TESTING CRYPTOGRAPHIC PROTOCOL IMPLEMENTATIONS

Verifying crypto protocols

- Lots of formal methods
  - Good representative: Blanchet's ProVerif
    - Mainly for its good spec language
    - Almost always gives an answer

- Not much on verifying implementations
  - fs2pv, Csur

- Running example today: TLS (the thing that runs when you browse https://...

A long history

- – 1994 – Netscape's Secure Sockets Layer (SSL)
- – 1994 – SSL2 (known attacks)
- – 1995 – SSL3 (fixed them)
- – 1999 – IETF's TLS1.0 (RFC2246, ≈SSL3)
- – 2006 – TLS1.1 (RFC4346)
- – 2008 – TLS1.2 (RFC5246)

- • Provides a layer between TCP and Application (in the TCP/IP model)
- – Itself a layered protocol: Handshake over Record
- • *Record (sub)protocol*
- – provides a private and reliable connection
- • *Handshake (sub)protocol*
- – authenticates one or both parties, negotiates security parameters
- – establishes secret connection keys for the Record protocol
- • *Resumption (sub)protocol*
- – abbreviated version of Handshake: generates connection keys from previous handshake

Transport layer security (TLS)

- Uses several cryptographic primitives
  - Asymmetric encryption (eg, RSA)
  - Symmetric encryption (eg, AES)
  - Hash functions (eg, SHA1, MD5)
  - MAC function (HMAC)

- Gathered in “ciphersuites”, eg
  TLS_RSA_WITH_AES_128_CBC_SHA ,
  TLS_DHE_DSS_WITH_DES_CBC_SHA

TLS (generic)

Client		Server
ClientHello	-------->	
		ServerHello
		[Certificate]
		[ServerKeyExchange]
	<--------	[CertificateRequest]
		ServerHelloDone
[Certificate]		
ClientKeyExchange		
[CertificateVerify]		
[ChangeCipherSpec]		
Finished	-------->	
		[ChangeCipherSpec]
	<--------	Finished
Application Data	<------->	Application Data

Handshake (RSA, client anonymous)

Client		Server
ClientHello	-------->	
(version, ciphers, nonce)		ServerHello
		(chosen version & cipher=RSA + nonce)
		Certificate
		ServerHelloDone
	<--------	
ClientKeyExchange		
(encrypts pre-master-secret w/servers pk)		
ChangeCipherSpec		
Client Finished	-------->	(master secret computed from nonces
(all the previous msgs hashed)		and pms), split in 6 keys:
		cek,sek,cmk,smk,civ,siv)
	←-------	Server Finished

TLS bugs / attacks

- Bugs and attacks keep being found!
  - This year a couple
- Errors:
  - “Bugs” -> crash the client or server, execute code,…
  - “Attacks” -> everything looks fine but the goals are violated
- 3 kinds:
  - Message-flow
  - Implementation
  - Cryptographic

TLS message-flow attacks

- Ciphersuite rollback (ssl 2):
  - Change the negotiated ciphersuite to the weakest
  - Hello messages were not included in the finished messages! Hence unauthenticated
- Same issue in resumption, it didn’t include finished messages

TLS implementation bugs 1/2

- From Advisory 2002:
  - 1. The client master key in SSL2 could be oversized and overrun a buffer.
  - 2. The session ID supplied to a client in SSL3 could be oversized and overrun a buffer.
  - 3. The master key supplied to an SSL3 server could be oversized and overrun a stack-based buffer.

TLS implementation bugs 2/2

- From Advisory 2009:
  - "Several functions inside OpenSSL incorrectly checked the result after calling the EVP_VerifyFinal function, allowing a malformed signature to be treated as a good signature rather than as an error."

```
ret=RSA_verify(NID_md5_sha1, buf,36, buf2, rsa_num,
    rsa_key[j]);
- if (ret == 0)        <- ERROR
+ if (ret <= 0)        ←- PATCH
{ BIO_printf(bio_err, "RSA verify failure\n");
```

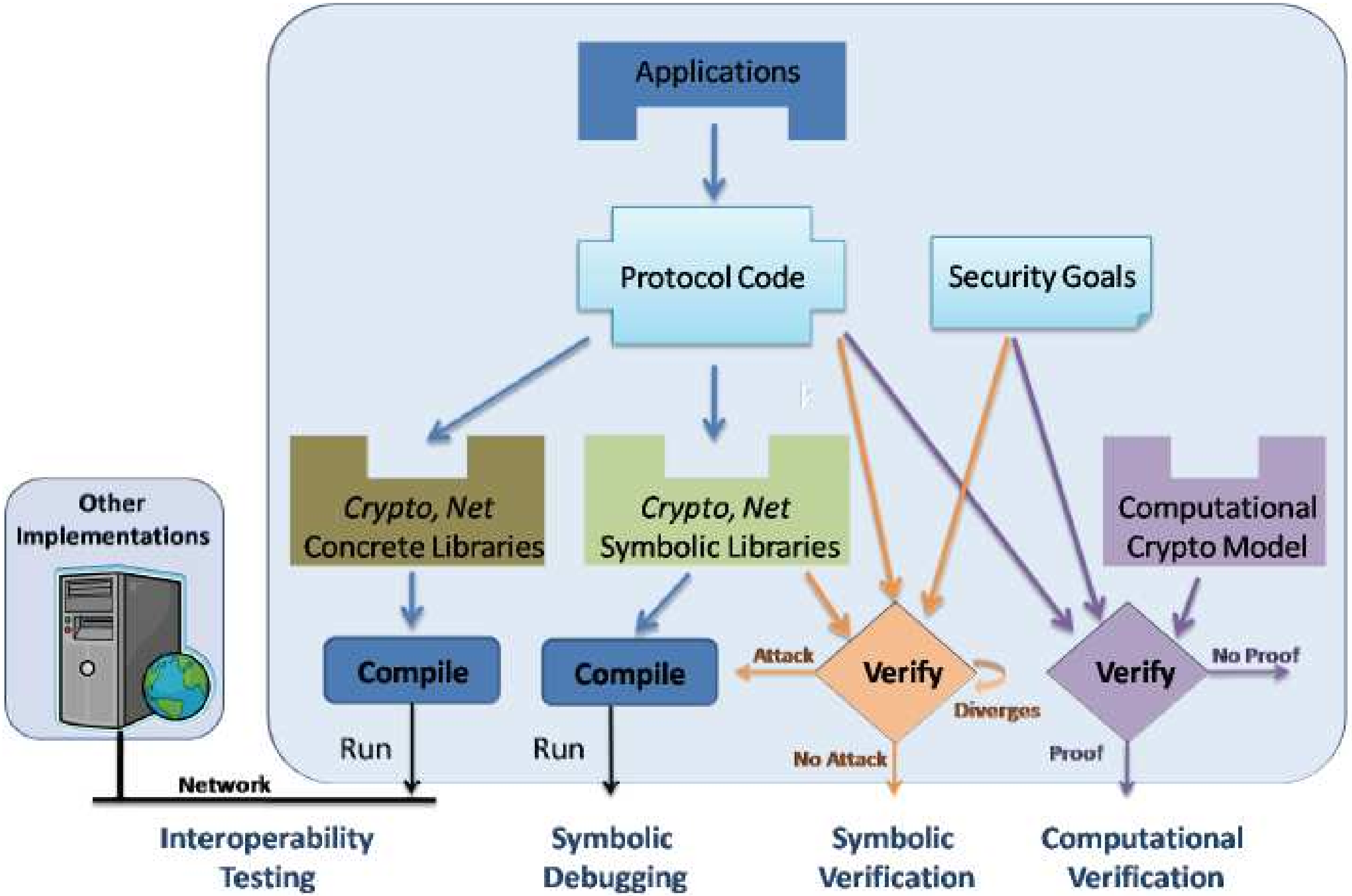
TLS cryptographic attacks

- Attacks more on the primitives
  - Predicting randomness
  - Timing attacks
  - Using alert messages as oracles in RSA mode
  - ….

How to verify TLS?

- Translates to 3 separate problems
- "How to verify *an implementation* of TLS {symbolically,cryptographically,implementation-wise}
- Mostly manual attempts
- Some work in verification for "symbolically"
  - Rest of this talk:
    - Will show earlier work for "symbolically"
    - This work's idea: put symbolic and impl. together

Verifying protocol implementations, Cambridge-Paris 's style



Demo

Results from that work:

All properties are automatically proved

- – But after a lot of hand-tuning on the source code
- (otherwise ProVerif runs out of memory or does not finish)
- – Final ProVerif script of *Handshake+Resumption+Record still large (2100LOC)*
- – Proving Record/Handshake separately is much easier (but less precise)
- • Experimental details:

Part of protocol verified	# of queries	PV running time	Memory used
Handshake (auth. queries)	2	16sec	60MB
Handshake (secr. queries)	2	10sec	80MB
Handshake + Resumption (resumption auth. queries)	2	4min	460MB
Handshake + Resumption + Record (record auth. queries)	2	6min	700MB
Handshake + Resumption + Record	8	2hours	1.7GB

+ and -

- +:
  - Model faithfully follows implementation
  - Automatic

- -:
  - Derived model unmanageable, too complex (resource hog)
    - → so, no spec, one believes in it because it interoperates
    - Also true for Csur:
      - “a running 229 line implementation (excluding included les) of A's role in the Needham-Schroeder protocol results in a set of 459 clauses”
  - Works only for (a subset of) F#
    - No legacy code

Verifying protocol implementations, Cordoba‘s style

- Instead of going from implementations to spec, go from spec to implementations
- Derive test cases from spec, try them on (any!) implementation
- -:
  - Spec writing is manual (but for some this is a +)
  - Can’t prove absence of impl. bugs (testing karma)
- +:
  - Spec readable and short, quick verification
  - Works on any implementation
- The role of testing is to *gain confidence that we’re verifying the correct spec*

How it works?

- loco’s style testing
- Find all execution interleavings i
- For each i, traverse it maintaining the knowledge of “known” and “unknown” terms
  - “known” terms come from eavesdropping
  - “unknown” terms are used by the procs but not immediately known
  - Accept each output made by the processes, “learn” as much as possible
    - may be delayed from previous “lets”
  - For each input made by the processes, branch new tests for each received subterm
    - Change size, change msg, …
  - Detect expected results and check conformance

Demo

The future

- If bugs_found -> JACM
- Elsif old_bugs_found -> JAIIO
- Else FAMAF_TR

Other things to try

- Complement with some white-box testing
  - Csur? Why tool?
  - Q: given that impl bugs (like buffer overflows) are sort of independent, why not check them with another tool?
    - Eg, Astree?
    - Best answer so far: this technique is more to check conformance with the spec; should be complementary with those
- Exhaustive coverage of protocols
  - TLS: Apache, openssl, gnutls, all browsers
  - Other prots: DNSSEC, openssh, ipsec,…