

# Cool stuff from this year 2018

Rehan MALAK

Séminaire FUN

Novembre 2018

- 1 Deux conférences en France
- 2 Pourquoi ces 2 conférences
- 3 Le langage **F**\*
- 4 HAACL\*
- 5 Wireguard

1 ) Deux conférences en France

Symposium sur la sécurité des technologies de l'information et des communications.

À Rennes du 13 au 15 juin 2018, au Couvent des Jacobins.



- conférence francophone annuelle sur le thème de la sécurité de l'information (hardware, systèmes, réseaux, crypto, menaces cyber ...)
- 16ème édition
- 30 conférences, présentées par 62 personnes (61 hommes et 1 femme)
- trustés par l'ANSSI, la DGA-MI, Quarkslab et des centaines d'adminsys paranos
- 800 places (early bird parties en qqs minutes)

## Vendredi 15 juin 2018

09:45	<i>A Practical Guide to Differential Power Analysis of USIM Cards</i>	Adrian Thillard christophe Devine Manuel San Pedro	15 min.
10:00	<i>Starve for Erlang cookie to gain remote code exec</i>	guillaume teissier Guillaume Kaim	15 min.
10:15	<i>HACL* une bibliothèque de cryptographie formellement vérifiée dans Firefox</i>	Benjamin Beurdouche Jean Karim Zinzindohoue	30 min.
10:45	<i>Pause</i>		
11:15	<i>WireGuard – Next Generation Secure Network Tunnel [Conférence invitée]</i>	Jason Donenfeld	45 min.
12:00	<i>Cà se passe à Paris I</i>	Yves CFM'ERT	30 min.
12:30	<i>Dans les coulisses de l'équipe sécurité Debian</i>	Yves-Alexis Perez	15 min.
12:45	<i>Déjeuner</i>		
14:45	<i>Hacking Harness open-source</i>	Ivan Kwiatkowski	15 min.
15:00	<i>DNS Single Point of Failure Detection using Transitive Availability Dependency Analysis</i>	Florian Maury	30 min.
15:30	<i>[Conférence de clôture par le directeur technique de la DGSE]</i>	Patrick Pailloux	60 min.
16:30	<i>Fin de la conférence</i>		

Profil différent mais utilisant les mêmes outils :









- ⇒ 1 doctorant et 1 permanent de l'Inria Paris
- ⇒ 1 chercheur indépendant en sécurité (conférencier habitué de Defcon, Black Hat, ...)

Embedded/Kernel Recipes. Paris du 24 au 28 septembre 2018.

À la fondation Mozilla Paris.



- conférence anglophone annuelle sur le thème du développement du noyau Linux
- par les développeurs, principalement pour les développeurs
- Kernel : 7ème édition : 21 présentations (19 hommes et 2 femmes)
- Embedded : 2ème édition : 15 présentations (14 hommes et 1 femme)

26/09/2018		27/09/2018	28/09/2018
		AM	PM
9h30-10h	Coffee		 New GPIO Interface for User Space Bartosz Golaszewski
10h-10h40	 KernelShark 1.0: What's new and what's coming Steven Rostedt	14h20-15h	 CLUP OS: Building a defense-in-depth OS around Linux kernel... Mickael Salaün, Timothée Ravier 
10h50-11h30	 XDP: a new programmable network layer Jesper Dangaard Brouer		Coffee
11h40-12h20	 CPU Idle Loop Rework Rafael J. Wysocki	16h20-17h	 Zinc: New Cryptography API for Linux Kernel Jason Donenfeld
12h30-14h20	Lunch time	17h10-17h50	 Democracy requires free software Matthias Kirschner Charity Auction in favor of FSFE

⇒ le même conférencier qu'à SSTIC

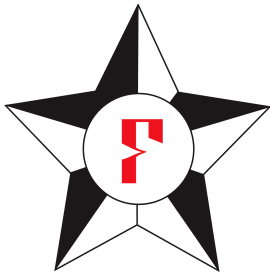
2 ) Pourquoi ces 2 conférences



Pleins de bonnes raisons :

- Réseaux
  - Sécurité
  - Cryptographie
  - Vérification
  - INRIA
- ⇒ Overlap intéressant entre les thématiques FUN et VESSEDIA
- ⇒ Une alternative à Frama-C pour la vérification mais aussi comme modèle de développement
- ⇒ Usage sans le savoir par des millions d'utilisateurs et d'utilisatrices...
- ⇒ Comment se déroule un processus d'intégration dans Mozilla Firefox ou dans le noyau Linux en 2018, les 2 projets libres les plus connus et répandus ?

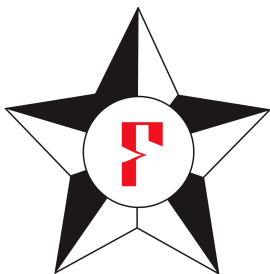
### 3 ) Le langage $F^*$



Microsoft Research - Inria  
**JOINT CENTRE**

**F\*** (“F-star”) :

- Langage fonctionnel de la famille ML comme OCaml, StandardML, ou **F#** (“F-sharp”), licence Apache2.0
  - conçu pour la vérification
  - théorie : types dépendants, effets monadiques, raffinement de types, WP (“Weakest Precondition”)
  - vérification déductive via SMT (Z3) et preuves manuelles si besoin
  - traduction vers OCaml ou **F#**
  - traduction vers du C **lisible** via KreMlin ! (**lisible** est important)
- ⇒ Projet Everest, HACL\* , Wireguard (noyau Linux ?), ...



Microsoft Research - Inria  
**JOINT CENTRE**

Compilateur  $F^*$  (“F-star”) est écrit en  $F^* \cap F\#$  :

- 1 mono (.NET pour Linux) + compilateur  $F\#$   $\Rightarrow$  compilateur  $F^*$
- 2 compilateur  $F^*$  + sources  $F^*$   $\Rightarrow$  sources OCaml compilateur  $F^*$
- 3  $\Rightarrow$  nouveau compilateur  $F^*$   $\Rightarrow$  librairies...

Pas réussi : anciennes sources OCaml du compilateur  $F^*$  à la place de l'étape 1 (c'est à dire 3-2-3 au lieu de 1-2-3)

Avec  $F^*$ , on part d'une syntaxe purement fonctionnelle OCaml :

- Recursive functions

```
val factorial : int -> int
let rec factorial n = (if n = 0 then 1 else n * (factorial (n - 1)))
```

- Inductive datatypes (immutable) and pattern matching

```
type list (a:Type) =
| Nil : list a
| Cons : hd:a -> tl:list a -> list a

val map : ('a -> 'b) -> list 'a -> list 'b
let rec map f x = match x with
| [] -> []
| h :: t -> f h :: map f t
```

- Lambdas (unnamed, first-class functions)

```
map (fun x -> x + 42) [1;2;3]
```

du raffinement de types :

```
type nat = x:int{x>=0}
```

et du type dépendant :

- Dependent function types (II), here together with refinements:

```
val incr : x:int -> y:int(x < y)
let incr x = x + 1
```

## Raffinement de types :

```
type filename = string

(** [canWrite] is a function specifying whether a file [f] can be written *)
let canWrite (f:filename) =
  match f with
  | "demo/tempfile" → true
  | _ → false

(** [canRead] is also a function ... *)
let canRead (f:filename) =
  canWrite f          (* writeable files are also readable *)
  || f="demo/README" (* and so is demo/README *)

val read : f:filename{canRead f} → ML string
let read f = FStar.IO.print_string ("Dummy read of file " ^ f ^ "\n"); f

val write : f:filename{canWrite f} → string → ML unit
let write f s = FStar.IO.print_string ("Dummy write of string " ^ s ^ " to file " ^ f ^ "\n")
```

Le système de calcul des types de  $\mathbf{F}^*$  est beaucoup plus riche. Il prend en compte :

- type dépendant avec raffinement
- effets
- spécification = précondition + postcondition

Inférence de types beaucoup plus précise, prenant en compte les effets organisés en treillis :

- Tot “totale” pas d’effet de bord, termine dans tous les cas
- Dv “diverge”, peut ne pas terminer
- ST “state” = Dv ou I/O ou allocation
- Exn “exception” = Dv ou exception
- ML = tous les effets possibles (pire des cas)

```
val is_positive : Z → Tot bool
let is_positive i = i > 0

val max : Z → Z → Tot Z
let max i1 i2 = if i1 > i2 then i1 else i2
```

De même qu’on a des sous-types  $\text{int}\{x>0\} <: \text{int}$ , on a des sous-effets  $\text{Tot} <: \text{Dv}$  L’utilisateur peut aussi définir des effets à l’aide de structures appelées monades.



Supposons le prototype d'une fonction qui vérifierait si un élément se trouve dans une liste :

```
val mem: `a → list `a → Tot bool
```

En fait, cette fonction qui doit marcher pour une liste avec un type générique `a fonctionne en fait seulement si le type possède une égalité décidable.

#a : eqtype

est ici un raccourci pour un type générique raffiné

#a:Type{hasEq a}

```
val mem: #a:eqtype → a → list a → Tot bool
let rec mem #a x xs =
  match xs with
  | [] → false
  | hd :: tl → hd = x || mem x tl
```

On peut aider le système qui vérifie la bonne terminaison d'une fonction récursive à l'aide de clause expliquant que les arguments "décroissent" pour un ordre :

```
val ackermann_swap: n:N → m:N → Tot N (decreases %[m;n])
let rec ackermann_swap n m =
  if m=0 then n + 1
  else if n = 0 then ackermann_swap 1 (m - 1)
  else ackermann_swap (ackermann_swap (n - 1) m)
    (m - 1)
```

Exemple d'ordres :

- ordre des entiers naturels
- ordre lexicographique
- ordre des sous-terme (une sous liste d'une liste par exemple)

## Spécification du tri sort :

```
val sorted: list Z → Tot bool
let rec sorted l = match l with
| [] → true
| [x] → true
| x::y::xs → x ≤ y && sorted (y::xs)

val partition: (α → Tot bool) → list α → Tot (list α × list α)
let rec partition f = function
| [] → [], []
| hd::tl →
  let l1, l2 = partition f tl in
  if f hd
  then hd::l1, l2
  else l1, hd::l2

val sorted_concat_lemma: l1:list Z{sorted l1}
  → l2:list Z{sorted l2}
  → pivot:Z
  → Lemma (requires ((∀ y. mem y l1 ⇒ not (pivot ≤ y))
    ∧ (∀ y. mem y l2 ⇒ pivot ≤ y))
    (ensures (sorted (append l1 (pivot::l2))))
    [SMTPat (sorted (append l1 (pivot::l2)))]))

let rec sorted_concat_lemma l1 l2 pivot = match l1 with
| [] → ()
| hd::tl → sorted_concat_lemma tl l2 pivot

val sort: l:list Z → Tot (m:list Z{sorted m ∧ (∀ i. mem i l = mem i m)})
  (decreases (length l))

let rec sort l = match l with
| [] → []
| pivot::tl →
  let hi, lo = partition (λ j → pivot ≤ j) tl in
  let m = append (sort lo) (pivot::sort hi) in
  assert (∀ i. mem i (pivot::sort hi) = mem i (append [pivot] (sort hi)));
  m
```

## Program verification: Shall the twain ever meet?

### Interactive proof assistants

Coq, CompCert,  
Isabelle, seL4,  
Agda, Bedrock,  
Lean, 4 colors

*air*

*gap*

### Semi-automated verifiers of imperative programs

Dafny, Verve,  
FramaC, IronClad,  
Why3, miTLS  
Vale

- **In the left corner:** Very expressive dependently-typed logics, but only purely functional programming
- **In the right:** effectful programming, SMT-based automation, but only first-order logic

•  $F^*$  est un langage OCaml-friendly pour prouver des vrais programmes *stateful* (état de la mémoire, variable globale, etc...) et pas juste le cas facile purement fonctionnel.

• le compilateur KremLin permet d'extraire vers du C lisible un sous ensemble  $Low^* \subset F^*$

⇒  $F^*$  propose donc une alternative à Frama-C pour développer des logiciels sûrs et performants

4 ) HA CL \*

```

1 (* Field types and parameters *)
2 let prime = pow2 255 - 19
3 type elem : Type0 = e:int{e ≥ 0 ∧ e < prime}
4 let fadd e1 e2 = (e1 + e2) % prime
5 let fsub e1 e2 = (e1 - e2) % prime
6 let fmul e1 e2 = (e1 * e2) % prime
7 let zero : elem = 0
8 let one : elem = 1

```

Fig. 1. Spécification du corps  $\mathbb{Z}/(2^{255} - 19)\mathbb{Z}$  en F\*

```

1 let lbytes (l:nat) = b:seq FStar.UInt8.t{length b = l}
2
3 let scalarmult (k:lbytes 32) (u:lbytes 32) : Tot (lbytes 32) =
4   let k = decodeScalar25519 k in
5   let u = decodePoint u in
6   let res = montgomery_ladder u k in
7   encodePoint res

```

Fig. 2. Spécification de la fonction d'exponentiation de Curve25519

```

1 type uint8_p = buffer Hacl.UInt8.t
2
3 val crypto_scalarmult:
4   output:uint8_p{length output = 32} →
5   secret:uint8_p{length secret = 32} →
6   point:uint8_p{length point = 32} →
7   Stack unit
8   (requires (λ h → live h output ∧ live h secret ∧ live h point))
9   (ensures (λ h0 res h1 → live h1 output ∧ modifies_1 output h0 h1 ∧
10    live h0 output ∧ live h0 secret ∧ live h0 point ∧
11    h1.[output] == Spec.scalarmult h0.[secret] h0.[point]))

```

Fig. 3. Signature Low\* de la fonction d'exponentiation de Curve25519

```
void Hacl_Curve25519_crypto_scalarmult(uint8_t *output, uint8_t *secret, uint8_t *point);
```

Fig. 4. Prototype C de la fonction d'exponentiation de Curve25519 dans HACL\*

Algorithm	HACL*	OpenSSL	libsodium	TweetNaCl	OpenSSL (asm)
SHA-256	13.43	16.11	12.00	-	7.77
SHA-512	8.09	10.34	8.06	12.46	5.28
Salsa20	6.26	-	8.41	15.28	-
ChaCha20	6.37 (ref) 2.87 (vec)	7.84	6.96	-	1.24
Poly1305	2.19	2.16	2.48	32.65	0.67
Curve25519	154,580	358,764	162,184	2,108,716	-
Ed25519 sign	63.80	-	24.88	286.25	-
Ed25519 verify	57.42	-	32.27	536.27	-
AEAD	8.56 (ref) 5.05 (vec)	8.55	9.60	-	2.00
SecretBox	8.23	-	11.03	47.75	-
Box	21.24	-	21.04	148.79	-

**Tableau 1.** Intel64-GCC : Comparaison de performance en cycles/octet pour un Intel(R) Xeon(R) CPU E5-1630 v4 @ 3.70GHz opérant sous Debian Linux 4.8.15 64-bit. Toutes les mesures (sauf Curve25519) sont basées sur des messages de 16KB ; pour Curve25519 le nombre indiqué correspondance au nombre de cycles CPU pour une seule exponentiation modulaire ECDH. L'ensemble du code a été compilé avec GCC 6.3. La version de OpenSSL est 1.1.1-dev (avec l'option `no-asm`) ; celle de Libsodium est 1.0.12-stable (avec l'option `--disable-asm`) et celle de TweetNaCl : 20140427.

⇒ meilleur résultat non vectorisé (portable)

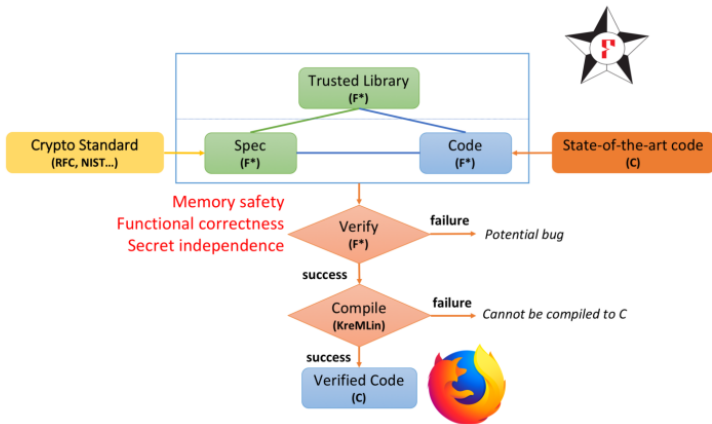


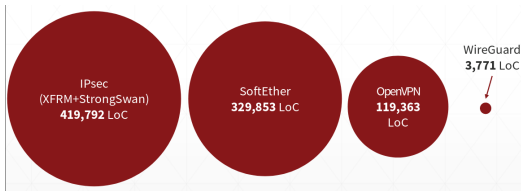
Fig. 6. Méthodologie de verification de HACL\*



## 5 ) Wireguard

## Wireguard ?

- tunnel VPN layer 3 pour IPv4 et IPv6
- dans le noyau Linux
- designé pour la performance et la simplicité
- basé sur UDP : paquet encrypté puis encapsulé dans un paquet UDP
- soutenu par Linus Torvalds
- utilise les primitives crypto de (notamment) HACL\*



### Server Config

```
[Interface]
PrivateKey =
yAnz5TF+LXJTe14tj13zLMNq+hd2rYU
IgJ8g83fBmk=
ListenPort = 41414

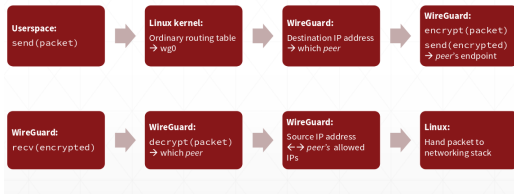
[Peer]
PublicKey =
xTIBA5rboUvnh4htodj6e697QjLERt1
NAB4mZqp8Dg=
AllowedIPs =
10.192.122.3/32,10.192.124.1/24

[Peer]
PublicKey =
TrMvSoP4jYQlY6RIzBgbssQqY3vxI2P1
+y71LOWX0=
AllowedIPs =
10.192.122.4/32,192.168.0.0/16
```

### Client Config

```
[Interface]
PrivateKey =
gI6EdUSYvn8ugXot8QQD6Yc+Jy1zXihp
3GInSWRfWGE=
ListenPort = 21841

[Peer]
PublicKey =
HIgo9xNzJMwLKASSh1TqIybxZ0U3wGL1
UeJ1PKf8Ykw=
Endpoint = 192.95.5.69:41414
AllowedIPs = 0.0.0.0/0
```



## WireGuard Didn't Make it To The Mainline Linux Kernel This Cycle

Written by [Michael Larabel](#) in [Linux Kernel](#) on 3 November 2018 at 07:00 AM EDT. [10 Comments](#)



While there are a lot of great new features, hardware support improvements, and other changes with the [Linux 4.20](#) development cycle, not found in this mainline kernel is the long-awaited WireGuard functionality for an in-kernel secure VPN tunnel.

[WireGuard](#) didn't make it into net-next and no pull request otherwise was issued for getting this big ticket networking feature into the next version of the Linux kernel. The code [continues to be improved upon](#) but looks like it came up just short of making it into this current development cycle.

With WireGuard also comes the [new Zinc crypto API](#) for the Linux kernel, which wasn't marked for inclusion either this cycle.



It's a pity as even Linus Torvalds himself expressed hope earlier this year of [merging WireGuard sooner rather than later](#). WireGuard has received a lot of other praise from upstream developers too and [even a recommendation from a US senator](#).

At least there are out-of-tree (DKMS) packages for WireGuard available for many distributions and other resources to get started even without mainline kernel support. Those wanting to learn more about WireGuard this weekend can do so at [WireGuard.com](#).

Hopefully WireGuard will be merged for the next kernel cycle and by then even in better shape for being the flagship Linux VPN solution.