FICHE TECHNIQUE DE PROJET
SYSTEMATIC PARIS-REGION
Groupe Thématique Logiciel Libre

APPEL A PROJET : FUI 18
ACRONYME DU PROJET : SecurOCaml
NOM DU PROJET : SecurOCaml

Leader du projet :
- **Entité : OCamlPro**
- **Nom du contact : Fabrice LE FESSANT**
- **Coordonnées :**
  Email : Fabrice.Le_Fessant@OCamlPro.com
  Tel : +33 6 72 73 37 53

Durée et financement du projet :

Date de début du projet	**01/09/2014**
Durée du projet en mois	**36**
Coût du projet en k€	**1988**
Coût de R&D en Ile-de-France en k€	**1988**
Aide envisagée en k€	**849**

Table des matières

1 Résumé **4**

1.1 Modifications par rapport à la dernière soumission 5

2 Objet du projet **6**

2.1 Contexte scientifique autour du langage OCaml 6

2.2 Contexte industriel autour du langage OCaml 6

2.3 Une version du langage OCaml offrant des garanties de sécurité 7

3 Résultats attendus **7**

4 État de l'art **8**

4.1 Recommandations de l'étude LaFoSec 8

4.2 Analyse et audit de code source 9

4.3 Analyse des exceptions non rattrapées 10

4.4 Vérification et chargement de bytecode OCaml 10

4.5 Vérification de la désérialisation 11

4.6 Lien avec d'autres projets de R&D 11

5 Verrous technologiques **12**

5.1 Conception d'attaques logicielles à partir des types avancés d'OCaml (Sous-Projet 2) . . 12

5.2 Algorithmes innovants : gestion de mémoire, et autres composants centraux du système (Sous-Projet 3) 12

5.3 Algorithmes innovants d'analyse statique (Sous-Projet 4) 12

5.4 Conception de nouveaux systèmes de types (Sous-Projet 5) 12

6 Impacts : marchés et retombées **13**

6.1 Marchés et modèles d'affaires 13

6.2 Retombées 14

6.2.1 Retombées Clients 14

6.2.2 Retombées industrielles en termes de souveraineté et d'export 15

7 Organisation en sous-projets **15**

7.1 Récapitulatif 23

8 Partenaires et tableau de financement **25**

8.1 Partenaires PME 25

8.1.1 OCamlPro (porteur du projet, solutions pour OCaml) 25

8.1.2 SafeRiver (développement en sécurité–sûreté) 25

8.1.3 LexiFi (applications bancaires) 26
8.1.4 TrustInSoft (vérifications de cybersécurité) 27
8.2 Partenaires académiques 27
8.2.1 Inria (maintenance et évolution d'OCaml) 27
8.2.2 ENSTA-ParisTech (typage et vérification formelle) 28
8.2.3 CEA LIST (vérification de sûreté de fonctionnement) 28
8.3 Partenaire grand groupe (non financé) 29
8.3.1 Trusted Labs (sécurité des cartes à puces) 29
8.4 Tableau de financement 30

9 Informations complémentaires **30**

1 Résumé

La sécurisation des systèmes d'information et des infrastructures informatiques sensibles est un enjeu stratégique pour notre économie, et pourtant ils restent vulnérables aux attaques toujours plus fréquentes, tant à but d'espionnage industriel que de revente d'informations sensibles. L'ANSSI a recherché les causes profondes de ce problème [AA14] en commandant deux études de trois ans : la première a montré que l'utilisation du langage Java ne pouvait fournir cette sécurisation qu'à un prix exorbitant ; la deuxième, l'étude LaFoSec, a d'une part montré que le langage OCaml fournit des garanties de sécurité que ne peuvent fournir les autres langages de programmation, et d'autre part a émis des recommandations pour mettre à profit ces qualités afin d'offrir un socle de conception d'applications sécurisées aux industriels.

Le projet SecurOCaml mettra en oeuvre ces recommandations pour lever les verrous de recherche en termes de théorie des langages et compilation, et fournir un langage et un environnement dont la fiabilité est prouvée, et fournissant des garanties démontrables en termes de sécurité et de sûreté de fonctionnement.

Le **marché précisément visé par le projet SecurOCaml** est celui des systèmes informatiques hébergeant des données très sensibles : systèmes de paiement en ligne de grands sites commerciaux hébergeant des numéros de cartes bleues, infrastructures réseaux spécialisées telles que des détecteurs d'intrusion ou des extranets, extranets bancaires, etc. Le projet SecurOCaml permettra aux membres du projet à la fois de proposer de tels systèmes, plus fiables, mais aussi de conseiller les acteurs traditionnels de ce marché, pour qu'ils puissent lancer des projets en OCaml, par de la formation, du conseil, du support et le développement de composants génériques réutilisables.

Les **livrables** et leurs **verrous technologiques** sont :
(1) Une version du système OCaml sécurisée. Le principal verrou en est d'ordre algorithmique, l'implantation des recommandations ne devant pas impacter les performances de composants centraux du système, tels que le ramasse-miettes [DG94, DL93] ou la représentation compacte des données [Ler90].

(2) Un environnement d'audit en logiciel libre pour la certification de systèmes d'information en OCaml. Le verrou en est la conception d'algorithmes innovants d'analyse statique tenant compte de la richesse et de la complexité du langage OCaml par rapport à d'autres langages classiques.

(3) Une infrastructure de vérification à l'exécution. Le principal verrou en est de concevoir deux systèmes de types, un pour enrichir le système de types d'OCaml avec des types à runtime, et l'autre pour vérifier le bytecode chargé dans une application OCaml.

(4) La mise à jour des recommandations de l'étude LaFoSec, et la spécification d'un sous-ensemble d'OCaml propre au développement d'applications de sécurité. Le verrou ici est de trouver des failles de sécurité exploitant des objets complexes de la théorie des types (polymorphisme de rangée, types algébriques généralisés, etc.) pour pouvoir s'en prémunir.

Le **consortium** du projet SecurOCaml est composé de 4 PME, 3 partenaires académiques, 1 grand groupe démonstrateur.

Enfin, l'ANSSI a confirmé son intention de suivre les travaux en tant qu'observateur enthousiaste et de participer à un "comité des utilisateurs".

1.1 Modifications par rapport à la dernière soumission

Ce dépôt pour l'appel à projets FUI 18 est la **seconde ressoumission** du projet SecurOCaml. Suite aux retours sur la seconde soumission, les éclaicissements suivants ont été apportés :

- **L'intérêt stratégique du projet** pour LexiFi et TrustInSoft a été éclairci : il s'agit principalement de lever une barrière à l'achat, en permettant à leurs futurs clients de pouvoir auditer le code de leurs solutions ;
- **La pérennité du langage OCaml** est assurée par l'Inria depuis plus de 20 ans et continuera de l'être, en s'appuyant sur le Consortium Caml déjà existant ;
- **La pérennité du travail effectué dans SecurOCaml** sera assurée par, d'une part, une maintenance effectuée par OCamlPro, et des demandes de clients déjà existants (ANSSI, Thalès) ;
- **L'utilisabilité du sous-langage SecurOCaml** est assurée par les résultats de l'étude LaFoSec, qui montrent que la base du langage OCaml (l'équivalent du langage `caml-light` sur lequel OCaml est bâti) est à la fois suffisament expressive pour développer des applications de sécurité, et fournit les garanties recherchées dans ce projet ;
- **L'intérêt des démonstrateurs** (TrustInSoft et TrustedLabs) va être de valider que les résultats du projet SecurOCaml sont immédiatement exploitables sur des applications OCaml existantes, avec les contraintes de leurs environnements, et de tester si certaines failles dévoilées par LaFoSec sont présentes dans leurs applications ;
- **La difficultée du projet en terme de R&D** n'est pas dans la création de la boîte à outils, mais dans la conception des outils eux-mêmes, dont certains demandent une expertise très rare (analyse statique de code, typage) ; en complexité, il s'agit pas moins de développer l'équivalent de logiciels tels que Frama-C (CEA LIST) ou Astrée (École Normale Supérieure) pour OCaml, un travail qui n'est possible que parce que les membres du projets sont des experts de ces domaines ;
- **La gestion de projet** a été remaniée, pour mettre en évidence les livrables qui seront fournis par chaque partenaire par sous-projet, et pour montrer l'analyse des risques, assez faibles dans la mesure où la plupart des sous-projets sont indépendants et découlent des résultats de l'étude LaFoSec.
- **Le marché visé** a été mieux défini : outre le marché actuel de la sûreté, où OCaml est déjà très présent, il s'agit de s'ouvrir celui de la sécurité (estimé à 400 M€en 2017 juste pour les outils d'analyses de code), où certains partenaires (SafeRiver) enregistrent déjà de premières demandes d'audit sur des applications OCaml de sécurité ;
- **En terme de retombées économiques**, outre la suppression d'une barrière à la vente pour plusieurs partenaires (LexiFi, TrustInSoft), SecurOCaml pourrait devenir un **label de sécurité**, les organismes de sécurité (ANSSI en France) pouvant exiger un "niveau de sécurité" équivalent à celui fourni par SecurOCaml avant d'autoriser les déploiement d'une application. La trousse à outils de SecurOCaml sera intégrée à un environnement de développement professionnel pour OCaml vendu par OCamlPro, et des audits de sécurité seront proposés par SafeRiver et TrustedLabs ;
- Réécriture de l'état de l'art et de l'organisation du projet pour **mettre en avant le fait que la recherche effectuée est à la pointe de l'existant** ;
- L'**intérêt national** du projet est double : **économique**, en créant une filière technologique en sécurité sur laquelle la France aurait une **position dominante** et un **avantage compétitif**, et **stratégique**, l'ANSSI soutenant le projet SecurOCaml comme un pas important pour la sécurisation des infrastructures informatique en France (voir leur article [AA14]).
- L'**adéquation temporelle** du projet est idéale : OCaml connaît un intérêt industriel croissant, démontré par l'annonce, en mars 2014, de son utilisation au coeur de **l'infrastructure de Facebook** [VBL+14], et d'un projet d'OCaml bare-metal pour la **sécurité de son Cloud** pour Citrix/XenServer [MMR+13].

2 Objet du projet

L'objet du projet SecurOCaml est d'étendre l'exploitation industrielle, parmi les industriels ayant des exigences de sécurité et sûreté de fonctionnement, du langage OCaml dont les qualités, issues de 30 ans de recherche française, ont été formellement démontrées par une étude commandée par l'ANSSI, plaçant ce langage **en tête des langages de programmation en termes de garanties de sécurité**. OCaml, langage fonctionnel au typage statique fort, est doté de l'un des systèmes de types les plus riches. Il permet d'implanter des algorithmes plus complexes, et les applications écrites dans ce langage sont plus faciles à faire évoluer. Déjà très utilisé dans l'avionique (chez Esterel Technologies par exemple), le secteur bancaire (LexiFi [Ebe09]) et financier (Jane Street [MW08]), ainsi que dans le cloud, avec Citrix [MMR+13], il poursuit sa lente mais sûre **pénétration du marché mondial** avec l'annonce récente en avril 2014 [VBL+14] de **Facebook d'introduire OCaml au cœur de son infrastructure** avec Hack, un PHP typé écrit en OCaml.

Pour qu'OCaml puisse être utilisé pour le développement de systèmes d'information sensibles, l'étude LaFoSec recommande d'une part des améliorations du langage pour accroître encore les garanties de sécurité fournies et, d'autre part, la création de nouveaux outils pour faciliter le travail de certification en sécurité effectué par les CESTI (Centres d'Évaluation de la Sécurité des Technologies de l'Information). Le projet SecurOCaml mettra en œuvre toutes ces recommandations.

2.1 Contexte scientifique autour du langage OCaml

Depuis plus de trente ans, l'un des buts de la recherche académique en informatique est de fournir un langage permettant à la fois d'exprimer concisément des algorithmes complexes tout en effectuant un maximum de vérifications avant l'exécution pour s'assurer du bon fonctionnement du programme. Aujourd'hui, le langage OCaml, développé à l'Inria, est la meilleure concrétisation de cette recherche, profitant à la fois des dernières avancées de la recherche et d'une longue expérience d'utilisation dans le milieu industriel. Il est aussi la cible privilégiée d'outils comme l'assistant de preuve Coq, qui permet de prouver un algorithme puis de générer automatiquement le code OCaml correspondant pour l'inclure dans une application (cas de CompCert, compilateur C utilisé par Airbus et Dassault Aviation [Ler09]). Enfin, OCaml dispose d'une implantation très efficace, avec un compilateur optimisant générant des applications performantes, mais aussi économes en mémoire, donc pouvant traiter des quantités de données plus importantes.

2.2 Contexte industriel autour du langage OCaml

Fort de ces qualités, OCaml est aujourd'hui très utilisé dans l'industrie des systèmes critiques, pour lesquels la performance et la sûreté de fonctionnement sont capitales.

- Ainsi, dans **l'avionique**, des logiciels comme Astrée (ENS Ulm – Absint, [CCF+09]), Scade KCG (Esterel-Technologies, [PAM+09]), Frama-C (CEA LIST – TrustInSoft, [CKK+12]), Taster et Fan-C (ATOS – Airbus, [DDLS10, CDDM12]), Newspeak (EADS, [HL08]), Why3/Alt-Ergo (LRI, [CCKL07]) ou CompCert (Inria, [Ler09]) sont maintenant utilisés pour générer du code embarqué critique ou pour prouver sa correction, c'est-à-dire l'absence de bugs qui pourraient entraîner une défaillance de l'avion.
- En **finance**, l'infrastructure informatique de Jane Street[MW08], entièrement en OCaml, lui permet de négocier automatiquement (sans intervention humaine) un volume de transactions de 10 milliards de dollars par jour, avec une confiance rare dans son milieu.
- Dans le **Cloud computing**, Citrix, l'un des leaders mondiaux avec 15 % du marché avec sa solution XenServer (Amazon EC2 par exemple), utilise OCaml pour développer l'outil de contrôle présent dans chacune des machines virtualisées avec Xen, pièce critique de son architecture.

– Bien sûr, il faut également citer le récent choix d'OCaml par Facebook comme fondation de son infrastructure, au coeur du langage Hack.

2.3 Une version du langage OCaml offrant des garanties de sécurité

Pour que OCaml devienne le **langage de référence en sûreté industrielle et en sécurité logicielle**, des verrous technologiques restent à lever, édictés dans l'étude LaFoSec de l'ANSSI.

Les résultats de cette étude, publiés sur le site de l'ANSSI [1], ont largement confirmé les apports d'OCaml en terme de sécurité, pour développer des applications pour lesquelles la sécurité est critique (serveurs réseaux, algorithmes et protocoles cryptographiques, banque et finance, etc.).

L'étude a aussi fourni :
· un ensemble de recommandations pour les développeurs qui voudraient développer des applications de sécurité en OCaml,
· ainsi qu'un ensemble **d'évolutions à apporter à OCaml**, pour en faire l'outil idéal pour la sécurité.
Ces évolutions portent à la fois :
· sur le langage lui-même,
· sur son implantation (distribuée par l'Inria, avec un support industriel fourni par la société OCamlPro),
· ainsi que sur l'existence d'outils, permettant à la fois d'analyser le code pour un audit de sécurité, et de vérifier à l'exécution que certaines propriétés sont correctes.
Le travail des membres du projet SecurOCaml consistera donc à mettre à jour les recommandations de l'étude LaFoSec, les mettre en œuvre, puis les tester et confirmer les apports de ces évolutions sur des cas d'étude.

L'ANSSI a confirmé son enthousiasme pour le projet SecurOCaml, et son intention de participer aux réunions techniques du projet. Le projet SecurOCaml a été soutenu par les Groupes Thématiques Logiciel Libre (GTLL) et Confiance Numérique et Sécurité (CN&S) du pôle Systematic. Enfin, à l'heure où les langages fonctionnels sont de plus en plus à la mode, le projet SecurOCaml est aussi l'opportunité de soutenir une technologie française, dans un domaine très élitiste mais aussi très critique, face à la concurrence d'autres langages, tels que Haskell (FPComplete), Scala[Ode06] (TypeSafe) et F# (Microsoft).

3 Résultats attendus

Les résultats attendus du projet SecurOCaml sont, pour chaque sous-projet :

Sous-Projet 1	Gestion, suivi du projet, diffusion des résultats et animation de la filière OCaml

1. Une partie des livrables officiels de LaFoSec ont été rendus publics par l'ANSSI le 23 mai 2013 : `http://www.ssi.gouv.fr/fr/anssi/publications/publications-scientifiques/autres-publications/lafosec-securite-et-langages-fonctionnels.html`

Sous-Projet 2	Une **mise à jour des recommandations de l'étude LaFoSec** pour les nouvelles versions de la distribution OCaml (4.00.1 et suivantes). D'une part, les recommandations à destination des développeurs d'applications en OCaml devront évoluer pour couvrir tout le langage, et tenir compte des améliorations apportées par le projet SecurOCaml ; d'autre part, les propositions d'amélioration d'OCaml pouront fournir de nouvelles pistes de développements utiles dans le cadre du projet SecurOCaml pour sécuriser OCaml. La **spécification d'un sous-ensemble d'OCaml propre au développement d'applications de sécurité**, et à l'utilisation des outils développés dans ce projet.

Sous-Projet 3	Une **version sécurisée du système OCaml**. Cette version permettra de développer des applications de sécurité, en fournissant une protection contre les attaques identifiées dans l'étude LaFoSec. En particulier, cette version améliorera le cloisonnement mémoire d'OCaml, pour permettre la manipulation de données sensibles en minimisant les risques d'interception, et effectuera des vérifications supplémentaires lors des interactions avec l'extérieur (chargement de code, de données structurées).

Sous-Projet 4	Un **framework en logiciel libre permettant d'analyser et d'auditer le code OCaml** pour y repérer des problèmes de sécurité ou de sûreté. Le projet SecurOCaml fournira une liste d'outils utiles pour auditer du code OCaml (couverture de code, détection de code mort, détection d'exceptions non rattrapées, etc.), étudiera les outils existant et développera les outils manquant. Cet environnement sera mis à disposition des industriels souhaitant développer des applications OCaml avec un niveau de sécurité accru. Il permettra aussi, dans le cadre de normes de certification, de garantir certaines propriétés pour la sûreté de fonctionnement.

Sous-Projet 5	Un **ensemble de bibliothèques permettant de vérifier à l'exécution certaines propriétés de sécurité**. En particulier, le projet SecurOCaml étudiera la possibilité de vérifier le bytecode OCaml pour les applications effectuant du chargement dynamique de code, ainsi que la désérialisation de données, en vérifiant certaines propriétés de conformité sur les données reçues.

Sous-Projet 6	Un **rapport d'expérimentation de ces outils sur du code existant**, effectuées par les démonstrateurs de ce projet. L'environnement développé dans le cadre du projet sera testé sur deux applications industrielles réelles : – le système Frama-C, pour la vérification de code C, développé par le CEA LIST, déjà déployé dans l'avionique (Airbus), et utilisé par TrustInSoft pour certifier des composants de cyber-sécurité. – le système de formalisation des contrats financiers développé par la société LexiFi, qui doit être régulièrement audité pour être intégré dans des infrastructures bancaires sécurisées.

4 État de l'art

4.1 Recommandations de l'étude LaFoSec

Les rapports de l'étude LaFoSec se divisent en deux types de recommandations.

Les premières recommandations sont à destination des développeurs programmant en OCaml, et sont disponibles sur le site officiel de l'ANSSI. Elles permettent aux développeurs d'éviter les pièges du langage en terme de failles de sécurité. Ces recommandations portaient sur la version 3.12.1 de l'implémentation, depuis laquelle plusieurs ajouts importants ont été faits au langage, dont il serait important de connaître l'impact au niveau de la sécurité.

Les secondes recommandations sont à destination des développeurs du langage OCaml. Non disponibles sur le site de l'ANSSI, elles ont été présentées lors du séminaire JFLA'2013 (`http://jfla.inria.fr/2013/JFLA-partie3.pdf`), et l'ANSSI s'est engagée à fournir les documents officiels aux membres du consortium au démarrage du projet.

La présentation aux JFLA'2013 fournit une liste d'évolutions à apporter à OCaml, que nous avons résumée ici :

1. Ajout d'avertissements dans le compilateur
   - Sur les annotations de type
   - Sur le masquage d'identificateurs
2. Modifications de la machine virtuelle
   - Effacement des données sensibles répliquées
   - Meilleur contrôle de l'environnement
3. Modifications des bibliothèques standards
   - Maintient de blocs en mémoire
   - Limitation en temps de certaines fonctions
   - Fonctions sécurisées sur les chaînes de caractères
   - Fonctions de log/debug
4. Ajout de vérifications (compilateur, ou outils séparés)
   - Analyse des exceptions levées par chaque fonction
   - Analyse des données transportées par les exceptions
   - vérification des débordements d'entiers
   - sérialisation/désérialisation typées et vérifiées
5. Développements
   - Suppression des initialisations/codes inutilisés dans le runtime et le code utilisateur
   - Annotation explicite de la récursion terminale
   - Documentation du préprocesseur Camlp4,
   - Vérificateur de byte-code OCaml
6. Évolutions plus complexes
   - Chaînes de caractères non mutables
   - Renforcement de l'encapsulation par suppression des comparaisons polymorphes
   - Définition d'un noyau du langage pour la sécurité
   - Bytecode typé et vérification du code chargé dynamiquement
   - Cloisonnement de la mémoire
   - Contrôle du code d'initialisation des modules liés
   - Typage avec unités de mesure
   - Outils de test, debug, profilage

4.2 Analyse et audit de code source

Le compilateur de la distribution OCaml effectue plusieurs analyses complexes pour détecter des bugs éventuels : en plus de vérifier les types, dont la richesse permet d'exprimer simplement des

contraintes de correction sur les structures de données [Gar01, Gar04], le compilateur signale par exemple la non-exhaustivité des filtrages [Mar08, LFM01, Mar03] et la déclaration d'identifiants qui ne sont pas utilisés par la suite, qui indiquent souvent des erreurs dans le code. Néanmoins, il existe encore plusieurs motifs de bugs fréquents qui ne sont pas signalés par le compilateur ("open" masquant des valeurs, etc.), et mériteraient d'être détectés.

De plus, la distribution OCaml ne fournit pas d'outils officiels pour faciliter l'audit de code. Il existe différents outils qui peuvent être utiles pour un tel audit, mais ces outils ne sont pas officiellement maintenus ni synchronisés avec la dernière version d'OCaml. Par exemple, pour la couverture de code, il existe plusieurs projets, tels que "Bisect", "mlcov"[PCC+07] et "zamcov"[WJC11], mais aucun n'est clairement maintenu et utilisable dans un contexte industriel.

4.3 Analyse des exceptions non rattrapées

Le système de types d'OCaml est l'un des plus évolués [Ler93, Ler94] parmi ceux des langages dits typés statiquement, il permet à la fois de détecter précocément de très nombreux bugs (variables non définies, fonctions ou variables mal utilisées, etc.), mais aussi d'aider le programmeur à exprimer simplement, par la richesse de ses types, toute la complexité de certaines structures de données (arbres, graphes, etc.). Néanmoins, contrairement au système de types d'Haskell, celui d'OCaml ne fournit aucune information sur les exceptions qui peuvent être levées par certaines fonctions, et capturées par d'autres[Yi94, YR02]. L'un des problèmes récurrents des programmes OCaml est donc la possibilité d'être interrompus par une exception dont la levée n'avait pas été anticipée, ou pas au bon endroit[TSV09].

En 1999, François Pessaux (aujourd'hui, enseignant-chercheur à l'ENSTA-ParisTech, partenaire du projet) avait effectué sa thèse de doctorat ("Détection statique d'exceptions non rattrapées en OCaml", voir aussi [PL99]) sur la conception d'un outil pour retrouver les exceptions levées et non rattrapées dans un programme OCaml. Le prototype développé pendant cette thèse, `ocamlexc`, était complexe, n'a pas été maintenu, et n'a pas suivi les évolutions du système de types, bien plus complexe aujourd'hui qu'à l'époque. Cette complexité rend la simple mise à jour de l'outil impossible, et impose le développement d'un nouvel outil, utilisant une approche innovante facilitant la synchronisation avec les futures évolutions du langage OCaml.

4.4 Vérification et chargement de bytecode OCaml

Un motif fréquent dans la conception des logiciels modernes est la spécialisation de l'application par le chargement dynamique de code, souvent du bytecode compilé à la volée. OCaml offre la possibilité de charger dynamiquement du code compilé du même type que l'application en cours d'exécution (bytecode dans une application bytecode, code natif dans une application native).

Cependant, dans le contexte d'une application de sécurité, il serait important de pouvoir vérifier le code avant de le charger dynamiquement dans l'application, pour vérifier que le code est conforme aux interfaces qu'offre l'application, et ne peut pas provoquer de défaillance, ou accéder à des données sensibles. Une telle vérification est faite, par exemple, par la JVM (Java Virtual Machine) et par la plateforme `.Net` (Microsoft). Dans le cas d'OCaml, il faudrait permettre de charger du bytecode OCaml dans une application native, après l'avoir vérifié, puis le compiler à la volée pour obtenir les performances attendues.

Des travaux sur la vérification de bytecode OCaml ont déjà été effectués ([LR98] et "typage du bytecode Caml", Sébastien Ailleret, JFLA'2002), mais ce travail était partiel, ne spécifiait pas réellement les garanties apportées par l'approche, et ne supporte pas toutes les extensions récentes d'OCaml. Une approche plus puissante, mais aussi plus complexe, serait basée sur les travaux de Greg Morrisset

("Typed Assembly Language" [TMC+96]). Pour la compilation à la volée de bytecode dans une application native, le travail le plus proche est celui portant sur `ocamljit2`, un compilateur à la volée ciblant le runtime bytecode d'OCaml, qu'il faudrait modifier pour cibler le runtime natif, légèrement différent.

4.5 Vérification de la désérialisation

OCaml permet, pour toute valeur calculable dans un programme, de la décrire (sérialisation) de façon compacte dans une chaîne de caractères (qui peut ensuite être sauvée dans un fichier, ou transmise sur le réseau), puis à l'inverse, de générer une nouvelle valeur à partir d'une telle description dans une chaîne de caractères.

Ces opérations ne sont pas sûres du point de vue du typage, car OCaml ne vérifie pas que la description correspond au type de la valeur attendue, ce qui peut provoquer, dans le meilleur des cas, une erreur à l'exécution, et dans le pire cas, un résultat faux non détecté.

La vérification de la sérialisation a été abordée plusieurs fois[LPSW03, KRJ09], et en particulier dans la thèse de Grégoire Henry (aujourd'hui, ingénieur chez OCamlPro, membre du projet). Son approche nécessitait d'étendre le langage OCaml avec des descriptions des types à l'exécution, qui n'existent pas aujourd'hui dans le langage. Ces descriptions de types sont utilisées pour vérifier que la structure de la valeur générée est bien compatible avec la description du type attendu. Un problème supplémentaire serait de vérifier aussi des invariants sémantiques sur la valeur (par exemple sous la forme de contrats) : par exemple, si un couple (tableau, longueur du tableau) est sérialisé, il faudrait vérifier à la désérialisation que non seulement les données sont bien compatibles avec les types attendus, mais aussi que la longueur reçue est bien celle du tableau.

4.6 Lien avec d'autres projets de R&D

Le projet SecurOCaml est en lien avec d'autres projets de R&D :

Projet FUI Safe-Python (2012-2015, 3.3 M€) : ce projet vise à fournir un outil de vérification de code Python pour des applications de sécurité. Les garanties qui seront fournies par ce projet sont très limitées (absence d'erreurs simples à runtime), et ne permettent pas de détecter des erreurs algorithmiques, qu'OCaml détecte par défaut. La cause en est l'impossibilité en Python de définir des types plus complexes que les types de base. La certification en sécurité apportée par SecurOCaml dépassera donc nettement celle apportée par Safe-Python. En outre, le projet Safe-Python bénéficiera lui-même des avancées du projet SecurOCaml, une partie des outils qui y sont développés étant basés sur Frama-C, cas d'usage de SecurOCaml.

Projet FUI CEEC (2012-2015, 3.3 M€) : le projet CEEC (Certification d'Environnements de Confiance) vise à fournir un environnement de développement de composants sûrs et sécurisés. Cet environnement dépend fortement d'OCaml, avec l'utilisation de l'assistant de preuve "Coq" pour générer le code du compilateur Inria "CompCert", et le compilateur "KCG" de la suite "Scade 6" d'Esterel Technologies[PAM+09]. L'ensemble de ces outils bénéficieront de la possibilité d'être audités par les outils développés dans le cadre du projet SecurOCaml.

Projet FUI Hilite (2010-2013, 3.9 M€) : ce projet visait à fournir des outils de vérification de code Ada pour des applications en sûreté de fonctionnement, et aboutira en 2014 au produit "Spark 2014" d'AdaCore. La plupart des outils utilisés ("why3", "jessie", "alt-ergo") ont été développés en OCaml, et doivent être audités dans le cadre de la certification avionique (DO-178B/C). Les outils développés dans SecurOCaml pourront aussi être utilisés dans le cadre de cette certification pour apporter des garanties supplémentaires de qualité du code.

5 Verrous technologiques

Contrairement à la plupart des langages de programmation, OCaml est un langage issu de la recherche, sa richesse et sa cohérence mathématique en font un **langage difficile à modifier sans une connaissance importante en méthodes formelles**, et son expressivité rend beaucoup des analyses statiques traditionnelles inadaptées à ses constructions. Cette difficulté sera compensée par l'expertise inhabituelle des membres du consortium : les ingénieurs des PMEs qui travailleront sur le projet sont presque tous issus de laboratoires académiques, et combinent une grande expertise des méthodes formelles et une bonne expérience en programmation, et seront associés aux chercheurs qui ont conçu le langage.

Outre cette difficulté intrinsèque à OCaml, les principaux verrous technologiques sont les suivants :

5.1 Conception d'attaques logicielles à partir des types avancés d'OCaml (Sous-Projet 2)

Pour la mise à jour des recommandations de l'étude LaFoSec, le principal verrou est de concevoir des attaques logicielles à partir de tous les objets de la théorie des types d'OCaml (polymorphisme de rangée, types algébriques généralisés, etc.), comme cela avait été débuté dans le cadre de l'étude LaFoSec [ASIN13]. Cela suppose de maîtriser la représentation de ces objets parfois très complexes, pour en exploiter l'usage afin de mettre en évidence des failles de sécurité exploitables par des attaquants ou par un développeur mal-intentionné.

5.2 Algorithmes innovants : gestion de mémoire, et autres composants centraux du système (Sous-Projet 3)

Pour la version du système OCaml sécurisé, le principal verrou est d'ordre algorithmique : une grande partie des recommandations touchent des composants centraux du système, tel que le gestionnaire de mémoire, et une implantation naïve pourrait impacter sévèrement les performances des applications. Il s'agira donc de développer des algorithmes innovants intégrant parfaitement les nouveaux mécanismes de sécurité dans le système existant.

5.3 Algorithmes innovants d'analyse statique (Sous-Projet 4)

Pour l'environnement d'audit pour la certification de systèmes d'information, le principal verrou est de concevoir des algorithmes innovants d'analyse statique tenant compte de la richesse et de la complexité du langage OCaml par rapport à d'autres langages plus classiques. Comme pour l'analyse de valeur dans Frama-C [CKK+12] ou dans Astrée [CCF+07], il s'agit de trouver les bons domaines d'interprétation abstraite pour, dans un temps de calcul raisonnable, obtenir des résultats suffisament précis pour être exploitables.

5.4 Conception de nouveaux systèmes de types (Sous-Projet 5)

Pour l'infrastructure de vérification à l'exécution, le principal verrou est la conception de deux nouveaux systèmes de types, un pour enrichir le système de types actuel d'OCaml avec des types à runtime, et l'autre pour typer le bytecode chargé dans une application. Ce travail est donc à la fois théorique (conception du système de types et vérification formelle de sa correction) et pratique (implantation du système de types dans le compilateur et dans une bibliothèque).

6 Impacts : marchés et retombées

6.1 Marchés et modèles d'affaires

Les marchés visés par le projet SecurOCaml sont :

- celui des systèmes d'information hébergeant des données très sensibles, d'abord au niveau national, puis au niveau international : systèmes de paiement en ligne de grands sites commerciaux hébergeant des numéros de cartes bleues, infrastructures réseaux spécialisées (détecteurs d'intrusion, extranets, etc.), sites web de banques, gros acteurs de réseaux sociaux (Facebook, LinkedIn, etc.). etc.
- celui des systèmes critiques et de sûreté industrielle soumis à des exigences de certification, et les sociétés qui en produisent les garanties et audits associés.
- à plus long terme, celui de toutes les infrastructures stables conçues dans d'autres langages occasionnant des coûts de maintenance élevés et chroniques (par exemple, on peut s'attendre à une migration d'applications écrites dans les langages Java et PHP vers OCaml).

Le projet SecurOCaml aboutira in fine à un **label de confiance SecurOCaml**, pouvant alimenter différents modèles d'affaires.

- Pour **OCamlPro**, il s'agit de conquérir le marché de la sûreté industrielle avec son expertise OCaml. Une fois le projet SecurOCaml terminé, les garanties seront telles qu'il sera possible de démarcher les Directions informatiques de secteurs où l'informatique est une composante critique opérationnelle. Il faudra en parallèle assurer formations et conseil pour une mise à niveau technique de ces mêmes clients. L'idée est également de maintenir sur le territoire français des compétences pointues et rares - qui auraient tendance à être débauchées vers la Silicon Valley.
- Pour **Lexifi**, il s'agit de lever un verrou commercial mais aussi de consolider la base de fonctionnement interne.
- Pour **SafeRiver :** SafeRiver a été le partenaire chef de file de l'étude LaFoSec et développe en OCaml des composants dédiés à la sécurité, soumis à des évaluations. L'accès à un environnement de développement et de production de code offrant des garanties de confiance est un avantage compétitif important. SafeRiver compte donc élargir cette offre. Par ailleurs, SafeRIver développe également en OCaml ses outils propriétaires dans le domaine de l'analyse de code et de la preuve de modèle. Ces outils doivent être qualifiés au sens des nouvelles normes (DO178C, ISO26262, EN50128 par exemple.) Le projet facilitera la démonstration de ce niveau d'assurance pour nos plates-formes de vérification.
- Pour **TrustInSoft**, il s'agit d'améliorer son offre commerciale en augmentant les garanties des outils d'analyse sur lesquels s'appuient ses composants certifiés.

Les clients visés sont donc :

1. Les **organismes devant traiter des données très sensibles** (organismes d'état, organismes bancaires, infrastructures réseau sensibles, etc.) : pour ces organismes, les membres du consortium SecurOCaml pourront développer des logiciels de traitement spécialisés en OCaml, offrant des garanties supérieures aux offres actuelles en terme aussi bien de sécurité que de sûreté. Le projet SecurOCaml fournira les outils qui permettront aux organismes de certification (CESTI/ANSSI)de vérifier la qualité de ces logiciels et leur adéquation avec les contraintes d'utilisation (critères communs, EALs).
2. Les **fournisseurs traditionnels de systèmes d'information** : pour fournir des systèmes d'information plus sûrs, les acteurs traditionnels de la sécurité devront exploiter la technologie OCaml. Pour cela, ils pourront utiliser l'outillage (dont OCamlPro fournira le support technique et la maintenance) développé dans SecurOCaml, ou des composants existants (certifiés par TrustInSoft via les outils du projet SecurOCaml). Ils feront aussi appel aux services des membres du projet, que ce soit pour de la formation ou pour du conseil (développement à façon en OCaml par OCamlPro, ou développement sécurisé en OCaml par SafeRiver).

3. Les **éditeurs et utilisateurs de logiciels soumis à de la certification** dans le domaine de la sûreté de fonctionnement. Ces entreprises sont parfois déjà utilisatrices d'OCaml, en particulier dans le domaine de l'avionique. Soumises à des normes de certifications draconiennes (DO-178 B/C), les outils développés dans le cadre du projet SecurOCaml leur permettront de diminuer notablement leurs coûts de certification. Elles sont principalement clientes de support critique sur les logiciels qu'elles utilisent, avec des contrats de support à très long terme (plusieurs décennies, pour du logiciel embarqué sur un avion).

6.2 Retombées

6.2.1 Retombées Clients

Les prévisions chiffrées de retombées économiques par partenaires sont résumées dans le tableau suivant :

Entreprise	Chiffre d'affaire généré (k€)				Emplois créés
	2018	2019	2020	Total	en IDF
OCamlPro					
Développement	500	600	800	1 900	8
Conseil/audit	100	200	300	600	2
Support	100	200	300	600	2
Total	700	1 000	1 400	**3 100**	**12**
LexiFi					
Licenses	300	100	100	**500**	**1**
SafeRiver					
Étude et développement	100	200	300	**600**	**2**
TrustInSoft					
Licenses	200	300	500	1 000	1
Support	200	300	500	1 000	3
Développement	150	150	200	500	1
Total	550	750	1 200	**2 500**	**5**
Total	1 650	2 050	3 000	**6 700**	**20**

Pour **OCamlPro**, le CA généré par SecurOCaml sera lié (1) au lancement de plusieurs projets de développement de systèmes sécurisés en OCaml. En particulier, OCamlPro souhaite travailler sur des solutions de **sécurité pour le Cloud**, basées sur l'utilisation de la technologie Mirage[MHD+07], permettant de **virtualiser des applications complètes sans leur système d'exploitation**, et fournissant par conséquent des performances réseaux supérieures avec une sécurité accrue. Les résultats du projet SecurOCaml sont indispensables au lancement de tels projets, et devraient permettre de lancer de ces 2 à 3 projets (1 en France, 2 à l'international). D'autre part, le CA généré par SecurOCaml proviendra d'activités de **conseil/audit** sur des applications en OCaml, pour lesquels les outils développés dans SecurOCaml permettront une meilleure productivité, ainsi que du support et de la maintenance technique qui sera facturée aux nouveaux clients.

Pour **LexiFi**, le CA généré directement par SecurOCaml sera lié à une accélération des négociations commerciales, en particulier dans la cible "redistributeur technologique" pour laquelle un argumentaire relatif à la sécurité du langage d'implémentation permettrait de lever certaines réticences à l'achat. Cette estimation en terme de CA ne prend pas en compte des effets indirects, non quantifiables, mais que LexiFi espère significatifs : d'une part, l'amélioration des propriétés de sécurité autour du langage OCaml facilitera pour LexiFi la diffusion de ses technologique sous forme de service (en mode SaaS) ; d'autre part, les évolutions envisagées pour le langage OCaml et ses outils améliore-

ront la productivité des équipes de développement de LexiFi.

Pour **SafeRiver**, le CA généré par SecurOCaml sera obtenu par de nouveaux contrats d'**étude et développement** de systèmes de sécurité en OCaml, rendus possibles par la capacité de valider ces systèmes grâce aux outils de SecurOCaml.

Pour **TrustInSoft**, le CA généré par SecurOCaml sera lié à la fois aux **licenses et au support** sur les composants de cyber-sécurité qu'ils auront pu développer et certifier grâce à Frama-C, lui-même certifié grâce aux outils du projet SecurOCaml, fournissant ainsi une chaîne complète de certification sécuritaire. Enfin, une part du CA viendra aussi de la demande de développement de nouveaux composants de cyber-sécurité, offrant les mêmes garanties.

6.2.2 Retombées industrielles en termes de souveraineté et d'export

Un autre objectif du projet SecurOCaml est aussi structurant : le langage OCaml est aujourd'hui un atout pour l'industrie informatique française, qui lui permettrait, s'il était plus utilisé, de fournir des logiciels plus fiables (moins de bugs), plus puissants (plus de fonctionnalités), plus rapidement (time-to-market court) avec un coût de maintenance plus faible (meilleure marge).

Le projet SecurOCaml vise donc aussi à créer et structurer une **nouvelle filière basée sur une technologie française**, autour d'acteurs experts de cette technologie (OCamlPro d'abord, mais aussi LexiFi et TrustInSoft). Une attention particulière sera portée, pendant le projet, à comprendre comment organiser cette nouvelle filière, et communiquer autour de l'utilisation d'OCaml dans des projets industriels, tout autant auprès des acteurs industriels qu'académiques.

Finalement, l'ensemble des outils du projet SecurOCaml étant développés en logiciel libre, ils amélioreront aussi la **visibilité et l'utilisabilité d'OCaml** face à ses concurrents, langages fonctionnels, dans un domaine où la compétition est devenue très forte, avec l'arrivée de**F#** (version très simplifiée d'OCaml intégrée à Visual Studio, chez Microsoft) et de **Scala** (couche fonctionnelle au dessus Java, chez TypeSafe).

7 Organisation en sous-projets

Le projet SecurOCaml est décomposé en 6 sous-projets, dont le premier traite des aspects administratifs (gestion de projet et communication) et le dernier des cas d'étude. Un récapitulatif des livrables et des efforts par sous-projet, ainsi qu'un GANTT sont fournis en section 7.1.

Sous-projet 1 : Gestion de projet et Diffusion

Le premier sous-projet traite des aspects administratifs du projet et de la communication du projet SecurOCaml.

Tâche 1.1 : Gestion de projet

Durée	T0+0 à T0+36
Responsable	OCamlPro
Participants	OCamlPro (6 HM) TrustedLabs (1 HM) TrustInSoft (1 HM) SafeRiver (1 HM) LexiFi (1 HM) Inria (1 HM) ENSTA (1 HM) CEA (1 HM)

Cette tâche regroupe toutes les activités liées à la gestion du projet SecurOCaml :

– Mise en place du Comité de Pilotage du projet (le comité de pilotage se réunira nominalement tous les semestres). Des réunions du comité de pilotage pourront être suscitées également selon les besoins.
– Coordination des aspects administratifs et financiers du projet
– Intégration, gestion des revues et éléments contractuels du projet
– Gestion de projet / suivi des activités des différents lots
– Interface avec les différents responsables pour chaque sous-projet.
– Vérification de l'adéquation des livrables
– Mise en place des outils nécessaires pour l'établissement des liens et échanges entre les lots (documents, réunions communes, etc.)

Numéro	Échéance	Intitulé livrable	Type	Resp.
1.1.1	T0+6	Kick-off et accord de consortium	Document	OCamlPro
1.1.2	T0+12	Rapport annuel d'activité	Document	OCamlPro
1.1.3	T0+24	Rapport annuel d'activité	Document	OCamlPro
1.1.4	T0+36	Rapport annuel d'activité	Document	OCamlPro

Tâche 1.2 : Diffusion et valorisation

Durée	T0+0 à T0+36
Responsable	OCamlPro
Participants	OCamlPro (4 HM) Inria (2 HM)

Cette tâche se concentrera sur la diffusion et la valorisation des résultats du projet SecurOCaml afin d'accélérer les retours sur investissement attendus par les différents partenaires.

La valorisation des résultats du projet SecurOCaml va se dérouler à différents niveaux de communication :

Les travaux seront tout d'abord présentés à mi-projet et en fin de projet au travers de différents événements, les Journées SSTIC organisées chaque année à l'IRISA, qui réunissent les experts français en sécurité informatique, et les Journées JFLA, qui réunissent les experts français des langages fonctionnels.

Ces événements permettront notamment de présenter les travaux en cours, et d'intéresser des clients potentiels pour les outils développés, ou les services que les partenaires pourront proposer suite à ce projet.

Enfin, l'intégration de ces résultats au sein des offres de OCamlPro sera également effectuée. Ceci passera notamment par la réalisation de pages web afin d'assurer la bonne diffusion des outils développés permettant ainsi leur valorisation et diffusion sous licence libre. Un travail important sera fait pour remonter systématiquement les évolutions dans la version officielle de la distribution OCaml. Si des évolutions ne sont pas acceptées, une version spécifique d'OCaml avec ces extensions sera diffusée et maintenue par OCamlPro, sous le nom SecurOCaml.

Numéro	Échéance	Intitulé livrable	Type	Resp.
1.2.1	T0+6	Mise en place des sites web et forges	Logiciel	OCamlPro

Sous-projet 2 : Recommandations et Sous-Langage

Ce sous-projet a pour but de mettre à jour les recommandations de l'étude LaFoSec en tenant compte des modifications récentes d'OCaml, et d'utiliser ces recommandations pour définir un sous-langage

excluant les composants sur lesquels des garanties de sécurité seraient trop difficiles à fournir.

Tâche 2.1 : Mise à jour des recommandations

Durée	T0+0 à T0+30
Responsable	SafeRiver
Participants	SafeRiver (14 HM) Inria (5 HM)

L'étude LaFoSec portait sur la version 3.12.1 d'OCaml. La version actuelle est 4.01, et présente plusieurs fonctionnalités nouvelles importantes par rapport à la version 3.12.1 : ajout de modules de première classe, types algébriques généralisés, substitutions en place, etc. De plus, tous les aspects du langage n'avaient pas été complètement traités dans l'étude. L'objectif de cette tâche est donc de mettre à jour les recommandations de LaFoSec (recommandations pour l'utilisation du langage, mais aussi recommandations pour l'évolution du langage) pour porter sur l'ensemble du langage OCaml, tel que fourni par la version officielle au démarrage du projet SecurOCaml.

Numéro	Échéance	Intitulé livrable	Type	Resp.
2.1.1	T0+12	Mise à jour des recommandations pour 4.00	Document	SafeRiver
2.1.2	T0+30	Mise à jour des recommandations pour 4.02	Document	SafeRiver

Tâche 2.2 : Définition d'un sous-langage sécurisé

Durée	T0+0 à T0+36
Responsable	OCamlPro
Participants	OCamlPro (4 HM) LexiFi (2 HM) TrustInSoft (1 HM) SafeRiver (1 HM) Inria (1 HM) ENSTA (1 HM) CEA (1 HM)

Cette tâche travaillera sur une définition précise d'un sous-ensemble du langage OCaml, permettant d'offrir des garanties de sécurité. Le périmètre de ce sous-langage sera spécifié en partant, d'une part, des recommandations de LaFoSec afin d'éviter des traits, des outils ou des bibliothèques d'OCaml jugées risqués en terme de sécurité, et d'autre part, des cas d'étude du projet, jugés représentatifs des besoins actuels des industriels pour développer des applications en OCaml. Enfin, l'incapacité des analyses effectuées par les outils d'audit à traiter certains aspects du langage pourra amener à exclure ces aspects du périmètre du sous-langage recommandé. L'étude LaFoSec a montré qu'un tel sous-langage existe, et peut être exploité industriellement.

Numéro	Échéance	Intitulé livrable	Type	Resp.
2.2.1	T0+18	Définition du sous-langage SecurOCaml v1	Document	OCamlPro
2.2.2	T0+36	Définition du sous-langage SecurOCaml v2	Document	OCamlPro

Sous-projet 3 : Évolution du langage

Ce sous-projet a pour but de fournir une nouvelle version du langage OCaml, spécialisée pour la mise en œuvre d'applications de sécurité. Ce travail impliquera à la fois des évolutions du compilateur, du runtime, des bibliothèques (distribution) et de certains outils (préprocesseur).

Tâche 3.1 : Évolution de la distribution OCaml

Durée	T0+0 à T0+36
Responsable	Inria
Participants	Inria (12 HM) OCamlPro (5 HM) LexiFi (3 HM)

Cette tâche propose un certain nombre d'évolutions de la distribution OCaml, portant sur le compilateur, le runtime et les bibliothèques distribuées.

Pour le **compilateur**, l'étude LaFoSec recommande les améliorations suivantes :
- (L1) Import-Export de noms : le compilateur devra fournir la possibilité d'indiquer précisément dans le langage les identifiants qui sont exportées par un module, ainsi que les identifiants qui sont importés par un module. Pour cela, le langage sera modifié, ainsi que le parseur et le typeur dans le compilateur. Le compilateur devra aussi fournir une indication si des identifiants précédemment importés sont redéfinis par des imports suivant ;
- (L2) Mutabilité restreinte : le compilateur fournira une nouvelle construction `let mutable x =` permettant de limiter la portée de valeurs mutables d'une fonction.
- (L3) Indication de récursion terminale : le compilateur fournira pour les fonctions récursive une information précise sur la compilation de la récursion terminale, pour éviter les débordements de pile sur des données trop larges ;
- (L4) Retrait de l'option `-unsafe` qui permet de supprimer certaines vérifications de débordement.
- (L5) Débordements d'entiers : le compilateur devra générer des vérifications pour détecter les débordements d'entier lors de l'exécution. Pour cela, plusieurs passes d'optimisation du compilateur devront être modifiées, ainsi que la génération finale de code assembleur ;

Pour le **runtime**, l'étude LaFoSec recommande les améliorations suivantes :
- (R1) Forme d'effacement des données : le GC sera modifié pour permettre l'effacement automatique de certaines données sensibles (clés cryptographiques) lors de leurs copies et de leurs finalisations.
- (R2) Cloisonnement mémoire : la gestion mémoire d'OCaml sera modifiée pour permettre de protéger certaines zones de mémoire par des protections matérielles, pour éviter les accès sans permission.
- (R3) Renforcement de l'encapsulation : le runtime effectuera des tests supplémentaires pour vérifier que des données protégées ne sont pas accédées par des fonctions polymorphes, telles que l'égalité, la comparaison, la sérialisation et le hachage, qui pourraient permettre de générer des fuites d'informations.

Pour les **bibliothèques**, l'étude LaFoSec recommande les améliorations suivantes :
- (D1) Mutabilité restreinte : La nouvelle bibliothèque fournira des chaînes non modifiables, ainsi que des tableaux d'octects, permettant d'éviter complètement l'usage des `strings` actuelles d'OCaml, dont la mutabilité pose de nombreux problèmes pour les applications de sécurité.
- (D2) Ajout de modules de débogue : des modules de déboguage seront développés, permettant d'afficher plus d'information en mode deboguage, sans coût en performance en mode de production.
- (D3) Séparation langage-système : la distribution sera réorganisée pour séparer les modules purement fonctionnels des modules faisant des accès sensibles en terme de sécurité, permettant ainsi de détecter plus facilement les zones sensibles d'une application.

Ces modifications seront implantées, et soumises pour inclusion dans les nouvelles releases d'OCaml, avec le ciblage suivant :
- OCaml 4.03 : (L1), (L2), (R1), (D1)
- OCaml 4.04 : (L3), (L4), (R2), (D2)
- OCaml 4.05 : (L5), (R3), (D3)

Numéro	Échéance	Intitulé livrable	Type	Resp.
3.1.1	T0+12	Release d'OCaml 4.03	Logiciel	Inria
3.1.2	T0+24	Release d'OCaml 4.04	Logiciel	Inria
3.1.3	T0+36	Release d'OCaml 4.05	Logiciel	Inria

Tâche 3.2 : Évolution des outils

Durée	T0+0 à T0+24
Responsable	LexiFi
Participants	OCamlPro (8 HM) LexiFi (4 HM)

Dans cette tâche, les partenaires travailleront sur l'amélioration de plusieurs outils qui font partie de l'usine logicielle habituelle de développement en OCaml, et dont il est nécessaire de vérifier les propriétés de sécurité :

– L'étude LaFoSec avait montré l'importance de clarifier l'étape de préprocessing du code OCaml, souvent présente dans beaucoup de développements en OCaml. Pour se faire, cette tâche travaillera sur le remplacement du préprocesseur actuel, Camlp4, par une extension du compilateur permettant de fournir des points d'extension dans la syntaxe et des préprocesseurs basés sur ces points d'extension : ces points d'extension ont pour but de permettre d'étendre la syntaxe d'OCaml sans avoir besoin de faire appel à Camlp4. Des premiers préprocesseurs basés sur ces points d'extension seront fournis pour remplacer ceux fournis par défaut par Camlp4. Un outil de vérification de style pourra aussi être développé sur cette technologie, pour vérifier en particulier l'inclusion des sources dans les limites du sous-langage SecurOCaml.
– OCamlPro fournit deux outils très utiles pour le développement en OCaml : OPAM, un nouveau gestionnaire de paquets source pour OCaml (près de 700 contributions open-source pour OPAM sont par exemple disponibles sur http ://opam.ocaml.org/), et `ocp-build`, un gestionaire de compilation. Ces outils forment l'environnement TypeRex, similaire à celui de `maven` pour Java, et vont prendre une place critique dans une usine logicielle pour OCaml. Cette tâche travaillera donc aussi à étudier les failles de sécurité que peuvent présenter les outils de TypeRex (OPAM et `ocp-build` en priorité), et à proposer des modifications des outils pour les éviter.

Numéro	Échéance	Intitulé livrable	Type	Resp.
3.2.1	T0+18	Préprocesseurs et points d'extensions	Logiciel	LexiFi
3.2.2	T0+24	OPAM et ocp-build sécurisés	Logiciel	OCamlPro

Sous-projet 4 : Outils d'audit de code OCaml

Ce sous-projet fournira un environnement (trousse à outils) pour vérifier l'absence de failles de sécurité classiques dans du code OCaml, et donc la qualité du code et l'utilisation de bonnes pratiques lors du développement de l'application. Ces outils pourront aussi être utilisés tout au long du développement, et amélioreront au final aussi bien la qualité en termes de sécurité qu'en terme de sûreté de fonctionnement. Tous les outils développés dans le cadre du projet seront diffusés en logiciel libre pour accroître l'attractivité du langage OCaml, et au sein d'une offre commerciale avec support professionnel par OCamlPro .

Tâche 4.1 : Détecteur statique d'exceptions

Durée	T0+0 à T0+24
Responsable	OCamlPro
Participants	OCamlPro (15 HM) ENSTA (4 HM) TrustInSoft (2 HM) CEA (2 HM)

L'étude LaFoSec a mis en évidence le problème des exceptions dans OCaml, qui ne sont pas vérifiées par le système de types, et constituent aujourd'hui un problème de sécurité important. Cette tâche vise donc à remédier à cette situation en développant un outil d'analyse statique de programmes OCaml pour détecter les exceptions qui ne sont pas gérées correctement. Cet outil sera développé de façon générique (à la Frama-C), pour pouvoir ensuite être étendu (après le projet SecurOCaml) avec de nouvelles analyses de sécurité. C'est une des tâches les plus complexes de SecurOCaml, l'analyse abstraite étant un des domaines de recherche faisant appel à une combinaison d'expertise à la fois en informatique (programmation et sémantique des programme) et en mathématiques (conception de domaines abstraits, fonctions de Galois, pour obtenir des résultats calculables dans un temps réalistes et suffisament précis pour être utilisables), comme les travaux de l'École Normale Supérieure sur Astrée [CCF+07] le montrent.

La conception du détecteur statique d'exceptions sera décomposée suivant le plan suivant :

- **Préparation :** Cette première partie a pour but de fournir un outil prenant en entrée l'ensemble des arbres de syntaxe typés (fichier `.cmt`) des modules d'une application OCaml, et effectuant la fusion de ces modules pour obtenir un seul arbre de syntaxe typé[Wee06], sur lesquelles les analyses sémantiques pourront être effectuées par la suite. La transformation implique la simplication de l'arbre par des analyses du type monomorphisation, contification [FW01], suppression de l'ordre supérieur [CJW00], etc.
- **Abstraction :** Cette partie consiste dans le développement proprement dit de la passe de détection des exceptions non-rattrapées (imprévues) dans le code d'un programme OCaml complet. Étant donné l'ordre supérieur d'OCaml, cette analyse complexe va devoir combiner des analyses de valeur et de flot, ainsi qu'une ré-inférence partielle des types. L'analyse initiale portera sur le noyau du langage OCaml, puis sera étendue pour traiter de plus en plus de constructions, d'abord du sous-langage à sécuriser, puis de l'ensemble du langage.
- **Présentation :** Cette partie aura pour but de développer une interface de présentation des résultats du détecteur d'exceptions, permettant à l'utilisateur de facilement retrouver l'origine des exceptions en naviguant dans le code, et de corriger le problème ou de l'annoter comme un faux-positif qui ne devra plus être affiché. L'interface pourra par la suite être étendue pour intégrer les sorties d'autres outils (détecteur de code inutilisé, couverture de code, etc.).

La plus grande partie du développement sera effectué par OCamlPro, en collaboration avec TrustInSoft et le CEA LIST, forts de leur expérience dans le développement d'un module d'analyse statique dans Frama-C, et avec l'ENSTA-ParisTech, où François Pessaux a déjà travaillé sur la détection d'exceptions.

Numéro	Échéance	Intitulé livrable	Type	Resp.
4.1.1	T0+12	Version initiale	Logiciel	OCamlPro
4.1.2	T0+24	Version sur SecurOCaml v1	Logiciel	OCamlPro

Tâche 4.2 : Détection de code inutilisé

Durée	T0+0 à T0+24
Responsable	LexiFi
Participants	OCamlPro (5 HM) LexiFi (5 HM)

Cette tâche travaillera sur la conception d'un outil de détection de code inutilisé. Cet outil permet de supprimer le code mort, dont on sait qu'il n'est pas utilisé par l'application, et permet donc de diminuer notablement le périmètre d'un audit du code. Il peut être aussi utilisé pour supprimer dès le développement le code mort du logiciel, comme certaines normes de certification l'imposent (DO-178B de l'avionique). L'outil effectuera une analyse en arrière, qui pourra être partiellement mélée à analyse de valeur pour pouvoir retirer du code atteignable par le flot mais pas par les valeurs. L'outil devra porter aussi bien sur les valeurs que sur les types, et sur leur décomposition (champ inutilisé

d'un record, par exemple), rendant l'analyse plus complexe que dans d'autres langages.

Numéro	Échéance	Intitulé livrable	Type	Resp.
4.2.1	T0+12	Version initiale	Logiciel	LexiFi
4.2.2	T0+24	Version sur SecurOCaml v1	Logiciel	OCamlPro

Tâche 4.3 : Interprète alternatif

Durée	T0+0 à T0+24
Responsable	Inria
Participants	Inria (11 HM) OCamlPro (1 HM)

Cette tâche travaillera sur un interprète alternatif pour exécuter du code OCaml sans faire appel à l'un des compilateurs actuels. Cet interprète alternatif ne devra pas partager de code avec les compilateurs actuels (pour garantir l'absence de bug commun), tout en traitant une large partie du langage. Une difficulté importante provient de la difficulté d'exécuter certaines constructions d'OCaml tout en ne maintenant qu'une information de typage très restreinte. Un tel interprète permettra d'exhiber des erreurs de programmation, liées à des hypothèses faites sur la suite utilisée pour exécuter le code de l'application.

Numéro	Échéance	Intitulé livrable	Type	Resp.
4.3.1	T0+18	Version initiale	Logiciel	Inria
4.3.2	T0+24	Version sur SecurOCaml v1	Logiciel	Inria

Sous-projet 5 : Vérifications à l'exécution

Ce sous-projet ajoutera à l'environnement d'exécution d'OCaml la possibilité de vérifier dynamiquement la correction de ses interactions avec le monde extérieur, en particulier pour deux types d'opérations : le chargement de bytecode durant l'exécution, et le chargement de données structurées.

Tâche 5.1 : Vérificateur de bytecode

Durée	T0+0 à T0+24
Responsable	ENSTA
Participants	ENSTA (15 HM) OCamlPro (3 HM) Inria (2 HM)

Cette tâche travaillera sur la vérification du bytecode chargé dynamiquement dans un programme OCaml. Pour cela, un nouveau système de types sera spécifié au niveau du bytecode, dont la correction garantira l'exécution correcte du bytecode par rapport à un certain nombre de propriétés de sûreté et de sécurité (par exemple, qu'un entier n'est jamais utilisé comme pointeur). Ce système de types sera implanté, d'une part dans le compilateur pour générer l'information de type du bytecode lors de la génération de celui-ci, puis dans la bibliothèque, pour tester lors du chargement d'un bytecode la conformité à la fois du bytecode à l'information de type, et de l'information de type au contexte d'exécution au moment du chargement du bytecode. Enfin, il s'agira de permettre l'exécution de ce bytecode dans des applications natives, ce qui n'est pas aujourd'hui possible.

Numéro	Échéance	Intitulé livrable	Type	Resp.
5.1.1	T0+24	Prototype d'interprète typé	Logiciel	ENSTA

Tâche 5.2 : Vérificateur de données structurées

Durée	T0+0 à T0+24
Responsable	ENSTA
Participants	ENSTA (15 HM) OCamlPro (3 HM) LexiFi (3 HM) Inria (2 HM)

Cette tâche travaillera sur la vérification des données structurées extraites de chaînes de caractères (issues du réseau ou de fichiers), et en particulier l'adéquation entre les données lues et le type attendu par le système. Pour cela, le système de types d'OCaml sera étendu avec de nouveaux types, les types à runtime, permettant de représenter à l'exécution le type des valeurs manipulées[Yal07]. Ces types seront sérialisés en même temps que les valeurs qu'ils représentent, pour permettre de vérifier, à la désérialisation, que les données sont compatibles avec celles attendues dans le contexte de chargement. Une partie du travail d'implantation consistera à modifier le typeur du compilateur pour gérer ces nouveaux types, et une autre partie à développer la bibliothèque qui manipulera ces types, d'une part pour vérifier que les données sont bien compatibles avec le type qui leur est associé, et d'autre part que ce type associé est bien compatible avec les contraintes du contexte de chargement.

Dans ce développement, l'ENSTA-ParisTech bénéficiera aussi de l'expérience dans le domaine des types à runtime engrengée par LexiFi (dont l'extension d'OCaml contient une version très spécifique de types à runtime), et par OCamlPro et l'Inria, qui ont aussi travaillé sur ce sujet dans le passé.

Numéro	Échéance	Intitulé livrable	Type	Resp.
5.2.1	T0+24	Prototype de vérificateur de données	Logiciel	ENSTA

Sous-projet 6 : Cas d'étude

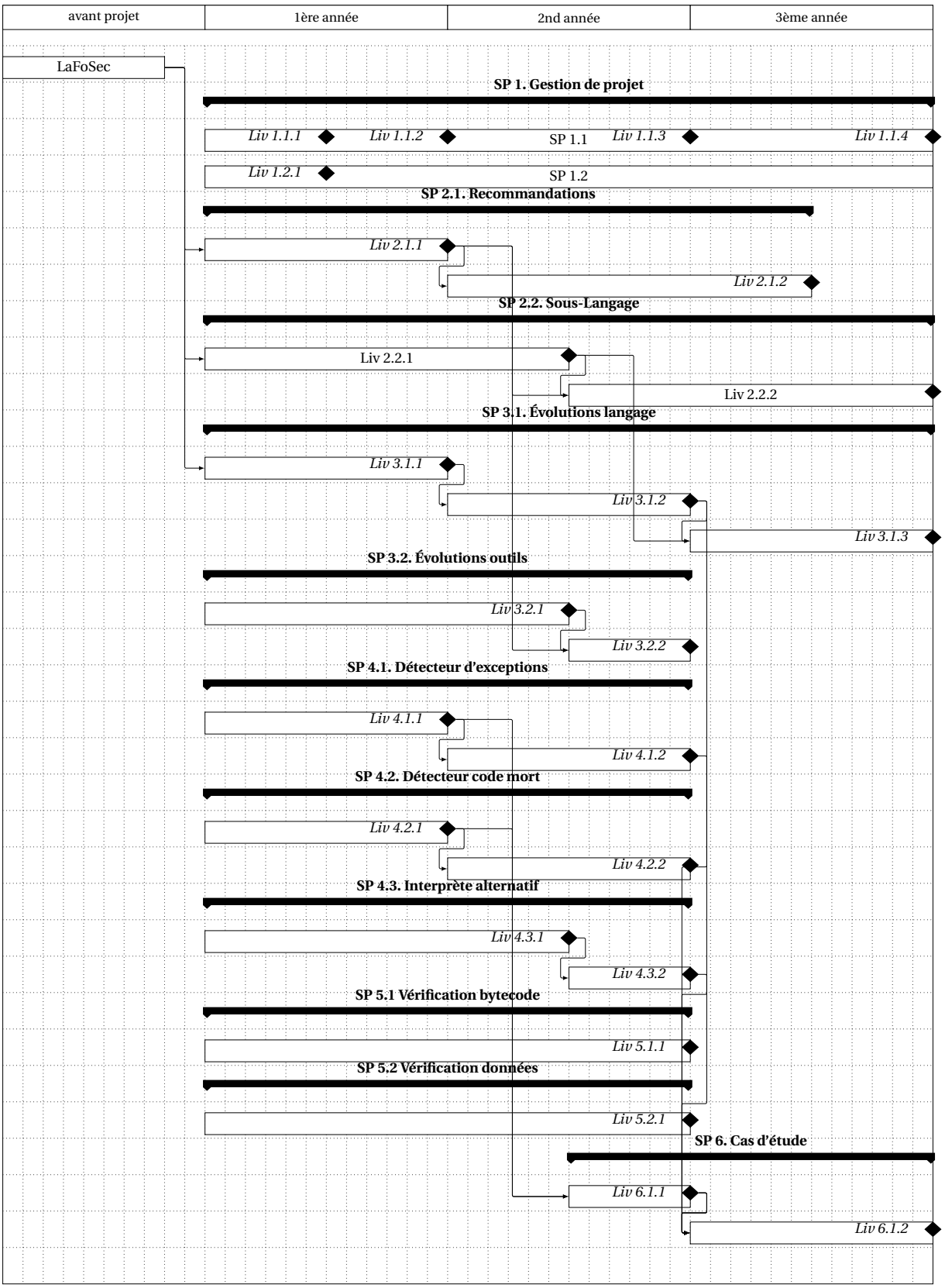
Durée	T0+18 à T0+36
Responsable	TrustInSoft
Participants	TrustInSoft (12 HM) CEA (10 HM) TrustedLabs (6 HM)

Ce sous-projet fournira des retours sur l'utilisation des résultats de SecurOCaml sur des applications sensibles en OCaml. Il s'agira principalement de la plateforme Frama-C[CSB+09], utilisée pour analyser des programmes écrits en C, à l'origine dans le domaine des systèmes critiques (avionique). Frama-C se compose d'un cœur permettant de lire des programmes, et de greffons (plugins) permettant d'ajouter des analyses à effectuer sur les programmes. La plateforme Frama-C est développée par le CEA LIST, qui la déploie chez plusieurs de ses partenaires industriels, et utilisée par TrustInSoft pour vérifier leurs composants de cyber-sécurité.

Numéro	Échéance	Intitulé livrable	Type	Resp.
6.1.1	T0+24	Retours d'évaluation de SecurOCaml v1	Document	CEA
6.1.2	T0+36	Retours d'évaluation de SecurOCaml version finale	Document	TrustInSoft

7.1 Récapitulatif

Le GANTT suivant montre les dépendances entre les sous-projets, tâches et leurs livrables. L'étude LaFoSec est indiquée, car beaucoup de ses résultats sont à l'origine des différentes tâches.



Le tableau suivant récapitule les livrables et leurs responsables :

Numéro	Échéance	Intitulé livrable	Type	Resp.
SP 1. : Gestion de projet				
SP 1.1 : Gestion de projet				OCamlPro
1.1.1	T0+6	Kick-off et accord de consortium	Document	OCamlPro
1.1.2	T0+12	Rapport annuel d'activité	Document	OCamlPro
1.1.3	T0+24	Rapport annuel d'activité	Document	OCamlPro
1.1.4	T0+36	Rapport annuel d'activité	Document	OCamlPro
SP 1.2 : Diffusion et valorisation				OCamlPro
1.2.1	T0+6	Mise en place des sites web et forges	Logiciel	OCamlPro
SP 2. : Recommandation et Sous-Langage				
SP 2.1 : Recommandations				SafeRiver
2.1.1	T0+12	Mise à jour des recommandations pour 4.00	Document	SafeRiver
2.1.2	T0+30	Mise à jour des recommandations pour 4.02	Document	SafeRiver
SP 2.2 : Sous-langage SecurOCaml				OCamlPro
2.2.1	T0+18	Définition du sous-langage SecurOCaml v1	Document	OCamlPro
2.2.2	T0+36	Définition du sous-langage SecurOCaml v2	Document	OCamlPro
SP 3. : Évolution du langage				
SP 3.1 : Évolution de OCaml				Inria
3.1.1	T0+12	Release d'OCaml 4.03	Logiciel	Inria
3.1.2	T0+24	Release d'OCaml 4.04	Logiciel	Inria
3.1.3	T0+36	Release d'OCaml 4.05	Logiciel	Inria
SP 3.2 : Évolution des outils				LexiFi
3.2.1	T0+18	Préprocesseurs et points d'extensions	Logiciel	LexiFi
3.2.2	T0+24	OPAM et ocp-build sécurisés	Logiciel	OCamlPro
SP 4. : Outils d'audit				
SP 4.1 : Analyseur d'exceptions				OCamlPro
4.1.1	T0+12	Version initiale	Logiciel	OCamlPro
4.1.2	T0+24	Version sur SecurOCaml v1	Logiciel	OCamlPro
SP 4.2 : Détection de code mort				LexiFi
4.2.1	T0+12	Version initiale	Logiciel	LexiFi
4.2.2	T0+24	Version sur SecurOCaml v1	Logiciel	OCamlPro
SP 4.3 : Interprète alternatif				Inria
4.3.1	T0+18	Version initiale	Logiciel	Inria
4.3.2	T0+24	Version sur SecurOCaml v1	Logiciel	Inria
SP 5. : Vérifications à l'exécution				
SP 5.1 : Vérificateur de bytecode				ENSTA
5.1.1	T0+24	Prototype d'interprète typé	Logiciel	ENSTA
SP 5.2 : Vérificateur de données				ENSTA
5.2.1	T0+24	Prototype de vérificateur de données	Logiciel	ENSTA
SP 6. : Cas d'étude				
SP 6.1 : Validation de SecurOCaml				TrustInSoft
6.1.1	T0+24	Retours d'évaluation de SecurOCaml v1	Document	CEA
6.1.2	T0+36	Retours d'évaluation de SecurOCaml version finale	Document	TrustInSoft

Finalement, le tableau ci-dessous montre l'implication de chaque partenaire dans les différents sous-projets (en gras si responsable de la tâche) :

Partenaire	HM par sous-projet												Total
	1.1	1.2	2.1	2.2	3.1	3.2	4.1	4.2	4.3	5.1	5.2	6	HM
OCamlPro	**6**	**4**		**4**	5	8	**15**	5	1	3	3		54
SafeRiver	1		**14**	1									16
LexiFi	1			2	3	**4**		**5**			3		18
TrustInSoft	1			1			2					**12**	16
Inria	1	2	5	1	**12**				**11**	2	2		36
ENSTA	1			1			4			**15**	**15**		36
CEA	1			1			2					10	14
TrustedLabs	1											6	7
Total	13	6	19	11	20	12	23	10	12	20	23	28	197

8 Partenaires et tableau de financement

8.1 Partenaires PME

8.1.1 OCamlPro (porteur du projet, solutions pour OCaml)

OCamlPro est une jeune spin-off de l'INRIA, qui propose des solutions (logiciels et services) aux industriels désirant utiliser OCaml dans leurs développements. Ainsi, OCamlPro édite TypeRex, une studio de développement pour OCaml, et fournit différents outils pour améliorer la productivité des développeurs, et la qualité des logiciels en OCaml. OCamlPro fournit aussi tous les services (formation, consulting, support, développement, etc.) dont une entreprise peut avoir besoin pour utiliser OCaml. Finalement, OCamlPro a pris un rôle moteur dans le développement et la maintenance du langage OCaml, fourni par l'INRIA, et en particulier ses compilateurs et runtimes. OCamlPro est aujourd'hui la **seule entreprise dans le monde** à fournir du service professionnel sur OCaml, et presque tout **son chiffre d'affaire est à l'international** (Jane Street, Citrix, en négociation avec Facebook, etc.).

Rôle du partenaire : Coordinateur, responsable du sous-projet 1, et des tâches 2.2 et 4.2, développement d'outils d'analyse de code OCaml, modification du langage OCaml.

Les dépenses envisagées dans le cadre du projet sont de 54 homme.mois d'ingénieur R&D.

Partenaire	OCamlPro (N^o SIRET : 531 468 429 00018)
Département	91 (Orsay)
Dépenses engagées	504 k€ (Aide demandée : 226 k€)
Contact	Fabrice LE FESSANT (+33 6 72 73 37 53) Fabrice.Le_Fessant@ocamlpro.com

8.1.2 SafeRiver (développement en sécurité–sûreté)

SafeRiver développe depuis 2008 des services et prestations autour de la convergence sûreté-sécurité des systèmes embarqués, rendue nécessaire du fait de l'ouverture de ces systèmes (interactions plus nombreuses avec d'autres systèmes, intégration de composants « COTS » ou libres notamment). Le modèle économique est fondé sur la fourniture et la vente de services outillés à forte valeur ajoutée.

SafeRiver développe une plateforme d'analyse statique, qui intègre d'une part des moteurs d'analyse

tiers (Polyspace The Mathworks, Frama-C du CEA LIST) et d'autre part des composants propriétaires qui s'appuient sur ces moteurs pour réaliser des analyses plus poussées : vérification de propriétés sur le flot de contrôle, vérification de propriétés et de politiques de sécurité, analyse de valeurs par exemple. Cette approche a été validée dans le domaine des propriétés de sûreté sur des codes embarqués dans des applications ferroviaires.

Dans le cadre de ses activités de recherche et développement, SafeRiver cherche également à appliquer ces techniques dans le domaine de la sécurité des systèmes d'information embarqués ou critiques.

L'objectif est de disposer de méthodes et outils pour calculer des propriétés de robustesse en regard de faiblesses de code qui pourraient être exploitées par des attaquants. SafeRiver a ainsi développé un prototype en OCaml qui intègre des noyaux d'analyse propriétaire offrant différentes fonctionnalités nécessaires aux analyses de sécurité pour le langage C. Plusieurs prototypes ont été développés à partir de ces noyaux et expérimentés sur différents périmètres d'attaque dans le cadre des projets FEDER Baccarat, ISI Prometheus. D'autres sont en cours de développement pour les projets CORAC AME et PISCO.

SafeRiver a dirigé l'étude LaFoSec commanditée par l'ANSSI qui avait pour but d'évaluer les apports et les défauts des langages fonctionnels pour la sécurité. Plusieurs langages fonctionnels ont été étudiés OCaml, F#, Scala, Haskell, Scheme, CLISP, Erlang puis l'étude a été approfondie sur les langages OCaml, F# et Scala et finalement la sécurité d'OCaml a été analysée exhaustivement. Ce travail sur OCaml a donné lieu à la rédaction par le consortium (Normation, Cédric, SafeRiver) de deux documents qui servent de socle à ce projet :

– "Recommandations relatives à l'utilisation du langage OCaml et à l'installation et la configuration des outils associés". Ce document décrit les recommandations à suivre par le développeur d'une application OCaml pour lequel la sécurité est importante.
– "Stratégie d'évolution du langage OCaml et des outils associés". Ce document décrit les évolutions d'OCaml permettant de mieux garantir la sécurité des applications développées dans ce langage.

Rôle du partenaire : Responsable de la tâche 2.1, mise à jour des évolutions du langage OCaml et des recommandations de codage, spécification d'un sous langage adapté aux applications de sécurité et des besoins en terme d'outils.

Les dépenses envisagées dans le cadre du projet sont de 14 homme.mois d'ingénieur R&D.

Partenaire	SafeRiver (N^o SIRET : 485 374 136 00029)
Département	75
Dépenses engagées	165 k€ (Aide demandée : 74 k€)
Contact	Christèle Faure (+33 4 93 77 71 08) Christele.Faure@safe-river.com

8.1.3 LexiFi (applications bancaires)

LexiFi est un éditeur de logiciels dans le domaine des produits financiers structurés, dont la principale contribution technologique est un langage embarqué de description formelle des contrats financiers ("algèbre de contrats"). LexiFi a fait dès sa création le choix d'OCaml comme langage d'implémentation privilégié, du fait de sa parfaite adéquation aux manipulations autour de l'algèbre de contrats. LexiFi expose à ses propres clients une version du compilateur OCaml pour leur permettre d'étendre ses applications via un mécanisme d'addin. LexiFi est de plus fortement impliqué dans le développement du langage OCaml lui-même, et expérimente en interne certaines évolutions du langage le plus souvent destinées à être intégrées dans la version officielle. Depuis 2007, LexiFi participe directement

à l'évolution du langage OCaml et a été impliqué dans certains des principaux ajouts au langage depuis cette date.

Rôle du partenaire : Responsable des tâches 3.2 (évolution des outils) et 4.2 (détection de code mort). Les dépenses envisagées dans le cadre du projet sont de 18 homme.mois d'ingénieur.

Partenaire	LexiFi (N^o SIRET : 430 477 000 00045)
Département	92 (Boulogne-Billancourt)
Dépenses engagées	253 k€ (Aide demandée : 113 k€)
Contact	Alain Frisch (+33 1 41 10 02 64) alain.frisch@LexiFi.com

8.1.4 TrustInSoft (vérifications de cybersécurité)

TrustInSoft est une jeune startup d'édition logicielle essaimée par le CEA LIST et portée par les fondateurs de la technologie Frama-C. TrustInSoft commercialise des outils construits à partir de Frama-C afin d'en permettre des usages industriels plus efficaces et plus automatiques dans le domaine de la cyber-sécurité et de la sûreté de fonctionnement.

Rôle du partenaire : Responsable de la tâche 6 (cas d'étude).

Les dépenses engagées dans le cadre du projet sont de 16 homme.mois d'ingénieur.

Partenaire	TrustInSoft (N^o SIRET : 793 371 907 00015)
Département	91 (Orsay)
Dépenses engagées	230 k€ (Aide demandée : 103 k€)
Contact	Benjamin Monate (+33 970 44 75 87) benjamin.monate@trust-in-soft.com

8.2 Partenaires académiques

8.2.1 Inria (maintenance et évolution d'OCaml)

Inria est un établissement public à caractère scientifique et technologique (EPST). Inria se compose de 4.100 personnes réparties dans 8 centres de recherche, dont le centre de Paris–Rocquencourt. L'équipe Gallium travaille sur les langages de programmation et la preuve de programmes. En particulier, l'équipe maintient et diffuse la version officielle du système OCaml.

Rôle du partenaire : Développements dans OCaml.

Les dépenses envisagées dans le cadre du projet sont de 12 homme.mois de chercheur et de 24 homme.mois de post-doc.

Partenaire	Inria (N^o SIRET : 180 089 047 00013)
Département	78
Dépenses engagées	273 k€ (Aide demandée : 109 k€)
Contact	Damien Doligez (+33 1 39 63 57 90) Damien.Doligez@inria.fr

8.2.2 ENSTA-ParisTech (typage et vérification formelle)

l'ENSTA-ParisTech est une grande école d'ingénieurs sous tutelle du Ministère de la Défense. École d'application de l'École Polytechnique, l'ENSTA-ParisTech est un établissement public à caractère administratif (EPA), attribue le diplôme d'ingénieur à plus de 180 élèves par an et emploie près de 200 personnes dont plus d'une centaine d'enseignants-chercheurs.

Le laboratoire U2IS (Informatique et Ingénierie des Systèmes) de l'ENSTA-ParisTech développe des recherche en ingénierie système, en robotique et vision et en sûreté et fiabilité des logiciels. C'est dans ce dernier thème que s'inscrivent les travaux décrits ici.

Rôle du partenaire : Responsable du sous-projet 5 (analyse, typage et compilation du bytecode, et désérialisation sûre). Les dépenses envisagées dans le cadre du projet sont de 10.8 homme.mois de directeur de recherche, et 24 homme.mois de post-doc.

Partenaire	ENSTA-ParisTech (N^o SIRET : 197 500 036 00011)
Département	91
Dépenses engagées	369 k€ (Aide demandée : 147 k€)
Contact	Michel Mauny (+33 1 81 87 20 32) Michel.Mauny@ensta-paristech.fr

8.2.3 CEA LIST (vérification de sûreté de fonctionnement)

Le CEA est un organisme de recherche présent dans quatre grands domaines : les énergies bas carbone (nucléaire et renouvelables), les technologies pour l'information et les technologies pour la santé, les Très Grandes Infrastructures de Recherche (TGIR), la défense et la sécurité globale. Pour chacun de ces quatre grands domaines, le CEA s'appuie sur une recherche fondamentale d'excellence et assure un rôle de soutien à l'industrie. Le CEA est un Établissement Public à Caractère Industriel et Commercial (EPIC), employant environ 15000 personnes dans 10 centres de recherches, dont le centre de Saclay.

L'institut CEA LIST regroupe des activités de recherche et d'innovation structurées autour des systèmes embarqués, des systèmes interactifs, des capteurs et du traitement du signal. Au sein de cet institut, le Laboratoire de Sûreté Logiciel (LSL) développe des outils pour la vérification et la validation de logiciels et de composants matériels. le LSL développe notamment l'outil Frama-C, écrit en OCaml, dédié à l'analyse de code C.

Rôle du partenaire : Le CEA travaillera sur le cas d'usage Frama-C avec TrustInSoft, qui l'utilise pour certifier ses composants. Les dépenses envisagées dans le cadre du projet sont de 14 homme.mois d'un ingénieur-chercheur.

Partenaire	CEA
N^o SIRET	77568501900488
Département	91
Dépenses engagées	194 k€ (Aide demandée : 77 k€)
Contact	Florent Kirchner (+33 1 69 08 45 19) florent.kirchner@cea.fr

8.3 Partenaire grand groupe (non financé)

8.3.1 Trusted Labs (sécurité des cartes à puces)

Société de conseil spécialisée dans la sécurité des systèmes ouverts et connectés, possède une longue expertise dans les évaluations sécuritaires, dans le domaine des Critères Communs ou dans des schémas prioritaires tel que EMVCo. Trusted Labs réalise depuis 8 ans des évaluations sécuritaires de cartes à puces et de systèmes embarqués, et a développé une forte expertise dans le niveau des attaques logiques, en particulier en combinaison avec des attaques physiques. Par ailleurs, Trusted Labs possède une expertise reconnue dans le domaine de la méthodologie sécuritaire et une connaissance approfondie des Critères Communs. La provenance de nos collaborateurs se partage entre recherche et industrie, favorisant un croisement de connaissances, ce qui nous permet d'être toujours à l'avant-garde et d'offrir des services très avancés (par exemple, basés sur les méthodes formelles) tels que les évaluations de niveau EAL7 ou les évaluations d'applications mobiles.). Certains des outils d'évaluation sécuritaires sont développés en interne en OCaml et ont pour vocation d'être utilisés dans les certifications. D'où l'importance du projet SecurOCaml qui permettra d'améliorer la confiance et la robustess dans ces outils et dans par conséquent dans les produits évalués.

Rôle du partenaire : Observateur, retour d'expérience sur l'utilisation d'OCaml dans un contexte de sécurité.

Partenaire	Trusted Labs (N^o SIRET : 483 191 227 00014)
Département	78
Aide demandée	0 k€ (non financé)
Contact	Boutheina Chetali (+33 1 30 97 26 07) boutheina.chetali@trusted-labs.com

8.4 Tableau de financement

Partenaire	Type	Dépt	Dépenses k€	Aide k€	Rôle
OCamlPro	PME	91	504	226	Coordinateur, Développement
SafeRiver	PME	91	165	74	Audit Sécurité
LexiFi	PME	92	253	113	Développement
TrustInSoft	PME	91	230	103	Démonstrateur
Total PME			1152	516	
Inria	EPST	78	273	109	Développement
ENSTA-ParisTech	EPA	91	369	147	Développement
CEA	EPIC	91	194	77	Démonstrateur
Total Laboratoires			836	333	
Trusted Labs	ETI	78	0	0	Observateur
Total Grands Groupes			0	0	
Total			1988	849	

9 Informations complémentaires

Le projet est-il soumis à d'autres pôles de compétitivité pour co-labellisation ? Non, le projet est labellisé par les deux groupes CN&S (Confiance Numérique et Sécurité) et GTLL (Logiciel Libre) du seul pôle Systematic Paris-Region.

Une première version de ce projet a-t-elle déjà été soumise à un autre appel à projet ? Oui, appel FUI16 et FUI17. Les modifications apportées au document sont présentées dans la section 1.1.

Annexes

Références

[AA14] Eric Jaeger (ANSSI) and Olivier Levillain (ANSSI). Mind your language, a discussion about languages and security. In 2014 IEEE Security and Privacy Workshops, 2014.

[ASIN13] ANSSI, SafeRiver, INRIA, and Normation. LaFoSec : Sécurité et langages fonctionnels, 2013. http ://www.ssi.gouv.fr/fr/anssi/publications/publications-scientifiques/autres-publications/lafosec-securite-et-langages-fonctionnels.html.

[CCF+07] Patrick COUSOT, Radhia COUSOT, Jerome FERET, Antoine MINE, Laurent MAUBORGNE, David MONNIAUX, and Xavier RIVAL. Varieties of static analyzers : A comparison with astree. In Proceedings of the First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE '07, pages 3–20, Washington, DC, USA, 2007. IEEE Computer Society.

[CCF+09] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. Why does astrée scale up ? Form. Methods Syst. Des., 35(3) :229–264, 2009.

[CCKL07] Sylvain Conchon, Evelyne Contejean, Johannes Kanig, and Stéphane Lescuyer. Lightweight integration of the ergo theorem prover inside a proof assistant. In AFM '07 : Proceedings of the second workshop on Automated formal methods, pages 55–59, New York, NY, USA, 2007. ACM.

[CDDM12] P. Cuoq, D. Delmas, S. Duprat, and V. Moya Lamiel. Fan-C, a Frama-C plug-in for data flow verification. In the Embedded Real-Time Software and Systems Congress (ERTS^2 2012), 2012.

[CJW00] Henry Cejtin, Suresh Jagannathan, and Stephen Weeks. Flow-directed closure conversion for typed languages. In In ESOP'00, pages 56–71. Springer-Verlag, 2000.

[CKK+12] Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-C : A software analysis perspective. In Proceedings of the 10th International Conference on Software Engineering and Formal Methods, SEFM'12, pages 233–247, Berlin, Heidelberg, 2012. Springer-Verlag.

[CSB+09] Pascal Cuoq, Julien Signoles, Patrick Baudin, Richard Bonichon, Géraud Canet, Loïc Correnson, Benjamin Monate, Virgile Prevosto, and Armand Puccetti. Experience report : Ocaml for an industrial-strength static analysis framework. In ICFP '09 : Proceedings of the 14th ACM SIGPLAN international conference on Functional programming, pages 281–286, New York, NY, USA, 2009. ACM.

[DDLS10] D. Delmas, S. Duprat, V. Moya Lamiel, and J. Signoles. Taster, a Frama-C plug-in to encode Coding Standards. In the Embedded Real-Time Software and Systems Congress (ERTS^2 2010), 2010.

[DG94] Damien Doligez and Georges Gonthier. Portable, unobtrusive garbage collection for multiprocessor systems. In POPL '94 : Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 70–83, New York, NY, USA, 1994. ACM.

[DL93] Damien Doligez and Xavier Leroy. A concurrent, generational garbage collector for a multithreaded implementation of ml. In POPL '93 : Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 113–123, New York, NY, USA, 1993. ACM.

[Ebe09] Jean-Marc Eber. The financial crisis, a lack of contract specification tools : What can finance learn from programming language design ? In Proceedings of the 18th European Symposium on Programming Languages and Systems : Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, ESOP '09, pages 205–206, Berlin, Heidelberg, 2009. Springer-Verlag.

[FW01] Matthew Fluet and Stephen Weeks. Contification using dominators. In ICFP '01 : Proceedings of the sixth ACM SIGPLAN international conference on Functional programming, pages 2–13, New York, NY, USA, 2001. ACM.

[Gar01] Jacques Garrigue. Labeled and optional arguments for objective caml. In In JSSST Workshop on Programming and Programming Languages, 2001.

[Gar04] Jacques Garrigue. Typing deep pattern-matching in presence of polymorphic variants. In JSSST Workshop on Programming and Programming Languages, Gamagori, Japan, March 2004.

[HL08] Charles Hymans and Olivier Levillain. Newspeak, Doubleplussimple Minilang for Goodthinkful Static Analysis of C. Technical Note 2008-IW-SE-00010-1, EADS IW/SE, 2008.

[KRJ09] Ming Kawaguchi, Patrick Rondon, and Ranjit Jhala. Type-based data structure verification. In PLDI '09 : Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation, pages 304–315, New York, NY, USA, 2009. ACM.

[Ler90] Xavier Leroy. Efficient data representation in polymorphic languages. In PLILP '90 : Proceedings of the 2nd International Workshop on Programming Language Implementation and Logic Programming, pages 255–276, London, UK, 1990. Springer-Verlag.

[Ler93] Xavier Leroy. Polymorphism by name for references and continuations. In POPL '93 : Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 220–231, New York, NY, USA, 1993. ACM.

[Ler94] Xavier Leroy. Manifest types, modules, and separate compilation. In POPL '94 : Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 109–122, New York, NY, USA, 1994. ACM.

[Ler09] Xavier Leroy. A formally verified compiler back-end. J. Autom. Reason., 43(4) :363–446, December 2009.

[LFM01] Fabrice Le Fessant and Luc Maranget. Optimizing pattern matching. In ICFP '01 : Proceedings of the sixth ACM SIGPLAN international conference on Functional programming, pages 26–37, New York, NY, USA, 2001. ACM.

[LPSW03] James J. Leifer, Gilles Peskine, Peter Sewell, and Keith Wansbrough. Global abstraction-safe marshalling with hash types. In ICFP '03 : Proceedings of the eighth ACM SIGPLAN international conference on Functional programming, pages 87–98, New York, NY, USA, 2003. ACM.

[LR98] Xavier Leroy and François Rouaix. Security properties of typed applets. In POPL '98 : Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 391–403, New York, NY, USA, 1998. ACM.

[Mar03] Luc Maranget. Les avertissements du filtrage. In Jean-Christophe Filliâtre, editor, JFLA, Collection Didactique, pages 3–20. INRIA, 2003.

[Mar08] Luc Maranget. Compiling pattern matching to good decision trees. In ML '08 : Proceedings of the 2008 ACM SIGPLAN workshop on ML, pages 35–46, New York, NY, USA, 2008. ACM.

[MHD+07] Anil Madhavapeddy, Alex Ho, Tim Deegan, David Scott, and Ripduman Sohan. Melange : creating a "functional" internet. In EuroSys '07 : Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pages 101–114, New York, NY, USA, 2007. ACM.

[MMR+13] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels : Library operating systems for the cloud. SIGPLAN Not., 48(4) :461–472, March 2013.

[MW08] Yaron Minsky and Stephen Weeks. Caml trading – experiences with functional programming on wall street. J. Funct. Program., 18(4) :553–564, 2008.

[Ode06] Martin Odersky. The scala experiment : can we provide better language support for component systems ? In POPL '06 : Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 166–167, New York, NY, USA, 2006. ACM.

[PAM+09] Bruno Pagano, Olivier Andrieu, Thomas Moniot, Benjamin Canou, Emmanuel Chailloux, Philippe Wang, Pascal Manoury, and Jean-Louis Colaço. Experience report : using objective caml to develop safety-critical embedded tools in a certification framework. In ICFP '09 : Proceedings of the 14th ACM SIGPLAN international conference on Functional programming, pages 215–220, New York, NY, USA, 2009. ACM.

[PCC+07] Bruno Pagano, Benjamin Canou, Emmanuel Chailloux, Jean-Louis Colaço, and Philippe Wang. Couverture de code caml pour la réalisation d'outils de développement certifiés. In Actes des Journées Francophones des Langages Applicatifs (JFLA 2007), 2007.

[PL99] François Pessaux and Xavier Leroy. Type-based analysis of uncaught exceptions. In POPL '99 : Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 276–290, New York, NY, USA, 1999. ACM.

[TMC+96] D. Tarditi, G. Morrisett, P. Cheng, C. Stone, R. Harper, and P. Lee. Til : a type-directed optimizing compiler for ml. In PLDI '96 : Proceedings of the ACM SIGPLAN 1996 conference on Programming language design and implementation, pages 181–192, New York, NY, USA, 1996. ACM.

[TSV09] David Teller, Arnaud Spiwack, and Till Varoquaux. Catch me if you can : Looking for type-safe, hierarchical, lightweight, polymorphic and efficient error management in ocaml. In Pre-proceedings of the international symposium on Implementation of Functional Languages (IFL 2009), 2009.

[VBL+14] Julien Verlaguet, Joel Beales, Eugene Letuchy, Gabriel Levi, Joel Marcey, Erik Meijer, Alok Menghrajani, Bryan O'Sullivan, Drew Paroski, James Pearce, Joel Pobar, and Joshua Van Dyke Watzman. Hack : a new programming language for HHVM, 2014. https ://code.facebook.com/posts/264544830379293/hack-a-new-programming-language-for-hhvm/.

[Wee06] Stephen Weeks. Whole-program compilation in mlton. In ML '06 : Proceedings of the 2006 workshop on ML, pages 1–1, New York, NY, USA, 2006. ACM.

[WJC11] Philippe Wang, Adrien Jonquet, and Emmanuel Chailloux. Non-intrusive structural coverage for objective caml. Electron. Notes Theor. Comput. Sci., 264(4) :59–73, February 2011.

[Yal07] Jeremy Yallop. Practical generic programming in ocaml. In ML '07 : Proceedings of the 2007 workshop on Workshop on ML, pages 83–94, New York, NY, USA, 2007. ACM.

[Yi94] Kwangkeun Yi. Compile-time detection of uncaught exceptions in standard ml programs. In Lecture Notes in Computer Science, pages 238–254. Springer-Verlag, 1994.

[YR02] Kwangkeun Yi and Sukyoung Ryu. A cost-effective estimation of uncaught exceptions in standard ml programs. Theor. Comput. Sci., 277(1-2) :185–217, 2002.