Our research

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

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

Developing practical tools and systems
• F*, miTLS, HACL*, ProVerif, CryptoVerif, ProScript, CryptoCat, QuickChick, ...
Finding attacks in TLS

SMACK: State Machine AttaCKs
Implementations of the Transport Layer handle a variety of protocol versions and key exchange methods, prescribe a different message sequence for each server. We address the problem of a machine that can correctly identify and fix these modes.

The Logjam Attack

Tracking the FREAK Attack

The BEAST Wins Again: Why TLS Keeps Failing to Protect HTTP

RISK ASSESSMENT / SECURITY & HACKTIVISM
HTTPS-crippling attack threatens tens of thousands of Web and mail servers

"FREAK" flaw in Android cripples HTTPS crypto protocol
Bug forces millions of sites to use easily broken encryption.
Researchers

Karthik Bhargavan
Bruno Blanchet
Harry Halpin
Cătălin Hrițcu
Graham Steel
Cryptosense
Christine Rizkallah
Current team

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

PhD Students (4)
Benjamin Beurdeuche
Nadim Kobeissi
Kenji Maillard
Jean Karim Zinzindohoue

PostDocs (2)
Danel Ahman
Marco Stronati

Engineers (2)
Tomer Libal
Marc Sylvestre

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

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

Collaborators at Microsoft Research, UPenn, MIT, Northeastern, Portland State, IMDEA, Imperial, UCL, ...

Diverse and international 11 nationalities  Our working language is English
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 phase
  - 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]
• 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
HTTPS ecosystem critical, complex and broken

• 20 years of attacks & fixes
  Buffer overflows
  Incorrect state machines
  Lax certificate parsing
  Weak or poorly implemented crypto
  Side channels

  Informal security goals
  Dangerous APIs
  Flawed standards

• Mainstream implementations
  OpenSSL, SChannel, NSS, ...
  Still patched every month!

Services & Applications

Clients

Servers

Certification Authority

X.509

ASN.1

TLS

HTTP

RSA

SHA

ECD

H

Crypto Algorithms

Network buffers

Untrusted network (TCP, UDP, ...)

Crypto Algorithms

Services & Applications

Edge cURL WebKit Skype IIS Apache Nginx

The Washington Post

‘FREAK’ flaw undermines security for Apple and Google users, researchers discover
Project Everest Goals

**Strong verified security**

**Widespread deployment**

- efficiency
- interoperability
- drop-in replacement for OpenSSL, NSS, ...

![Diagram of network buffers and services/applications]
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, ...
Is verified code secure in practice?

OK we can verify this

Everest HTTPS
30,000 LOC

F*

Unsafe languages

Web browser/server
2,000,000+ LOC

C/C++

ASM

Insecure interoperability

compiled F*

compiled C/C++

compiled ASM

Ooops

Unsafe languages

Web browser/server
2,000,000+ LOC

C/C++

ASM

Insecure interoperability

compiled F*

compiled C/C++

compiled ASM
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 ...