entrée peuvent être donnés pour chaque exécution (simulation, par exemple). De tels objets sont créés dynamiquement suivant les besoins ; ils sont détruits après le calcul.

[5] La création est redéfine pour gérer la réplication et la dissémination des sous-objets.

[6] L'invocation est redéfinie pour gérer les références chiffrées entre les objets, les mécanismes de contrôle d'accès, ainsi que l'accès à des objets répliqués.

L'approche par objet pour l'utilisation de la FRD est donc attirante pour implémenter différents types d'applications qui conservent et traitent des informations confidentielles. Cette approche a été expérimentée pour la conception d'un Agenda Électronique Réparti [FAB 92, FAB 94] et implémentée sur le système Delta–4 [POW 91]. Une description plus détaillée de la conception fragmentée d'objets confidentiels peut être consultée dans [FAB 95b].

2.4. *Comparaison avec d'autres techniques*

Le problème de la tolérance aux intrusions dans le traitement d'informations confidentielles a fait précédemment l'objet de recherches en utilisant des techniques de chiffrement tels que les "homomorphismes secrets" décrits dans [RIV 78]. Cette approche ne tient pas compte de la sémantique des données de l'application. La difficulté d'utilisation de telles techniques réside d'abord dans la recherche de chiffres efficaces pour lesquels il existe des homomorphismes pour toutes les opérations arithmétiques et logiques. De plus, il a été montré que des attaques simples peuvent permettre d'obtenir l'information en clair [RIV 78, AHI 87]. Enfin, ces techniques posent d'autres problèmes d'implémentation, en particulier pour la distribution des clés.

Notre approche est totalement différente et repose sur une démarche conceptuelle. Elle est basée sur l'utilisation de la sémantique de l'information confidentielle pendant la phase de conception et profite de la répartition. Elle repose sur l'observation simple que très souvent une information confidentielle peut être représentée par une collection d'éléments publics, des techniques de chiffrement fournies sont utilisées pour générer les références sur le site utilisateur en utilisant de l'information secrète afin d'empêcher des intrus de lier les fragments apparentés entre eux. Mais, pour quelques applications, les informations confidentielles ne peuvent pas être considérées comme un ensemble d'éléments publics particuliers, soit à cause de leur type, soit à cause de leur granularité. En particulier, lorsqu'il faut traiter de gros objets non-structurés, tels que des fichiers, d'autres techniques de FRD doivent être développées et appliquées.

3. Stockage de fichiers persistants

Un serveur de fichiers dans un système réparti est un exemple typique d'une telle application. Deux opérations de base sont à considérer : l'opération de *lecture* entraîne la copie d'un fichier persistant du serveur de stockage vers un fichier temporaire sur la station de travail utilisateur : l'opération *d'écriture* consiste à transférer des données depuis un fichier temporaire sur la station de travail vers un fichier persistant sur le serveur de stockage. Les fichiers appartiennent à différents utilisateurs et la sémantique des données est inconnue, a priori, et bien sûr non décrite dans les spécifications du serveur de fichiers.

Ce serveur est conçu de manière à stocker les fichiers persistants des utilisateurs de manière sûre, de façon à ce qu'un utilisateur puisse se connecter sur n'importe quelle station de travail et y récupérer ses propres fichiers. Ces fichiers sont vus par le système comme des flux d'octets non-structurés, par exemple comme des fichiers Unix standard. Le serveur est composé de plusieurs sites de stockage de fragments qui reçoivent/transmettent les fragments de/vers les stations de travail. La fragmentation

et le ré-assemblage sont exécutés sur le site utilisateur, afin qu'en dehors de ce site, l'information soit fragmentée. La redondance consiste à générer plusieurs exemplaires de chaque fragment, et la dissémination est réalisée en envoyant les différents exemplaires des fragments sur les différents sites de stockage.

3.1. *Principes*

Tandis que la sémantique du traitement des données peut être utilisée pour définir la granularité des fragments de manière à ce qu'aucune information significative ne soit conservée par un fragment, ceci ne peut être utilisé pour le stockage de données génériques, c'est-à-dire des données dont la sémantique est inconnue : la confidentialité de toute donnée, même aussi petite qu'un bit, est inconnue ; donc, quelle que soit la granularité du fragment, il n'y a aucun moyen d'assurer qu'un fragment en clair ne contiendra pas une information intéressante pour un intrus potentiel. C'est pourquoi il est nécessaire de chiffrer les données avant de les fragmenter. Ceci peut être fait pour la fonction de stockage qui ne transforme pas les données, tandis que cela aurait été difficile pour une fonction de traitement (voir §2.). Même si les données sont chiffrées, la FRD est utile pour renforcer la confidentialité des données, leur intégrité et leur disponibilité :

- confidentialité : un intrus aurait à rassembler tous les fragments dans l'ordre correct avant de tenter une cryptanalyse,
- intégrité et disponibilité : un intrus devrait modifier ou détruire de manière cohérente toutes les répliques d'un fragment pour pouvoir modifier ou détruire des données.

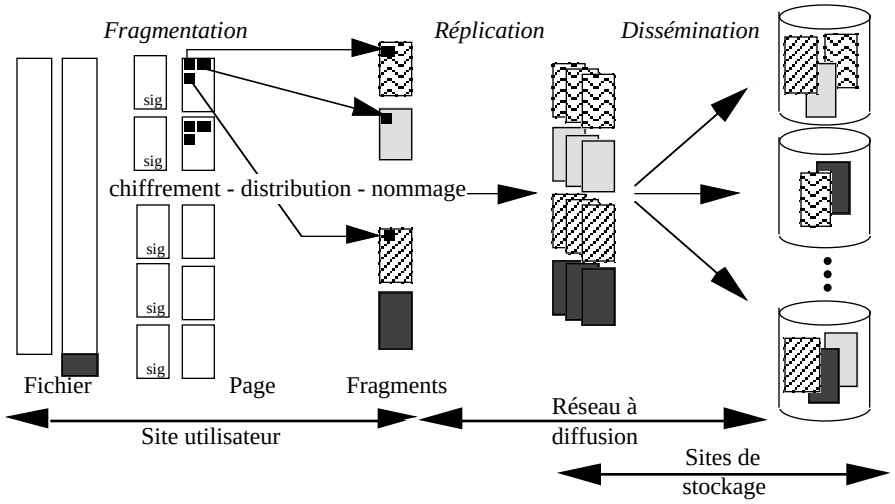

Figure 4. *La FRD appliquée aux fichiers*

La figure 4 montre les principales étapes du protocole de fragmentation des fichiers persistants. Les fragments sont disséminés sur un certain nombre de sites de

stockage sur lesquels un algorithme réparti décide s'il doit stocker ou non un exemplaire de chaque fragment.

L'application de la technique de FRD à un serveur de stockage de données génériques implique les opérations suivantes :

- chiffrement avant fragmentation,
- gestion sécurisée de la clé de chiffrement,
- fragmentation des données chiffrées en fragments de taille fixe,
- nommage sécurisé des fragments,
- diffusion des fragments à partir du site utilisateur vers tous les sites de stockage,
- exécution sur les sites de stockage d'un algorithme réparti pour sélectionner quels sites stockeront effectivement chaque fragment (voir §3.3).

3.2. *Fragmentation*

La fragmentation est réalisée sur le site utilisateur (cf. figure 5).

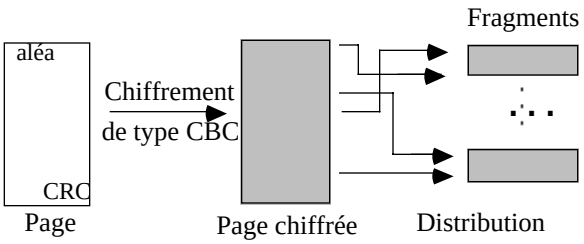

Figure 5. *Fragmentation d'une page*

Chaque fichier local est tout d'abord découpé en pages de taille fixe (si la taille du fichier n'est pas un multiple de la taille d'une page, du bourrage est rajouté à la fin du fichier pour remplir la dernière page). La taille d'une page est choisie comme un multiple de la taille d'un fragment. L'avantage de cette pagination est qu'elle permet qu'un utilisateur n'ait pas besoin de ré-assembler un fichier complet si une seule page est nécessaire. Une somme de contrôle est ajoutée à chaque page, et est vérifiée par l'opération de lecture pour contrôler l'intégrité de la page. Un nombre aléatoire est aussi inséré au début de la page pour rendre la cryptanalyse plus difficile. Ensuite la page est chiffrée à l'aide d'un "chiffre à clé courante" tel qu'un chiffrement par chaînage de blocs (CBC), utilisant la clé de chiffrement, et les octets de la page chiffrée sont répartis régulièrement dans les fragments : le premier octet de la page chiffrée est le premier octet du premier fragment, le deuxième octet est le premier octet du deuxième fragment, et ainsi de suite.

Puisque les données sont chiffrées, la granularité des fragments n'est pas déterminée par des contraintes de confidentialité : la taille des fragments est arbitraire et peut être choisie pour améliorer les performances du système[7]. La taille d'un

[7] La granularité des fragments peut être choisie pour améliorer soit le transfert des fragments sur le réseau (un fragment = un paquet), soit le stockage sur disque (un fragment = un secteur).

fragment peut être fixée pour tous les fragments stockés sur le serveur afin qu'il soit plus difficile pour un intrus d'identifier ceux appartenant au même fichier. De manière similaire, l'identification des fragments (références ou noms des fragments) ne doit pas permettre à un intrus de savoir quels fragments appartiennent à quel fichier. De plus, si les données sont chiffrées, la clé de chiffrement est une information sensible qui doit être stockée de manière sûre afin de maintenir la confidentialité et l'intégrité des données chiffrées (ceci est étudié dans la section 4).

Le chiffrement peut être très rapide étant donné que la sécurité repose plus sur la fragmentation que sur le chiffre lui-même : supposons qu'un intrus soit capable de rassembler tous les fragments d'une page donnée par écoute passive du réseau ou en prenant le contrôle d'un site de stockage ; si une page est découpée en N fragments, et si l'intrus ne connaît pas l'ordre des fragments, alors approximativement $N! / 2$ cryptanalyses seront nécessaires pour reconstituer la page en clair. Par exemple, si $N = 16$, il y a $16! \approx 2.10^{13}$ permutations possibles, soit autant de pages chiffrées possibles, l'intrus ayant à reconstituer et à décrypter environ la moitié d'entre elles pour obtenir la page en clair.

Pour empêcher un intrus de connaître l'ordre correct des fragments d'une page donnée, le nom de chaque fragment est généré par une fonction à sens unique à partir de la clé de chiffrement, du nom du fichier, de l'index de la page et de l'index du fragment. Ainsi aucune information sur un fragment ne peut être dérivée à partir de son nom.

3.3. *Réplication et dissémination*

Lorsqu'un fichier est écrit sur le serveur de fichiers réparti, le site utilisateur diffuse dans un ordre aléatoire tous les fragments de chaque page vers tous les sites de stockage. Pour chaque fragment, les sites de stockage exécutent un algorithme réparti pour sélectionner les sites qui stockeront un exemplaire du fragment. Le nombre d'exemplaires est déterminé par l'utilisateur selon la criticité du fichier. L'algorithme réparti peut être basé sur des générateurs de nombres pseudo-aléatoires, pondérés par l'espace disponible sur les dispositifs de stockage locaux. Le stockage des fragments peut être optimisé par rapport au stockage de fichiers traditionnels : sur les sites de stockage, chaque fragment est identifié par son nom dans un répertoire unique (sans sous-arborescence), et tous les fragments ont la même taille ; ils n'ont pas de propriétaire ni de date de création. Le temps d'accès à un fragment peut donc être beaucoup plus court que le temps d'accès à un fichier traditionnel. Par ailleurs, différents sites de stockage peuvent accéder à des fragments en parallèle, ce qui conduit à un débit global supérieur à celui de serveurs de fichiers classiques.

3.4. *Ré-assemblage*

Lors de l'opération de lecture, le site utilisateur reconstitue les noms de tous les fragments des pages qu'il doit récupérer (à l'aide de la fonction à sens unique), et diffuse dans un ordre aléatoire les requêtes de lecture vers tous les sites de stockage. Ceux qui possèdent un exemplaire du fragment le renvoient au site utilisateur. Si tous les exemplaires de chaque fragment sont identiques, le site utilisateur peut facilement reconstituer la page chiffrée, la déchiffrer et vérifier la somme de contrôle (qui est correcte, sauf si tous les exemplaires d'un fragment ont été altérés de manière

identique). Si des exemplaires différents existent pour un fragment, le site utilisateur reconstitue plusieurs pages chiffrées et essaie de les déchiffrer jusqu'à ce qu'il obtienne une somme de contrôle correcte. Ainsi, le fichier peut être ré-assemblé même si *C-1* sites de stockage sont défaillants où s'ils ont été pénétrés par un intrus (*C* étant le nombre d'exemplaires assigné à ce fichier par l'utilisateur).

Remarquons ici qu'aucune liste ou table d'implantation des fragments n'est stockée dans le système. La distribution des fragments sur les sites de stockage est aléatoire. Seul le nom des fragments permet de retrouver leur localisation et cette information est calculée lors de la fragmentation (à partir d'informations telle que la clé de fragmentation, le nom du fichier, etc.) et re-calculée dynamiquement à partir de ces mêmes informations lors de l'opération de ré-assemblage.

3.5. *Comparaison avec des techniques similaires*

Des techniques similaires ont été développées dans [RAB 89]. L'algorithme proposé de dispersion de l'information (*Information Dispersal Algorithm (IDA)*) combine fragmentation et redondance ; les fragments sont supposés être disséminés sur plusieurs sites. L'objectif de cette approche intéressante est principalement de tolérer des fautes accidentelles avec moins de redondance que la réplication. Ceci peut être relié aux techniques de codes correcteurs d'erreurs mais aussi aux schémas à seuil, puisque *M* fragments parmi *N* (*N* étant le nombre total de fragments produits) doivent être ré-assemblés pour reconstituer le fichier. Des expérimentations de l'IDA ont été réalisées sur des architectures massivement parallèles, parce que la technique entraîne des calculs sur des matrices de grande taille ; il nécessite aussi une identification correcte des fragments avant d'être capable d'établir la matrice conduisant au ré-assemblage du fichier. Cette technique de fragmentation assure une optimisation de l'espace disque au détriment d'un coût de calcul élevé.

En fait, les fonctionnalités de l'IDA correspondent aux étapes de la distribution des octets dans les fragments et de réplication de la technique de FRD appliquée aux fichiers. Du point de vue de la confidentialité, un chiffrement est néanmoins nécessaire avant la fragmentation. Comme dans l'approche FRD, les opérations telles que la pagination, le chiffrement, le nommage des fragments, la gestion des clés et la dissémination sont encore nécessaires pour une mise en œuvre opérationnelle de la technique IDA. Finalement, la FRD assure de meilleures performances du point de vue des communications puisque la quantité de données transmises à partir du site utilisateur correspond seulement à la transmission d'un fichier grâce aux mécanismes de diffusion sous-jacents [DES 91b].

4. Gestion de la sécurité

La FRD a aussi été appliquée avec succès à une classe hautement sensible de traitements, à savoir la gestion de la sécurité d'un système réparti. Les fonctions correspondantes incluent l'enregistrement et l'authentification des utilisateurs, l'autorisation (c'est-à-dire le contrôle d'accès aux objets ou aux serveurs), la journalisation des opérations liées à la sécurité, etc. Ces fonctions reposent sur le stockage et la gestion de données sensibles telles les informations relatives aux utilisateurs (identité, authentifiant), les droits d'accès, les clés de chiffrement, les journaux, etc. Certaines sont confidentielles (par exemple, les clés de chiffrement) et

doivent être fragmentées, tandis que d'autres le ne sont pas (par exemple, les identités des utilisateurs) et peuvent être répliquées.

Afin de permettre aux fonctions de sécurité de tolérer les fautes et les intrusions même si les intrusions sont réalisées par des administrateurs de sécurité, ces fonctions sont implémentées sous la forme d'un serveur de sécurité réparti[8] composé de plusieurs sites de sécurité gérés par différents administrateurs. L'implémentation est basée sur des protocoles à vote majoritaire et des algorithmes de schéma à seuil pour assurer que, tant qu'une majorité de ces sites n'est pas défaillante, ni pénétrée par un intrus, les fonctions de sécurité seront réalisées correctement et aucune donnée confidentielle ne sera révélée.

4.1. *Enregistrement des utilisateurs et authentification*

Pour être connu dans le système réparti, un utilisateur doit être enregistré par tous les sites de sécurité (ou au moins par une majorité d'entre eux). Cet enregistrement est réalisé séparément sur chaque site de sécurité sous le contrôle de l'administrateur de sécurité local. Pendant cette phase, un " authentifiant " est stocké pour cet utilisateur sur chaque site de sécurité. Selon les mécanismes d'authentification, l'authentifiant peut être un secret partagé par l'utilisateur et le site de sécurité (un mot de passe), ou des informations biométriques caractérisant l'utilisateur (une empreinte digitale), ou des informations publiques correspondant à un secret connu uniquement par l'utilisateur (authentification à apport nul de connaissance [BLA 94]). Les schémas d'authentification peuvent différer pour chaque site de sécurité de manière à améliorer l'efficacité de l'authentification : il sera plus difficile pour un intrus de se faire passer pour un autre utilisateur s'il doit tromper plusieurs mécanismes différents. Mais le même schéma peut aussi être implémenté sur tous les sites de sécurité avec différents authentifiants si le schéma d'authentification est basé sur des secrets partagés, et avec la même information publique répliquée dans le cas d'authentification à apport nul de connaissance. Sur le prototype que nous avons réalisé, l'authentification est basée sur différents secrets partagés stockés et vérifiés sur des cartes à puce (voir figure 6) : la carte de l'utilisateur contient différents secrets pour chaque site de sécurité, tandis que l'administrateur d'un site de sécurité possède une carte à puce qui permet de reconstruire un secret partagé et un seul pour chaque utilisateur enregistré [BLA 94].

La figure 6 montre le protocole d'authentification tel qu'il est implémenté. Un utilisateur se connecte avec sa carte à puce Bull CP8 contenant différents authentifiants. Chacun est écrit sur la carte et connu par un seul site de sécurité. Notre système impose un enregistrement et une authentification séparément pour chaque site de sécurité : un administrateur de sécurité malveillant ne peut donc, sans l'aide des autres administrateurs, se faire passer pour un utilisateur (il ne connaît pas les autres authentifiants), et il ne peut pas créer un nouvel utilisateur (celui-ci ne serait pas reconnu par les autres sites de sécurité). Son autorité se limite à son propre site de sécurité et il n'a aucun pouvoir sur les autres sites de sécurité. Cette approche assure donc une certaine séparation des pouvoirs.

---

[8] Différentes fonctions peuvent être implémentées sur différents serveurs répartis comme il est montré dans [DES 91], ou rassemblées sur le même serveur comme dans [BLA 90].

Lorsque l'utilisateur enregistré se connecte à une station de travail, une requête d'authentification (①) est diffusée vers tous les sites de sécurité, et chaque site de sécurité exécute un protocole d'authentification indépendant selon le schéma d'authentification local (②) afin de décider localement si l'identité proclamée de l'utilisateur est correcte ou non. Cette décision locale est ensuite diffusée vers les autres sites de sécurité. Chaque site vote alors avec sa décision et celles qu'il a reçues (③). Si la décision majoritaire est positive, chaque site de sécurité (non-défaillant) reconnaît l'utilisateur comme authentifié pour une session (④), et ensuite celui-ci peut envoyer des requêtes pour accéder à d'autres serveurs ou objets du système réparti. L'utilisateur obtient aussi des clés de session, une par site de sécurité, qui serviront pour les accès futurs vers chaque site de sécurité ; toutes les autres requêtes, dans le protocole d'autorisation, par exemple, seront chiffrées par ces clés de session.

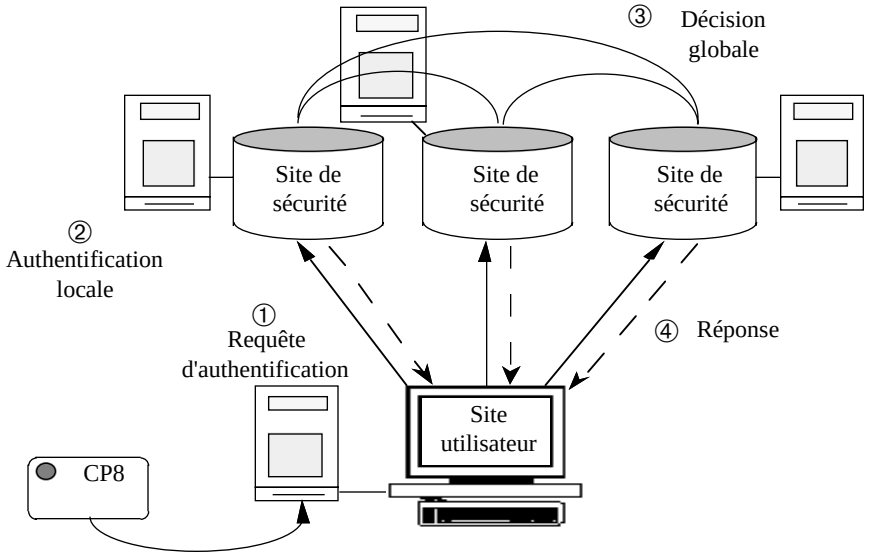

Figure 6. *Le serveur de sécurité (authentification)*

4.2. *Autorisation*

Pour accéder à un objet (par exemple, un fichier persistant) ou un serveur (par exemple, un serveur de traitement), le site utilisateur transmet sa requête (quand l'utilisateur est authentifié) au serveur de sécurité qui autorise ou refuse l'accès (voir figure 7). En fait, cette requête est diffusée vers tous les sites de sécurité (①), et chaque site décide localement si l'accès doit être refusé ou accordé, en fonction des droits d'accès, stockés localement, pour cet utilisateur sur l'objet. Ensuite les décisions locales sont échangées. Chaque site vote sur les décisions (②), et si une majorité d'entre elles est positive l'accès est autorisé. Un premier ticket, incluant une image de clé, est transmis au site utilisateur par chaque site de sécurité (③). Le site utilisateur obtient ainsi plusieurs images utilisées pour reconstruire les clés secrètes

d'application, par exemple, la clé de fragmentation du fichier. Un autre ticket est envoyé au serveur de traitement ou au serveur de stockage (④). Les droits d'accès sont stockés sur les sites de sécurité sous forme d'un serveur de répertoires basé sur la norme ITU-T X500. Pour chaque élément, quelques attributs sont stockés dans le répertoire ; certains ne sont pas confidentiels et sont répliqués sur chaque site de sécurité, tandis que d'autres, confidentiels, sont stockés sous la forme d'images générées par un schéma à seuil basé sur l'algorithme de Shamir [SHAM 79]. Le seuil choisi est de *(N+1)/2*, *N* étant le nombre de sites de sécurité : ainsi, on tolère jusqu'à *(N-1)/2* images altérées ou détruites, ce qui correspond à une minorité de sites de sécurité défaillants.

La figure 7 illustre le protocole d'autorisation. Toute requête d'ouverture de fichier, par exemple, doit être validée par une majorité de sites de sécurité. Après l'ouverture, la session d'accès peut être directement réalisée entre le site utilisateur et les sites du serveur. Les tickets sont utilisés pour contrôler chaque opération d'accès (par exemple, lecture ou écriture d'un fragment). Le serveur de sécurité est également responsable de la gestion des droits d'accès.

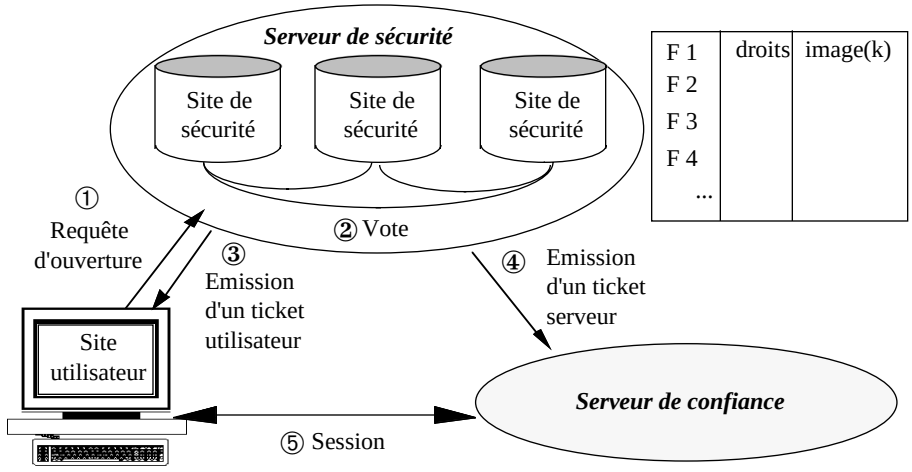

Figure 7. *Le serveur de sécurité (autorisation)*

Le contrôle d'accès est seulement réalisé à l'établissement du lien avec l'objet, c'est-à-dire lors de la connexion entre un site utilisateur et un serveur ou à l'ouverture d'un fichier persistant : une fois la session autorisée, les requêtes sont directement transmises depuis le site utilisateur au serveur (⑤) ; ceci inclut le ticket utilisateur qui est utilisé par le serveur pour vérifier que l'opération a été autorisée. Les tickets serveurs sont utilisés pour vérifier la validité de chacun des accès du site utilisateur au serveur. Ce schéma d'autorisation soulage les serveurs d'applications de la charge du contrôle d'accès et permet une gestion facilitée d'une politique de sécurité uniforme pour l'ensemble du système réparti. Des détails sur les protocoles d'autorisation peuvent être trouvés dans [DES 91a, BLA 92]

4.3. *Comparaison avec d'autres approches*

Kerberos [MIL 87, STE 88, NEU 94] est une autre approche bien connue fournissant des mécanismes de contrôle d'accès dans un système réparti. Kerberos est principalement un serveur d'authentification (il authentifie les utilisateurs, puis donne à l'utilisateur et au serveur sécurisé un ticket pour communiquer de manière sûre). Des mécanismes d'autorisation sont aussi fournis mais simplement pour les connexions client-serveur, mais pas pour des objets de fine granularité.

Kerberos gère un type de fautes beaucoup plus restreint que notre approche. Si Kerberos prend en compte les intrusions externes et celles d'utilisateurs enregistrés, il ne peut faire face à l'abus de privilèges des administrateurs de sécurité. Le contrôle du serveur leur permet d'autoriser et de refuser l'accès au serveur à tout le monde, y compris eux-mêmes. Il n'y a pas de séparation des pouvoirs. En ce qui concerne les fautes accidentelles, Kerberos offre une sauvegarde en ligne basée sur une solution maître-esclave (on suppose l'utilisation de machines à silence sur défaillance). La principale faiblesse de Kerberos est que le serveur d'authentification est un point dur de sécurité : une intrusion réussie sur le site Kerberos entraîne une violation de la sécurité de l'ensemble du système réparti.

5. Les performances de la FRD

En ce qui concerne l'utilisation de la FRD pour le traitement fiable d'informations confidentielles, les performances dépendent de la granularité de la fragmentation. La granularité des objets-fragments obtenus dans l'exemple de l'Agenda Électronique pour résoudre le problème de la confidentialité pourrait apparaître relativement petite (*réunion* composée de *sujet*, *lieu*, *date* et *liste des participants*). Néanmoins, cette technique peut être utilisée pour résoudre des problèmes en utilisant une granularité beaucoup plus grande ; par exemple, considérons *un système de gestion de dossiers médicaux* où l'information est classifiée en deux parties, une partie *administrative* et une autre plus proprement *médicale*. Dans cet exemple assez simple, il n'est pas nécessaire d'aller plus loin dans le processus de fragmentation pour peu que le lien entre ces deux grands fragments (base administrative et base médicale) soit conservé sur le site de confiance. L'accès aux objets dans chaque classe dépend du type d'utilisateur (personnel médical et administratif) et aussi des droits d'accès spécifiques sur des objets particuliers. L'accès aux objets des deux classes est uniquement autorisé à très peu d'utilisateurs, principalement le médecin personnel de chaque patient. Dans cet exemple, les calculs peuvent être lourds, en particulier sur les informations médicales, par exemple l'analyse statistique de dossiers médicaux.

La FRD introduit très peu de surcroît de traitement ou d'information ; elle conduit évidemment à un surcroît de communication par rapport à une simple réplication des traitements, par exemple dans une application qui ne vise pas à tolérer les fautes intentionnelles. Bien que le parallélisme ne soit pas le but de notre technique, les opportunités supplémentaires qu'il fournit peuvent amener des bénéfices significatifs en ce qui concerne les performances des applications dans des circonstances appropriées. Le parallélisme pourrait réduire l'impact du surcroît de communication sur les performances du système. D'autre part, nous considérons que les sites partagés sont beaucoup plus puissants (plusieurs ordres de grandeur) que les stations des utilisateurs, ce qui rend dans tous les cas l'exécution des objets sur ces

sites beaucoup plus performante. Enfin, la mise en œuvre de certains objets (calcul numérique, traitement d'images) peut tirer parti d'architectures spécifiques matérielles et logicielles (environnement de calcul sur des calculateurs massivement parallèles, par exemple).

L'évaluation de la préservation de la confidentialité de l'information traitée par une application FRD a aussi fait l'objet de recherches. Sur certaines des premières techniques de traitement de données fragmentées, des résultats théoriques ont été obtenus en utilisant des techniques d'évaluation quantitative de l'information basées sur la *Théorie de l'Information* [SHAN 48, SHAN 49]. Des calculs d'entropie étendus ont fourni quelques évaluations quantitatives sur l'information qui peut être déduite d'un flot informations d'entrée, mais son intérêt est limité à un petit nombre d'applications [TRO 91c].

Un prototype du serveur de fichiers persistants a été implémenté en utilisant des protocoles de diffusion expérimentaux de Delta-4. Il est intéressant de rappeler ici le coût relatif dû à la FRD par rapport aux techniques classiques comme la simple réplication de fichiers chiffrés et l'approche IDA. La quantité de données stockées et transmises sur le réseau en utilisant les mécanismes de diffusion est identique pour la FRD et la réplication de fichiers chiffrés. L'espace disque occupé est moins important dans l'approche IDA. En ce qui concerne le surcroît de traitements, la FRD est beaucoup moins coûteuse que les deux autres techniques. La fragmentation d'un fichier (incluant le chiffrement, la distribution des octets dans les fragments, le nommage) est plus rapide que le chiffrement d'un fichier avec des techniques classiques (dans notre prototype, la fragmentation est dix fois plus rapide que le chiffrement DES). Cette opération est aussi moins coûteuse que les calculs sur des grandes matrices que nécessite l'IDA surtout si ces calculs doivent être faits sur des stations de travail traditionnelles. Enfin, le temps d'accès à un fichier est aussi amélioré par parallélisation de l'accès aux différents fragments sur les sites de stockage.

Le serveur de sécurité nécessite des protocoles répartis qui sont évidemment plus coûteux que ceux des serveurs de sécurité centralisés ne tolérant pas les intrusions : il est N fois plus coûteux de faire fonctionner N sites de sécurité qu'un seul! Mais ceci est le prix à payer pour tolérer les intrusions des administrateurs du système.

6. Conclusion

La Fragmentation-Redondance-Dissémination fournit une méthodologie générale permettant d'améliorer la fiabilité et la sécurité dans un système réparti, ainsi que des services tolérant les fautes et les intrusions fournis par le système de base tels que le stockage de fichiers persistants et la gestion de la sécurité.

Bien que cela puisse paraître a priori paradoxal, l'idée essentielle consiste à profiter des propriétés de la distribution afin de tolérer les fautes accidentelles et les intrusions. Cette technique a fait l'objet de recherches variées : un cadre général de la fragmentation pour traiter de manière fiable des données confidentielles a été défini. L'approche de conception sous-jacente repose sur une compréhension originale de ce qu'est réellement une information confidentielle, c'est-à-dire qu'elle peut être souvent perçue comme une collection d'éléments non significatifs. La méthodologie actuelle de développement d'applications FRD est basée sur une approche par objet. Le cadre de conception par objet général a été défini et l'exemple de l'Agenda Électronique Réparti a été conçu et implémenté en utilisant cette méthodologie. Des propriétés

avancées des langages de programmation par objets font actuellement l'objet de recherches afin de fournir des outils de développement pour les applications FRD.

Des prototypes du serveur de fichiers persistants et du serveur de sécurité sont aujourd'hui disponibles sur un réseau de stations Sun sous Unix. Des techniques de réplication ont été récemment expérimentées en utilisant le langage réflexif Open C++ [CHI 93]. Au delà des diverses implémentations, ces travaux mettent en lumière les avantages de la FRD et fournissent quelques résultats concernant les techniques de fragmentation, les fonctions de chiffrement et de nommage, les algorithmes de contrôle d'accès et de dissémination.

Les recherches actuelles reposent sur une approche par objet de la FRD avec différents objectifs :

- l'utilisation du concept de réflexivité de certains langages de programmation par objets dans le but de fournir un mécanisme général pour concevoir des objets fiables et sûrs [FAB 95a] ;
- la définition de mécanismes de contrôle d'accès pour des objets de petite taille comme ils sont définis dans la programmation par objets [NIC 94, NIC 95] ;
- l'analyse et l'expérimentation de ces techniques dans des environnements répartis s'appuyant sur la technologie micro-noyau, Chorus [ROZ 90] en l'occurrence.

En conclusion, l'approche FRD est une méthode générale qui permet de tirer partie de la répartition pour mettre en œuvre des applications fiables traitant des données confidentielles, ce qui en fait son originalité.

Remerciements

Ces travaux ont été financés partiellement par le projet ESPRIT Pré-compétitif DELTA-4 n°2252, les projets ESPRIT Recherche de Base PDCS n°3092 et PDCS-2 n°6362, et aussi par une convention DRET n° 88.34.051.00.470.75.01. Les auteurs tiennent à remercier tous ceux qui ont participé à la conception et au développement de la Fragmentation-Redondance-Dissémination, en particulier Jean-Claude Laprie, David Powell, Joni Fraga Da Silva, Jean-Michel Fray, Pierre-Guy Ranéa, Gilles Trouessin et Brian Randell, ainsi que les évaluateurs anonymes pour leurs remarques judicieuses.

8. Bibliographie

[AHI 87] N. AHITUV, Y. LAPID, S. NEUMANN, " Processing Encrypted Data ", *Comm. of the ACM*, vol. 30, n°9, sept. 1987, pp. 777-780.

[BEN 85] K.H. BENNETT, L.F. MARSHALL, B. RANDELL, " Reliable Computing in a UNIX United Environment ", *IEEE Computer Architecture Technical Committee Newsletter*, juin 1985, pp. 23-38.

[BIR 85] K.P. BIRMAN, E. ABBADI, W. DIETRICH, T. JOSEPH, T. RAUECHLE, " An Overview of the ISIS project ", *IEEE Distributed Processing Technical Committee Newsletter*, 1985, pp. 16.

[BLA 90] L. BLAIN, Y. DESWARTE, " An Intrusion-Tolerant Security Server for an Open Distributed System ", *Proc. of the European Symposium in Computer Security (ESORICS'90)*, Toulouse (France), oct. 1990, Pub. AFCET, ISBN 2-9036778-9, pp. 97-104.

[BLA 92] L. BLAIN, " La tolérance aux fautes pour la gestion de la sécurité dans les systèmes répartis ", Thèse de Doctorat, INP de Toulouse, 27 jan. 1992, LAAS-CNRS, 153 p.

[BLA 94] L. BLAIN, Y. DESWARTE, " A smartcard fault-tolerant authentication server ", *1st Smart Card Research and Advanced Application Conference* (CARDIS'94), Lille (France), 24-26 oct. 1994, pp. 149-165.

[CHI 93] S. CHIBA, T. MASUDA, " Designing an Extensible Distributed Language with Meta-Level Architecture ", *Proc. of the ECOOP '93*, Lecture Notes in Computer Science n°707, Springer-Verlag, (O. Nierstrasz Ed.), Kaiserslautern (Allemagne), juil. 1993, pp. 483-502.

[DAU 94] B. d'Ausbourg, " Implementing Secure Dependencies over a Network by Designing a Distributed Security Subsystem ", *Proc. of ESORICS 94*, Brighton (Royaume-Uni), nov. 1994, Lecture Notes in Computer Science n°875, Springer-Verlag, (D. Gollmann Ed.), pp. 249-266.

[DES 91a] Y. DESWARTE, L. BLAIN, J.C. FABRE, " Intrusion Tolerance in Distributed Computing Systems ", *Proc. of the 1991 IEEE Symposium on Research in Security and Privacy*, mai 1991, Oakland (USA), pp.110-121.

[DES 91b] Y. DESWARTE, J.M. PONS, "Performance testing: Report LA3", Rapport LAAS n°91369, Contrat LAAS/CEE-ESPRIT-2 n°2252, Delta-4, nov. 1991, 9 pages.

[FAB 92] J.C. FABRE, B. RANDELL, " An Object-Oriented View of Fragmented Data Processing for Fault and Intrusion Tolerance in Distributed Systems ", *Proc. of ESORICS 92*, Toulouse (France) , nov.1992, Lecture Notes in Computer Science n°648, Springer-Verlag, (Y. Deswarte, G. Eizenberg, J.J. Quisquater, Eds), pp. 193-208.

[FAB 94] J. C. FABRE, Y. DESWARTE, B. RANDELL, " Designing Secure and Reliable Applications using Fragmentation-Redundancy-Scattering: an Object-Oriented Approach ", *Proc. of the First European Dependable Computing Conference (EDCC-1)*, Berlin (Allemagne) , oct. 1994, Lecture Notes in Computer Science n°852, Springer-Verlag, (K. Echtle, D. Hammer, D. Powell, Eds), pp. 23-38.

[FAB 95a] J. C. FABRE, V. NICOMETTE, T. PERENNOU, R. J. STROUD, Z. WU, " Implementing Fault Tolerant Applications using Reflective Object-Oriented Programming ", *Proc. 25th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-25)*, Pasadena, (USA), IEEE Computer Society Press, 1995, pp. 189-208.

[FAB 95b] J.C. FABRE, T. PERENNOU, " Fragmentation of Confidential Objects for Data Processing Security in Distributed Systems ", *Proc. of the 5th IEEE Int. Workshop on Future Trends in Distributed Computing Systems* (FTDCS'95), Cheju Island (Corée), août 1995, pp.395-403.

[LAP 95] J.C. LAPRIE et *al*., *Guide de la Sûreté de Fonctionnement*, Laboratoire d'Ingénierie de la Sûreté de Fonctionnement (LIS), Cépaduès Éditions, 320 p., 1995.

[LIS 85] B.H. LISKOV, " The Argus Language and System ", in *Distributed Systems: Methods and Tools*, Lecture Notes in Computer Science n°190, Springer-Verlag, (M. Paul, H.J. Siegert, Eds.), 1985.

[MAE 87] P. Maes, " Concepts and Experiments in Computational Reflection ", *Proc. of OOPSLA'87*, ACM, 1987, pp. 147-155.

[MIL 87] S.P. MILLER, B.C. NEUMAN, J.I. SCHILLER, J.H. SALTZER, " Kerberos Authentication and Authrisation System ", MIT Project Athena Technical Plan N, Sect. E.2.1, déc. 1987.

[NEU 94] B. C. NEUMAN, T.TS'O, " Kerberos: An Authentication Service for Computer Networks ", *IEEE Communications*, vol.32, n°9, sept. 1994, pp. 33-38.

[NIC 94] V.NICOMETTE, Y.DESWARTE, " An authorization scheme for distributed object systems ", *OOPSLA'94 Conference. Workshop 8: Standards for Security in Object-Oriented Systems*, Portland (USA), oct. 1994, 9 pages.

[NIC 95] V. NICOMETTE, L. BLAIN, Y. DESWARTE, " An Authorization Scheme for Distributed Object Systems ", LAAS Research Report n°95038, fév. 1995.

[POW 91] D. POWELL, *Delta-4: A Generic Architecture for Dependable Distributed Computing*, Research Reports ESPRIT, Project 818/2252, Delta-4, Vol. 1, 484 p., ISBN 3-540-54985-4 et 0-387-54985-4, Springer-Verlag, 1991.

[RAB 89] M.O. RABIN, " Efficient Dispersion of Information for Security, Load Balancing and Fault-Tolerance ", *Journal of ACM*, vol. 36, n°2, avril 1986, pp. 335-348.

[RIV 78] R.L. RIVEST, L. ADELMAN, M.L. DERTOUZOS, " On Data Bank and Privacy Homomorphisms ", *Foundations of Secure Computation*, Academic Press (R.. Demillo, D.Dobkin, A. Jones, R. Lipton eds.), ISBN 0-12-210350-5, pp. 169-179.

[ROZ 90] M. ROZIER, V. ABROSSIMOV, F. ARMAND, I. BOULE, M. GIEN, M. GUILLEMONT, F. HERMANN, C. KAISER, S. LANGLOIS, P. LEONARD, W. NEUHAUSER, " Overview of the Chorus Distributed Operating System ", Chorus Systèmes Technical Report, CS-TR-90-25, 45 pages.

[RUS 83] J.M. Rushby, B. Randell, " A distributed secure system ", *IEEE Computer*, vol.16, n°7 (July 1983), pp. 55-67.

[SHAM 79] A. SHAMIR, " How to Share a Secret ", *Communications of the ACM*, Vol. 22, n°11, nov. 1979, pp. 612-613.

[SHAN 48] C.E. SHANNON, " A Mathematical Theory of Communication ", *The Bell System Technical Journal*, Vol. 31, n°1, juil. 1948, pp. 379-423 & pp. 623-656.

[SHAN 49] C.E. SHANNON, " Communication Theory of Secrecy Systems ", *The Bell System Technical Journal*, Vol. 28, n°4, oct. 1949, pp. 656-715.

[SHR 91] S.K. SHRIVASTAVA, G.N. DIXON, G.D.PARRINGTON, " An Overview of the Arjuna Distributed Programming System ", *IEEE Software*, vol. 8, n°1, 1991, pp. 66-73.

[STE 88] J.G. STEINER, B.C. NEUMAN, J.I. SCHILLER, " Kerberos: An Authentication Service for Open Network Systems ", *Proc. of the USENIX Winter Conference*, Dallas (USA), fév. 1988.

[STR 92] R. STROUD, " Transparency and Reflection in Distributed Systems ", *Proc. of the 5th. ACM SIGOPS European Workshop on Distributed Systems*, Le Mont Saint-Michel (France), sept. 1992, 5 pages.

[TRO 91a] G. TROUESSIN, J.C. FABRE, Y. DESWARTE, " Reliable Processing of Confidential Information ", *Proc. of the 7th Internaltional Conference on Information Security, IFIP/Sec'91*, Brighton (Royaume-Uni), mai 1991, pp. 210-221.

[TRO 91b] G. TROUESSIN, Y. DESWARTE, J.C. FABRE, B. RANDELL, " Improvement of Data Processing Security by means of Fault Tolerance ", *Proc. of the 14th National Computer Security Conference*, NCSC, Washington (USA), oct. 1991, pp. 295-304.

[TRO 91c] G. TROUESSIN, " Quantitative Evaluation of Confidentiality by Entropy Calculation ", *Proc. of the Computer Security Foundation Workshop IV*, Franconia, New Hampshire (USA), juin 1991, pp.12-21.

[WEI 92] C. Weissman, " BLACKER : Security for the DDN, Examples of A1 Security Engineering Trades ", *Proc. of the IEEE Int. Symposium on Research in Security and Privacy*, Oakland (USA), mai 1992, pp.286-292.