Formal Methods and Tools for Distributed Systems

Thomas Ball
Microsoft

http://research.microsoft.com/~tball
Outline

• 20 Years at Microsoft (1999-present)

• The great work of others at Microsoft
20 Years at Microsoft

*From EULA to SLA*

*From Bugs and Bounties to Cyberweapons*

*From Spec to Spec+Check*

*From Closed to Open*
From EULA (1) to SLA

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

EULA
Software

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration
The GPL

11. **EXCLUSION OF IMPLIED WARRANTIES.** TO THE EXTENT PERMITTED BY APPLICABLE LAW, MICROSOFT AND ITS LICENSORS PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. **LIMITATION OF LIABILITY.** UNDER NO CIRCUMSTANCES AND IN NO EVENT SHALL MICROSOFT OR ITS LICENSORS BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF MICROSOFT OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
From EULA (1) to SLA

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

Software

EULA

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

System administration
From EULA to SLA (2)

Programs, Data, Users

Compute, Storage, Networking, Backups, Hdw/Sft updates, ...
System administration

Azure

Programs, Data, Users
Cloud Scale..
Cloud Scale....
Service Level Agreement (SLA)

“For all Virtual Machines that have two or more instances deployed in the same Availability Set, we guarantee you will have Virtual Machine Connectivity to at least one instance at least 99.95% of the time.”

<table>
<thead>
<tr>
<th>MONTHLY UPTIME PERCENTAGE</th>
<th>SERVICE CREDIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 99.95%</td>
<td>10%</td>
</tr>
<tr>
<td>&lt; 99%</td>
<td>25%</td>
</tr>
<tr>
<td>&lt; 95%</td>
<td>100%</td>
</tr>
</tbody>
</table>

https://azure.microsoft.com/support/legal sla/virtual-machines/v1_8/
From Bugs and Bounties to Cyberweapons

Bugs... because there are so many more ways for things to go wrong than there are for them to go right.
Bugs (2001): Nimda

https://en.wikipedia.org/wiki/Nimda
https://www.zdnet.com/article/nimda-rampage-starts-to-slow/

Availability: Our products should always be available when our customers need them. System outages should become a thing of the past because of a software architecture that supports redundancy and automatic recovery. ...

Security: The data our software and services store on behalf of our customers should be protected from harm and used or modified only in appropriate ways. ...

Privacy: Users should be in control of how their data is used. Policies for information use should be clear to the user. Users should be in control of when and if they receive information to make best use of their time. ...

SDLC Timeline

**The perfect storm**
- Growth of home PC's
- Rise of malicious software
- Increasing privacy concerns
- Internet use expansion

**SDL ramp up**
- Bill Gates' TwC memo
- Microsoft security push
- Microsoft SDL released
- SDL becomes mandatory policy at Microsoft
- Windows XP SP2 and Windows Server 2003 launched with security emphasis

**Setting a new bar**
- Windows Vista and Office 2007 fully integrate the SDL
- SDL released to public
- Data Execution Prevention (DEP) & Address Space Layout Randomization (ASLR) introduced as features
- Threat Modeling Tool

**Collaboration**
- Microsoft joins SAFECode
- Microsoft Establish SDL Pro Network
- Defense Information Systems Agency (DISA) & National Institute Standards and Technology (NIST) specify featured in the SDL
- Microsoft collaborates with Adobe and Cisco on SDL practices
- SDL revised under the Creative Commons License

**Selective tooling and Automation**
- Additional resources dedicated to address projected growth in Mobile app downloads
- Industry-wide acceptance of practices aligned with SDL
- Adaption of SDL to new technologies and changes in the threat landscape
- Increased industry resources to enable global secure development adoption

https://www.microsoft.com/en-us/securityengineering/sdl/about
“These produce wrong results. The first example does so only on 32 bit, the other three also on 64 bit.”

“I believe this affects both the SSE2 and AVX2 code. It does seem to be dependent on this input pattern.”

“I'm probably going to write something to generate random inputs and stress all your other poly1305 code paths against a reference implementation.”
The Impact of One Bug

“The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet.”

http://heartbleed.com/
1995
On Oct. 10, 1995, Netscape launched the first bug bounty program offering cash rewards for their Netscape Navigator 2.0 Beta.

2002
In 2002, iDefense’s iVulnerability Contest offered rewards for reporting vulnerabilities in softwares.

2004
Mozilla Foundation launches bug bounty program for identifying critical vulnerabilities in Firefox offering rewards up to $500.

2005
iDefense competitor TippingPoint launched another “middleman” program called the Xero Day Initiative connecting security researchers with vendors.

2007
PWN2OWN contest, a hunt for security bugs in Macs OSX launched. The contest was held within a limited time frame, with $10,000 reward provided by ZDI.

2010
Google kickstarts bug bounty for web applications similar to Mozilla. Mozilla expands into web applications and Barracuda Networks and others also launch their programs.

2011
Facebook follows Google to launch Whitehat program with no upper limit on rewards and a minimum reward of $500.

2015
Rewards increasing, Giants merging hands and the upcoming of popular bug bounty program marketplaces with numerous online opportunities is happening today.

https://blog.cobalt.io/the-history-of-bug-bounty-programs-50def4dcaab3
“Stuxnet is a malicious computer worm, first uncovered in 2010. Thought to have been in development since at least 2005, Stuxnet targets SCADA systems and is believed to be responsible for causing substantial damage to Iran's nuclear program.”

“Stuxnet attacked Windows systems using an unprecedented four zero-day attacks (...)... The number of zero-day exploits used is unusual, as they are highly valued and malware creators do not typically make use of (and thus simultaneously make visible) four different zero-day exploits in the same worm.”

https://en.wikipedia.org/wiki/Stuxnet
From Spec to Spec+Check

Formal Methods

• Mathematical/logical specification of desired (correct) behavior

• Automated/interactive checking of implementation against specification
Correctness Properties

• Memory safety
• No buffer overruns
• Functional correctness
• Termination
• Minimize side-channel leaks
• Cryptographic security
• ...

Verification
Is there a behavior of $S$ that violates $\varphi$?

Counterexample
Unknown / Diverge
Proof

Rice’s Theorem
I can’t decide!

Automatic verification of infinite-state systems

“Formal methods are the future of computer science.” William E. Aitken

Always have been, always will
**Deductive verification**

- **System S**
- **Inductive argument Inv**
- **Property \( \varphi \)**

### Deductive Verification
1. Is \( Inv \) an inductive invariant for \( S \)?
2. Does \( Inv \) entail \( \varphi \)?

- **Counterexample to Induction**
- **Unknown / Diverge**
- **Proof**

Slide from Mooly Sagiv
Inductive invariants

System $S$ is safe if all the reachable states satisfy the property $\varphi = \neg\text{Bad}$.
Inductive invariants

System $S$ is safe if all the reachable states satisfy the property $\varphi = \neg \text{Bad}$

System $S$ is safe iff there exists an inductive invariant $\text{Inv}$:

- $\text{Init} \subseteq \text{Inv}$ (Initiation)
- if $\sigma \in \text{Inv}$ and $\sigma \rightarrow \sigma'$ then $\sigma' \in \text{Inv}$ (Consecution)
- $\text{Inv} \cap \text{Bad} = \emptyset$ (Safety)

Slide from Mooly Sagiv
Logic-based deductive verification

• Represent *Init, →, Bad, Inv* by logical formulas
  • Formula $\iff$ Set of states

• Automated solvers for logical satisfiability made huge progress
  • Propositional logic (SAT) – industrial impact for hardware verification
  • First-order theorem provers
  • Satisfiability modulo theories (SMT) – major trend in software verification
Deductive verification by reductions to First Order Logic

Protocol
Init(V), Tr(V, V')

Loop Invariant Inv(V)

Safety Property \(~Bad(V)\)

Front-End

1) SAT(Init(V) \land \neg Inv(V))?
2) SAT(Inv(V) \land Tr(V, V') \land \neg Inv(V'))?
3) SAT(Inv(X) \land Bad(V))?

First Order SAT Solver

Y

N

Counterexample to Induction (CTI)

Proof

Slide from Mooly Sagiv
Automated Theorem Prover
Open Source (MIT License)
Leonardo de Moura, Nikolaj Bjorner, Christoph Wintersteiger, ...

Boolean Algebra
Bit Vectors
Linear Arithmetic
Floating Point
First-order Axiomitizations
Non-linear, Reals
Sets/Maps/...
Algebraic Data Types

Z3 reasons over a combination of theories

https://github.com/z3prover/z3
https://rise4fun.com/Z3/tutorial
int Puzzle(int x)
{
    int res = x;
    res = res + (res << 10);
    res = res ^ (res >> 6);
    if (x > 0 && res == x + 1)
        throw new Exception("bug");
    return res;
}

(declare-const x (_ BitVec 32))
(assert (bvsgt x #x00000000))
(assert (= (bvadd x #x00000001)
      (bvxor (bvadd x (bvshl x #x0000000A))
            (bvashr (bvadd x (bvshl x #x0000000A)) #x00000006))))
(check-sat)
(get-model)

x = 389306474

https://rise4fun.com/Z3/n6ZB6
Logic/Complexity Classes

Greater Expressiveness

Greater Automation

Practical problems often have **structure** that can be exploited.

Algorithmic advances

Large-scale evaluation and careful engineering
Symbolic Analysis Tools

- SLS, floats
- $\forall Z$: Opt+MaxSMT
- $\mu Z$: Datalog
- Generalized PDR
- Existential Reals
- Model Constructing SAT
- CutSAT: Linear Integer Formulas
- Quantified Bit-Vectors
- Linear Quantifier Elimination
- Model Based Quantifier Instantiation
- Generalized, Efficient Array Decision Procedures
- Engineering DPLL(T) + Saturation
- Effectively Propositional Logic
- Model-based Theory Combination
- Relevancy Propagation
- Efficient E-matching for SMT solvers
Formal Methods: Substantial Progress

Better Tools

• Automated + Interactive Theorem Provers
• Model Checking
• Program Analysis

Application to Real Systems

• Static Driver Verifier (Windows drivers)
• http://compcert.inria.fr/ (C compiler)
• https://sel4.systems/ (OS)
• ...

From Spec to Spec+Check
Open Source: Times have changed!

“We will move to a Chromium-compatible web platform for Microsoft Edge on the desktop” https://blogs.windows.com/

• Microsoft actively contributes to and use open source

• The tools presented in this talk are open source, or have open source equivalents
20 Years at Microsoft

From EULA to SLA

From Bugs and Bounties to Cyberweapons

From Spec to Spec+Check

From Closed to Open
Formal Methods and Tools

- **High-level Specification** (TLA+)
- **Correctness of Cryptography and Protocols** ($F^*$, Ivy, P#)
- **Bug Finding and Verification for C/C++** (SAGE, Corral)
- **Network Verification** (SecGuru)

thinking
programming
testing
verifying
Formal Methods and Tools

- High-level Specification (TLA+)
- Correctness of Cryptography and Protocols (F*, Ivy, P#)
- Bug Finding and Verification for C/C++ (SAGE, Corral)
- Network Verification (SecGuru)

thinking
programming
testing
verifying
SecGuru

Nikolaj Bjørner,
Karthick Jayaraman
A Cloud run by Masters of Complexity

Cloud Explosion

Arcane Systems and Languages

Masters of Complexity
A Cloud Harnessed by Logic/SE
Network Policies: Complexity, Challenge and Opportunity

Several devices, vendors, formats
- Net filters
- Firewalls
- Routers

Challenge in the field
- Do devices enforce policy?
- Ripple effect of policy changes

Arcane
- Low-level configuration files
- Mostly manual effort
- Kept working by “Masters of Complexity”

Human errors > 4 x DOS attacks

Human Errors by Activity

- Config Changes: 74%
- Device hw/sw updates: 13%
- WA Cluster Setup: 13%

74%
Intent = Reality?

Reality?
- Forwarding information base (FIB)
- Access Control Lists (ACL)
- Churn

Intent?
- Network Graph Service (NGS)
- Contracts derived from topology and architecture

Validation
- Continuous verification using local validation

Feedback
- Alerts
- Remediation
Access Control

**Contract:**
DNS ports on DNS servers are accessible from tenant devices over both TCP and UDP.

**Contract:**
The SSH ports on management devices are inaccessible from tenant devices.
Policies as Logical Formulas

Precise Semantics as bit-vector formulas

Traditional Low level of Configuration network managers use

(10.20.0.0 ≤ srcIp 10.20.31.255) ∧
Allow: (157.55.252.0 ≤ dstIp ≤ 157.55.252.255) ∧
(protocol = 6)

Deny: (65.52.244.0 ≤ dstIp ≤ 65.52.247.255) ∧
(protocol = 4)

Combining semantics

(\bigvee_i Allow_i) ∧ (\bigwedge_j \neg Deny_j)
Beyond Z3: a new idea to go from one violation to all violations

Representing solutions
- $2 \times 2^{16} \times 2 \times 2^8 \times 2 = 2^{27}$ single solutions, or
- 8 products of contiguous ranges, or
- A single product of ranges

SecGuru contains optimized algorithm for turning single solutions into all (product of ranges)

$srclp = 10.20.0.0/16, 10.22.0.0/16$
$dstlp = 157.55.252.000/24, 157.56.252.000/24$
$port = 80,443$
SecGuru in WANetmon

Cluster dc/dm/cluster/dm1prdstr08

Network ACL Validation Alerts for the cluster

This check validates the correctness of all the network ACLs in the devices in the cluster.

<table>
<thead>
<tr>
<th>Device</th>
<th>Timestamp</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>dm1-x3hl-cis-15-01</td>
<td>Sat Sep 14 2013 11:27:41 GMT-0700 (Pacific)</td>
<td></td>
</tr>
<tr>
<td>acl-name</td>
<td>IP Address Range</td>
<td>Error</td>
</tr>
<tr>
<td>mgmt-only</td>
<td>10.143.197.208/28</td>
<td>Partially blocked</td>
</tr>
<tr>
<td>mgmt-only</td>
<td>10.143.197.224/27</td>
<td>Partially blocked</td>
</tr>
<tr>
<td>mgmt-only</td>
<td>10.143.198.0/26</td>
<td>Partially blocked</td>
</tr>
<tr>
<td>mgmt-only</td>
<td>10.143.198.64/27</td>
<td>Partially blocked</td>
</tr>
<tr>
<td>mgmt-only</td>
<td>10.143.198.96/28</td>
<td>Partially blocked</td>
</tr>
<tr>
<td>ssh-only</td>
<td>10.143.197.208/28</td>
<td>Blocked</td>
</tr>
<tr>
<td>ssh-only</td>
<td>10.143.197.224/27</td>
<td>Blocked</td>
</tr>
</tbody>
</table>

Cluster dc/dm/cluster/dm1prdstr01

Network ACL Validation Alerts for the cluster

This check validates the correctness of all the network ACLs in the devices in the cluster.

40,000 ACL checks per month
Each check 50-200ms
20 bugs/month (mostly for build-out)
Self-contained Windows Firewall Checker

By Andrew Helwer, Azure

Two minimal tab-separated example firewall rule files are as follows (see Examples directory):

Firewall 1:

<table>
<thead>
<tr>
<th>Name</th>
<th>Enabled</th>
<th>Action</th>
<th>Local Port</th>
<th>Remote Address</th>
<th>Remote Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foo1</td>
<td>Yes</td>
<td>Allow</td>
<td>100</td>
<td>10.3.141.0</td>
<td>100</td>
<td>UDP</td>
</tr>
<tr>
<td>Bar1</td>
<td>Yes</td>
<td>Allow</td>
<td>200</td>
<td>10.3.141.0</td>
<td>200</td>
<td>TCP</td>
</tr>
</tbody>
</table>

Firewall 2:

<table>
<thead>
<tr>
<th>Name</th>
<th>Enabled</th>
<th>Action</th>
<th>Local Port</th>
<th>Remote Address</th>
<th>Remote Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foo2</td>
<td>Yes</td>
<td>Allow</td>
<td>100</td>
<td>10.3.141.0</td>
<td>100</td>
<td>UDP</td>
</tr>
<tr>
<td>Bar2</td>
<td>Yes</td>
<td>Allow</td>
<td>200</td>
<td>10.3.141.1</td>
<td>200</td>
<td>TCP</td>
</tr>
</tbody>
</table>

This generates the following output from FirewallEquivalenceCheckerCmd.exe:

Microsoft.FirewallEquivalenceCheckerCmd.exe --firewall1 ./firewall1.txt --firewall2 ./firewall2.txt
Parsing first firewall...
Parsing second firewall...
Running equivalence check...
Firewalls are NOT equivalent.

Inconsistently-handled packets:

<table>
<thead>
<tr>
<th>PID</th>
<th>Src Address</th>
<th>Src Port</th>
<th>Dest Port</th>
<th>Protocol</th>
<th>Allowed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10.3.141.0</td>
<td>100</td>
<td>200</td>
<td>TCP</td>
<td>First</td>
</tr>
<tr>
<td>1</td>
<td>10.3.141.1</td>
<td>200</td>
<td>200</td>
<td>TCP</td>
<td>Second</td>
</tr>
</tbody>
</table>

Firewall rules matching inconsistently-handled packets:

<table>
<thead>
<tr>
<th>PID</th>
<th>Firewall</th>
<th>Action</th>
<th>Rule Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>First</td>
<td>Allow</td>
<td>Bar1</td>
</tr>
</tbody>
</table>

https://github.com/Z3Prover/FirewallChecker
Formal Methods and Tools

- High-level Specification (TLA+)
- Correctness of Cryptography and Protocols ($\Gamma^*$, Ivy, P#)
- Bug Finding and Verification for C/C++ (SAGE, Corral)
- Network Verification (SecGuru)
Microsoft Security
Risk Detection

An important step in software security is identifying high-risk targets...

**Dataflow**, movement of bits between two network entities

**Entry Point**, where external data enters an entity

**Trust Boundary**, a dividing line across which data flows

**Security Bug**, any regular code or design bug
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 4) crash();
}

Path constraint:

input = "good"

Gen 1  Gen 2  Gen 3  Gen 4

10+ years of sustained investment
White Box Fuzzing (SAGE)

Check for Crashes → Code Coverage → Generate Path Constraints → Solve Constraints

Input0 → Coverage Data → Constraints

Input1, Input2, ..., InputN
Security Risk Detection and the SDL

SAGE used internally at Microsoft to meet SDL verification requirements
White Box Fuzzing (SAGE) Results

Since 2007: many new security bugs found
   – Apps: decoders, media players, document processors, ...
   – Bugs: Write A/Vs, Read A/Vs, Crashes, ...
   – Many triaged as “security critical, severity 1, priority 1”

• 100s of apps, 100s of bugs
   – Bug fixes shipped quietly (no MSRCs) to 1 Billion+ PCs
   – Millions of dollars saved (for Microsoft and the world)

• “Practical Verification”
   – <5 security bulletins in SAGE-cleaned parsers since 2009
**Step 1:** The user manually uploads the target binaries and seed files to the Customer VM, and uses the wizard to configure the job.

**Parallelized Runs**

**Step 2:** Security Risk Detection validates the job, minimizes the seed files, and then clones the customer VM dozens of times based on workload.

**Step 3:** Multiple fuzzers run for multiple days: the target app is executed roughly 8,000,000 times, each time with a slightly modified input file that’s intended to crash the target.

**Step 4:** Any time an execution fails, the offending file is sent to the repro VM to ensure the bug is reproducible.

**Step 5:** Bugs that repro (along with the file, stack trace, and other debug info) are available in the portal and API in real time.
More on Dynamic Symbolic Execution

For real programs, compiled through LLVM
• [https://klee.github.io/](https://klee.github.io/)

For a small subset of Python, using Z3
• [https://github.com/thomasjball/PyExZ3](https://github.com/thomasjball/PyExZ3)
Hot off the press

REST-ler: Automatic Intelligent REST API Fuzzing

• Vaggelis Atlidakis, Patrice Godefroid, Marina Polishchuk
• https://arxiv.org/abs/1806.09739
Formal Methods and Tools

- High-level Specification (TLA+)
- Correctness of Cryptography and Protocols ($F^*$, Ivy, P#)
- Bug Finding and Verification for C/C++ (SAGE, Corral)
- Network Verification (SecGuru)

Thinking

Programming

Testing

Verifying
MSR’s Project Everest

**Goal:** verified HTTPS replacement

**Challenges:**
- scalability of verification
- performance
- usable tool chain

https://project-everest.github.io/
Subgoal: Verified low-level crypto
Efficient crypto requires customizations

- **Poly1305**: Uses the prime field with $p = 2^{130} - 5$
  - Need 130 bits to represent a number
  - Efficient implementations require custom bignum libraries to delay carries
  - On X86: use 5 32-bit words, but using only 26 bits in each word
  - On X64: use 3 64-bit words, but using only 44 bits in each word

- **Curve25519**: Uses the prime field with $p = 2^{255} - 19$
  - On X64: use 5 64-bit words, but using only 51 bits per word

- OpenSSL has 12 unverified bignum libraries optimized for each case
A generic bignum library

Bignum code can be **shared** between Curve25519, Ed25519 and Poly1305, which all use different fields

Only modulo is specific to the field (optimized)

Consequently:
- write once
- verify once
- extract three times

```haskell
let prime = pow 2 255 - 19
let word_size = 64
let len = 5
let limb_size = 51
```

```haskell
let prime = pow 2 130 - 5
let word_size = 64
let len = 3
let limb_size = 44
```
Prove correct in F*, extract to efficient C

Mathematical spec in F*

poly1305_mac: (1) computes a polynomial in GF(2^{130}-5),
(2) stores the result in tag,
(3) does not modify anything else

Efficient C implementation

Verification imposes no runtime performance overhead

Sample code Poly1305 MAC
F* source: core-ML with dependent types and effects

```
let poly1305_mac: tag:nbytes 16 \rightarrow
  len:u32 \rightarrow
  msg:nbytes len\{disjoint tag msg\} \rightarrow
  key:nbytes 32 \{disjoint msg key \land disjoint tag key\} \rightarrow
  ST unit
  (requires (\lambda h \rightarrow msg \in h \land key \in h \land tag \in h))
  (ensures (\lambda h0 _ h1 \rightarrow \_)) = ...
```

C source, tuned for readability, compliance with C linters etc.

```
void poly1305_mac(uint8_t *tag, uint32_t len, 
  uint8_t *msg, uint8_t *key)
{
  uint64_t tmp[10] = { 0 }; 
  uint64_t *acc = tmp
  uint64_t *r = tmp + (uint32_t)5;
  uint8_t s[16] = { 0 }; 
  Crypto_Symmetric_Poly1305_poly1305_init(r, s, key);
  Crypto_Symmetric_Poly1305_poly1305_process(msg, len, acc, r);
  Crypto_Symmetric_Poly1305_poly1305_finish(tag, acc, s);
}
```
Performance of Everest’s High Assurance Crypto Library (HACL*)

- Several complete TLS ciphersuites
- Verification can scale up!

Verification enables using 64x64 bit multiplications, without fear of getting it wrong.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Code+Proofs (Low*)</th>
<th>C Code (C loc)</th>
<th>Verification (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salsa20</td>
<td>651</td>
<td>372</td>
<td>280</td>
</tr>
<tr>
<td>Chacha20</td>
<td>691</td>
<td>243</td>
<td>336</td>
</tr>
<tr>
<td>Chacha20-Vec</td>
<td>1656</td>
<td>355</td>
<td>614</td>
</tr>
<tr>
<td>SHA-256</td>
<td>622</td>
<td>313</td>
<td>798</td>
</tr>
<tr>
<td>SHA-512</td>
<td>737</td>
<td>357</td>
<td>1565</td>
</tr>
<tr>
<td>HMAC</td>
<td>56</td>
<td>22</td>
<td>773</td>
</tr>
<tr>
<td>Bignum-lib</td>
<td>-</td>
<td>313</td>
<td>798</td>
</tr>
<tr>
<td>Poly1305</td>
<td>648</td>
<td>355</td>
<td>614</td>
</tr>
<tr>
<td>X25519-lib</td>
<td>-</td>
<td>313</td>
<td>798</td>
</tr>
<tr>
<td>Curve25519</td>
<td>1901</td>
<td>798</td>
<td>246</td>
</tr>
<tr>
<td>Ed25519</td>
<td>7219</td>
<td>2479</td>
<td>2118</td>
</tr>
<tr>
<td>AEAD</td>
<td>309</td>
<td>100</td>
<td>606</td>
</tr>
<tr>
<td>SecretBox</td>
<td>171</td>
<td>132</td>
<td>62</td>
</tr>
<tr>
<td>Box</td>
<td>188</td>
<td>270</td>
<td>43</td>
</tr>
<tr>
<td>Total</td>
<td>22,926</td>
<td>7,225</td>
<td>9127</td>
</tr>
</tbody>
</table>

Table 1: HACL* code size and verification times

- With performance as good as or better than hand-written C

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>HACL* cycles/ECDH</th>
<th>OpenSSL cycles/ECDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-256</td>
<td>13.43</td>
<td>16.11</td>
</tr>
<tr>
<td>SHA-512</td>
<td>8.09</td>
<td>10.34</td>
</tr>
<tr>
<td>Salsa20</td>
<td>6.26</td>
<td>-</td>
</tr>
<tr>
<td>ChaCha20</td>
<td>6.37 (ref)</td>
<td>7.84</td>
</tr>
<tr>
<td></td>
<td>2.87 (vec)</td>
<td></td>
</tr>
<tr>
<td>Poly1305</td>
<td>2.19</td>
<td>2.16</td>
</tr>
<tr>
<td>Curve25519</td>
<td>154,580</td>
<td>358,764</td>
</tr>
<tr>
<td>Ed25519 sign</td>
<td>63.80</td>
<td>-</td>
</tr>
<tr>
<td>Ed25519 verify</td>
<td>57.42</td>
<td>-</td>
</tr>
<tr>
<td>AEAD</td>
<td>8.56 (ref)</td>
<td>8.55</td>
</tr>
<tr>
<td></td>
<td>5.05 (vec)</td>
<td></td>
</tr>
</tbody>
</table>
Mozilla has partnered with [INRIA](https://www.inria.fr) and [Project Everest](https://www.microsoft.com/en-us/research/project/everest) (Microsoft Research, CMU, INRIA) to bring components from their formally verified [HACL*](https://github.com/hacl-crypto) cryptographic library into [NSS](https://www.mozilla.org/security/products/nss/), the security engine which powers Firefox.
Project Everest: Open Source

• https://www.github.com/FStarLang/FStar
• https://www.github.com/FStarLang/kremlin

• https://www.github.com/mitls/mitls-fstar
• https://www.github.com/mitls/hacl-star

• https://www.github.com/project-everest/vale
Formal Methods and Tools

- High-level Specification (TLA+)
- Correctness of Cryptography and Protocols (F*, Ivy, P#)
- Bug Finding and Verification for C/C++ (SAGE, Corral)
- Network Verification (SecGuru)

thinking
programming
testing
verifying
TLA+ (Leslie Lamport)

- A language for high-level modelling of digital systems, especially concurrent and distributed systems

- Tools for checking the models (TLC)

- IDE for end-to-end experience (Toolbox)

- https://github.com/tlaplus

**How Amazon Web Services Uses Formal Methods**

SINCE 2011, ENGINEERS AT Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects. Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.

**key insights**

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.
Chris Newcombe, AWS

• Formal methods find bugs in system designs that cannot be found through any other technique we know of

• Formal methods are surprisingly feasible for mainstream software development and give good return on investment

• At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.
Chris Newcombe, AWS

• Formal methods find bugs in system designs that cannot be found through any other technique we know of.

• Formal methods are surprisingly feasible for mainstream software development and give good return on investment.

• At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

“TLA+ is the most valuable thing that I've learned in my professional career. It has changed how I work, by giving me an immensely powerful tool to find subtle flaws in system designs. It has changed how I think, by giving me a framework for constructing new kinds of mental-models, by revealing the precise relationship between correctness properties and system designs, and by allowing me to move from 'plausible prose' to precise statements much earlier in the software development process.”
Formal Methods and Tools

- **High-level Specification**
  - (TLA+)

- **Correctness of Cryptography and Protocols**
  - (F*, Ivy, P#)

- **Bug Finding and Verification for C/C++**
  - (SAGE, Corral)

- **Network Verification**
  - (SecGuru)

**Logic**

-thinking

-programming

-testing

-verifying