

# Automated Proof Techniques for Cryptographic Assurance

Bruno Blanchet

EPI Prosecco

INRIA Paris

September 2016

# Models of protocols

Two models of security protocols:

- **Symbolic model (Dolev-Yao):**

- Primitives are black boxes. `encrypt`
- Messages are terms on these primitives. `encrypt(Hello, k)`
- The adversary is restricted to apply only those primitives.

This model **facilitates the automation** of proofs.

- **Computational model:**

- Messages are bitstrings. `01100100`
- Primitives are functions on bitstrings. `encrypt(011, 100100) = 111`
- The adversary is any (probabilistic polynomial-time) Turing machine.

This model is **more realistic**: the adversary can apply any algorithm.

# Models of protocols

Two models of security protocols:

- **Symbolic model (Dolev-Yao):**

- Primitives are black boxes.
- Messages are terms on these primitives.
- The adversary is restricted to apply only those primitives.

ProVerif

encrypt

encrypt(*Hello*, *k*)

This model **facilitates the automation** of proofs.

- **Computational model:**

- Messages are bitstrings.
- Primitives are functions on bitstrings.
- The adversary is any (probabilistic polynomial-time) Turing machine.

CryptoVerif

01100100

encrypt(011, 100100) = 111

This model is **more realistic**: the adversary can apply any algorithm.

# Protocol verification in the symbolic model

Security protocols are **infinite state**:

- The attacker can create messages of **unbounded size**.
- **Unbounded number of sessions** of the protocol.

Solutions:

- Bound the state space arbitrarily:  
exhaustive exploration (model-checking, ... );  
find attacks but not prove security.
- Bound the number of sessions:  
the insecurity is **NP-complete** (with reasonable assumptions).
- Unbounded case:  
the problem is **undecidable**.

# Solutions to undecidability

To solve an undecidable problem, we can

- Use **approximations**, abstraction.
- **Terminate** on a **restricted** class.
- Rely on user interaction or annotations.

In ProVerif, we do the first two, using a very precise abstraction by **Horn clauses**.

# Features of ProVerif

- **Fully automatic.**
- Works for **unbounded** number of sessions and message space.
- Handles a **wide range** of cryptographic primitives, defined by rewrite rules or equations.
- Handles various **security properties**: secrecy, authentication, some equivalences.
- Limitations:
  - Does not always terminate. However, **efficient** in practice: small examples verified in less than 0.1 s; complex ones in minutes.
  - May answer “I don’t know” (false attack). However, **very precise** in practice: no false attack in our tests for secrecy and authentication.

# ProVerif

Protocol:  
Pi calculus + cryptography  
Primitives: rewrite rules, equations

Properties to prove:  
Secrecy, authentication,  
process equivalences

Automatic translator

Horn clauses

Derivability queries

Resolution with selection

Non-derivable: the property is true

Derivation

Attack: the property is false

False attack: I don't know

# The Horn clause representation

The main predicate:

$\text{attacker}(M)$  means “the attacker may have  $M$ ”.

Thanks to this predicate, we can model actions of the adversary:

Example: Shared-key encryption and decryption

$$\text{attacker}(m) \wedge \text{attacker}(k) \rightarrow \text{attacker}(\text{encrypt}(m, k))$$
$$\text{attacker}(\text{encrypt}(m, k)) \wedge \text{attacker}(k) \rightarrow \text{attacker}(m)$$

and of the protocol participants:

Example:  $A$  receives  $M$  and replies with  $M'$

$$\text{attacker}(M) \rightarrow \text{attacker}(M')$$



# CryptoVerif

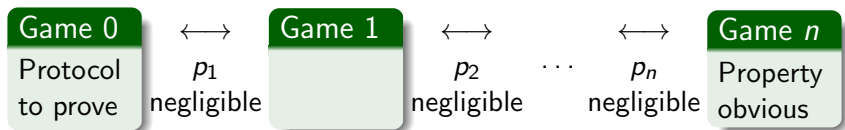
The protocol verifier CryptoVerif:

- works directly in the **computational model**.
- proves **secrecy** and **correspondence** (authentication) properties.
- provides a **generic** method for specifying properties of **cryptographic primitives** which handles MACs (message authentication codes), symmetric encryption, public-key encryption, signatures, hash functions, Diffie-Hellman key agreements, ...
- works for  **$N$  sessions** (polynomial in the security parameter).
- gives a bound on the **probability** of an attack (exact security).
- has **automatic** and **manual** modes.

# Produced proofs

As in Shoup's and Bellare&Rogaway's method, the proof is a **sequence of games**:

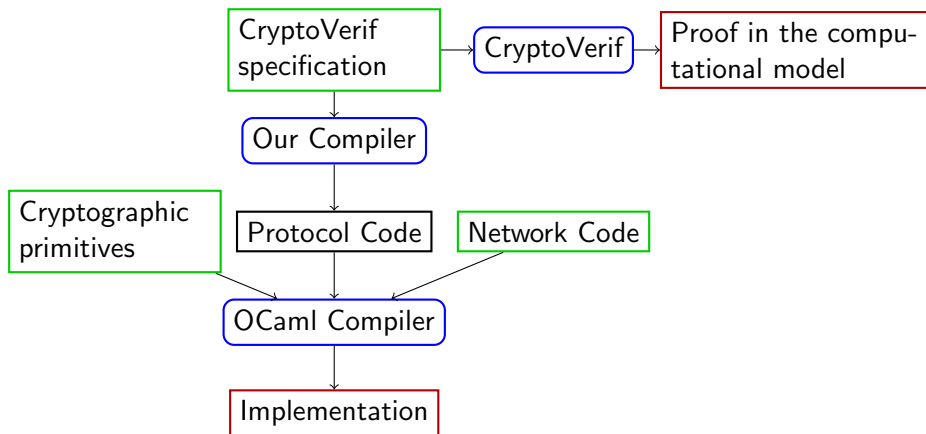
- The first game is the **real protocol**.
- One goes from one game to the next by syntactic transformations or by applying the definition of security of a cryptographic primitive. The difference of probability between consecutive games is negligible.
- The last game is **"ideal"**: the security property is obvious from the form of the game.  
(The advantage of the adversary is usually 0 for this game.)



# Models vs. implementations

- ProVerif and CryptoVerif **automatically** analyze protocol **models**.
- Just that **models are very abstract**:
  - Protocol models may miss implementation attacks.
- Verified models are good
  - ... but **verified implementations** are much better!

# Generation of implementations (David Cadé)



Caption: **Tool** **Input** **Result**

# Verified implementations with $F^*$ , <https://www.fstar-lang.org/>



- $F^*$  is a new programming language
- ... putting together:
  - **impure functional programming** in ML
    - extracts to OCaml and F#, interoperates
  - the **automation** of SMT-based verification systems
    - like in Why3, Frama-C, Boogie, VCC, Dafny
  - the **expressive power** of interactive proof assistants based on dependent types
    - like in Coq, Agda, or Lean

# miTLS, <http://www.mitls.org/>

- Formally verified reference implementation of TLS 1.2 in F7/F\* (working towards TLS 1.3)
- Written from scratch focusing on verification

miTLS - Home

<https://www.mitls.org:2443/wsgi/home>

miTLS Home Publications Download Browse TLS Attacks People

**miTLS**  
A verified reference TLS implementation

This page is served using the miTLS demo HTTPS server. ([Go back to production server](#))

- **ciphersuite:** TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA,
- **compression:** NullCompression,
- **version:** TLS\_1p2

# Lead to the discovery of many attacks in TLS implementations

MI T.L.S. **miTLS** **3SHAKE** **SMACK** **VHC**

**SMACK Introduction** **Threat Model** **SKIP-TLS Attack** **FREAK Att**

## SMACK: State Machine Attacks



Implementations of the Transport Layer Security (TLS) protocol handle a variety of protocol versions, cipher suites, key exchange methods, and message sequence numbers. We address the problem of designing a robust state machine that can correctly multiplex these modes.

## Triple Handshakes Considered Breaking and Fixing Authentication

March 4, 2014

Introduction	TLS Weaknesses	Triple Handshakes
Countermeasures	Disclosure	Other Attacks

Slides from the TLS WG session at IETF89 and our proposed Internet-Draft. Our research paper with more details on the attacks (see Sections 4-7) is available on the IETF website.

## The Logjam Attack

Warning! Your web browser is vulnerable to Logjam and can be tricked into using weak encryption. Update your browser.

Diffie-Hellman key exchange is a popular cryptographic algorithm that allows Internet protocol shared key and negotiate a secure connection. It is fundamental to many protocols including HTTP, SMTPS, and protocols that rely on TLS.

We have uncovered several weaknesses in how Diffie-Hellman key exchange has been deployed.

## Tracking the FREAK Attack

Good News! Your browser appears to be safe from the FREAK attack.

On Tuesday, March 3, 2015, researchers announced a new SSL/TLS vulnerability called the FREAK attack. It allows an attacker to intercept HTTPS connections between vulnerable clients and servers and force them to use weakened encryption. This is a serious security issue because it allows attackers to track the information you send over the internet.

The FREAK attack disclosure was made by researchers at the University of Michigan, including researchers from the team that can be contacted for additional details at this Washington

ars technica

MAIN MENU MY STORIES: 24 FORUMS SUBSCRIBE JOBS ARS CONSENTMENT

Ars Technica has arrived in Europe. Check it out!

## RISK ASSESSMENT / SECURITY & HACKTIVISM

### “FREAK” flaw in Android and Apple devices cripples HTTPS crypto protection

Bug forces millions of sites to use easily breakable key once thought to be dead.

by Dan Goodin - Mar 3, 2015 10:05am CET

## The BEAST Wins Again: Weaknesses in TLS

### Documents

- PDF of slides
- summary of briefing for non-experts
- paper: Virtual Host Conf
- paper: Triple Handshake

### Exploit videos

Disclaimer: The goal of these videos is to have a strong impact. The attack is not a real attack and is not a real exploit. We are also thankful to the discoverers, who are dedicated to the public good.

THE WALL STREET JOURNAL

Home World U.S. Politics Economy Business Tech Markets Opinion Arts Life Real Estate

## New Computer Bug Exposes Broad Security Flaws

Fix for Logjam bug could make more than 20,000 websites unreachable



ars technica

MAIN MENU MY STORIES: 24 FORUMS SUBSCRIBE JOBS ARS CONSENTMENT

Ars Technica has arrived in Europe. Check it out!

## RISK ASSESSMENT / SECURITY & HACKTIVISM

### HTTPS-cripping attack threatens tens of thousands of Web and mail servers

Diffie-Hellman downgrade weakness allows attackers to intercept encrypted data.

by Dan Goodin - Mar 20, 2015 7:41am CET



## FREAK Attack Threatens SSL Clients

Posted by Soutskill on Tuesday March 03, 2015 @04:29PM from the another-day-another-vuln dept.

msm1267 writes:

For the nth time in the last couple of years, security experts are warning about ;

[Internet-scale vulnerability, this time in some popular SSL clients](#). The flaw allows an attacker to force clients to downgrade to weakened ciphers and break their supposedly encrypted communications through a man-in-the-middle attack.

the discoverers, who are dedicated to the public good.

the discoverers, who are dedicated to the public good.

the discoverers, who are dedicated to the public good.

# Conclusion

Verified security protocols at several levels:

- Specifications:
  - In the **symbolic model**: ProVerif  
Available at <http://proverif.inria.fr/>
  - In the **computational model**: CryptoVerif  
Available at <http://cryptoverif.inria.fr/>
- Implementations:
  - **Generation** of implementations from specifications: CryptoVerif
  - **Direct verification** of implementations: F\*.  
Available at <https://www.fstar-lang.org/>

See <http://prosecco.inria.fr/>