Devising formal methods
• clear attacker models
• program verification tools
• bug finding techniques

Solving security problems
• programming securely with cryptography
• stopping web attacks
• building secure systems

Developing practical tools and systems
• F*, miTLS, HACL*, ProVerif, CryptoVerif, ProScript, CryptoCat, QuickChick, ...
PhD Students (4)
Benjamin Beurdeuche
Nadim Kobeissi
Kenji Maillard
Jean Karim Zinzindohoue

Interns (4)
Victor Dumitrescu
Guglielmo Fachini
Natalia Kulatova
Théo Laurent

PostDocs (2)
Danel Ahman
Marco Stronati

Visitors (3)
David Baelde (ENS Cachan)
Ana Nora Evans (Univ of Virginia)
David Evans (Univ of Virginia)

Researchers (6)
Karthik Bhargavan
Bruno Blanchet
Harry Halpin
Cătălin Hriţcu
Graham Steel
Christine Rizkallah

Engineers (2)
Tomer Libal
Marc Sylvestre

Current team
Diverse and international
Our working language is English

Collaborators at Microsoft Research, UPenn, MIT, Northeastern, Portland State, IMDEA, Imperial, UCL, ...
Use formal methods to achieve security of critical software

- HTTPS stack (miTLS, Everest)
- Modern cryptographic library (HACL*)
- Secure messaging app (CryptoCat, NEXTLEAP)
- Web browser core (CIRCUS)
- Compilers & monitors (Micro-Policies, SECOMP)
- TCP/IP network stack...
Tools for analyzing abstract models of crypto protocols

- **ProVerif** – symbolic model (Dolev-Yao)
  - fully automatic, efficient, precise, produces attack traces
  - wide range of crypto primitives and properties

- **CryptoVerif** – computational model
  - semi-automatic: sequence of crypto games
  - exact security: bound on attack probability

- Recent case studies: TLS 1.2 & 1.3, Signal, ARINC823
  - upcoming TLS 1.3: big redesign, new hope for verification
From verifying protocol models to actual implementations

- Protocol models capture core behavior: succinct, abstract, high-level
  - great for finding logical flaws [3Shake] and incorrect use of crypto [Lucky13] early in the protocol design
  - e.g. TLS 1.2 & 1.3 in ~1000 lines of ProVerif (best paper at Oakland'17)

- Protocol implementations
  - large software projects: interoperable, efficient
  - concrete packet formats, multiple protocol modes
  - support legacy ciphersuites, complex APIs, composable subprotocols
  - more attacks: message parsing [HeartBleed], state machine [FREAK]

- more attacks:
- Verified reference implementation of TLS 1.2 & 1.3
- Microsoft Research and Inria
- Built on top of our HACL* crypto library – verified and faster than OpenSSL libcrypto and Sodium
- Towards a verified HTTPS stack (Project Everest)
HTTPS ecosystem critical, complex

TLS
X.509
HTTPS
RSA
SHA
ECDH
Network buffers
Untrusted network (TCP, UDP, …)

Crypto Algorithms
4Q

Services & Applications
ASN.1
Certification
Authority

Servers
Apache
IIS
Skype
Nginx
Edge
cURL
WebKit
• 20 years of attacks & fixes
  w ffbp z ffb p
  p ffb ffb j ffb
  ffb j j ffb j
  ffb p pp a j ffb ffb ffb a p
  j ffb l ffb
  p ffb w j a p
  ffb w
  ffb

• Mainstream implementations
  ffb l ffb
  j l ffb ffb ffb p l
Everest stack verified with

- Functional programming language—like OCaml, F#, Haskell, …
- extracted to OCaml or F# by default
- subset of F* compiled to efficient C code

- Semi-automated verification using SMT—like Dafny, FramaC, Why3, …

- Interactive verification using dependent types—like Coq, Lean, Agda, …

- $j \text{ ffb} \quad 9 \quad \text{ ffb}$
- $\text{ ffb} \quad \text{ ffb} \quad p \quad p \quad 9 \quad a \quad \text{ ffb} \quad w$
- $w \quad \text{ ffb} p \quad \& p \quad j \text{ ffb} \quad p \quad \text{ ffb}j \quad j\text{ ffb} \quad p \quad \text{ ffb}$

- $j \text{ ffb} \quad a \quad l \quad a$

- $j \text{ ffb} p \quad \text{ ffb}$
Is verified code secure in practice?

F*  
C/C++ compiled F* compiled C/C++ ASM compiled ASM

Insecure interoperability

Everest HTTPS

0.000 LOC

Web browser/server 2.000.000+

OK we can verify this

Ooops

Unsafe languages
Secure compilation

• Secure interoperability with lower-level code – component separation, call and return discipline, types, ...

• Dynamic enforcement, but at what cost? – in software, 10x? 100x? 1000x?

• Micro-policies – new tagged hardware architecture – associates large metadata tag to each word – efficiently propagates and checks tags; hw caching – dynamic monitoring: software defined, very flexible, fine-grained (words, instructions), fast …

– … average 10% runtime overhead for complex policies!
Use formal methods to achieve security of critical software

- HTTPS stack (miTLS, Everest)
- Modern cryptographic library (HACL*)
- Secure messaging app (CryptoCat, NEXTLEAP)
- Web browser/server core (CIRCUS)
- Compilers & monitors (Micro-Policies, SECOMP)
- TCP/IP network stack...