2024 HIGH CONFIDENCE SOFTWARE AND SYSTEMS CONFERENCE

THEME: ASSURED OPEN SOURCE AND MEMORY SAFETY

Formal Verification of AWS-LibCrypto

Work completed by AWS and Galois, Inc. through collaboration

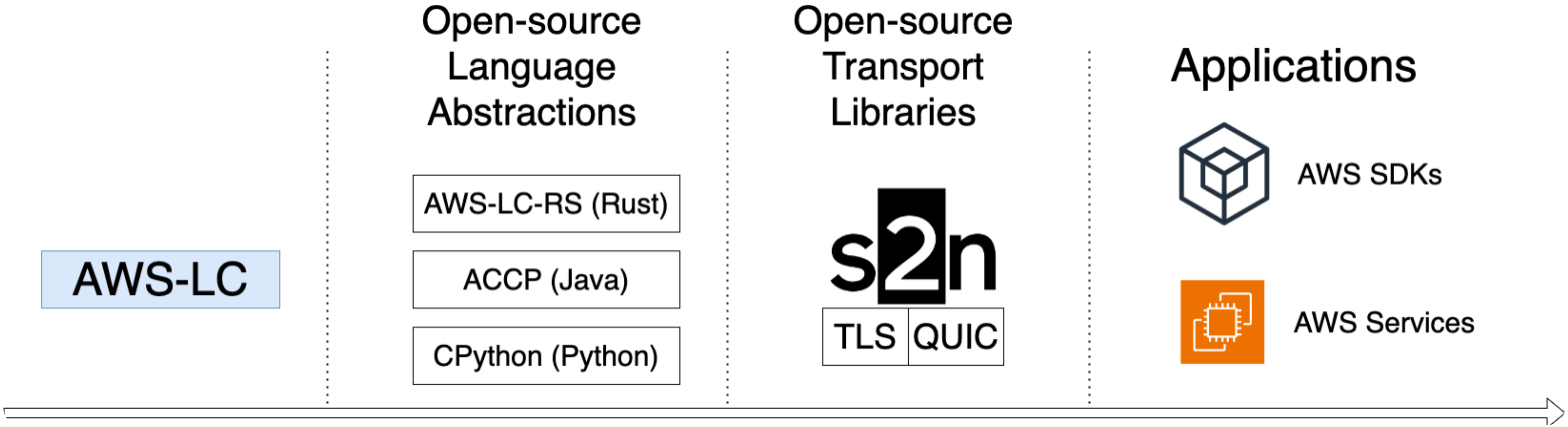
Speaker: Yan Peng (she/her)

Applied Scientist
Amazon Web Services, Inc.

Outline

1. AWS-LibCrypto
2. Formal Verification Overview
3. C and x86 Verification using SAW
4. Arm Verification
5. s2n-bignum
6. CI and Proof Maintenance

AWS-LibCrypto (AWS-LC)



- An open-source general-purpose cryptographic library owned and maintained by AWS
- Forked from BoringSSL and optimized for AWS use cases
- FIPS 140-3 validated
- Support multiple platforms for customer needs

Performance Optimization

- Cryptographic primitives have cumulative performance and cost impact over network connections

- Algorithm level:
  - EC: windowed double-and-add scalar point multiplication
  - AES-GCM: Karatsuba multiplication & aggregated reduction
- Micro-architecture level:
  - Access to all machine instructions
  - Precise control over the scheduling of operations - parallelism

Safety Mechanisms

- Cryptography is the foundation for protecting customer data
  - David A. Wheeler – *How to Prevent the next Heartbleed* [1]

“Do not use just one of these tools and techniques to develop secure software.”

- Testing and dynamic analysis: positive and negative unit tests, fuzz tests, Clang sanitizers, Valgrind, etc.
- Also, ***formal verification***
  - Use of automated logical reasoning to prove *properties* of a program or system
  - Properties: memory safety and functional correctness

Highly-optimized open-source cryptographic library is challenging to verify

- Written in multiple languages (C, assembly for various platforms)
  - Use of multiple formal verification tools is often unavoidable
  - Proof integration
- Highly-optimized
  - Each optimization requires some proof effort to prove soundness
    - Large proof terms, we want to build robust automation using SAT/SMT
    - Some optimization could not be automatically solved, need user guidance
- Formal proofs need to catch up with new changes/optimizations

Verified Algorithms

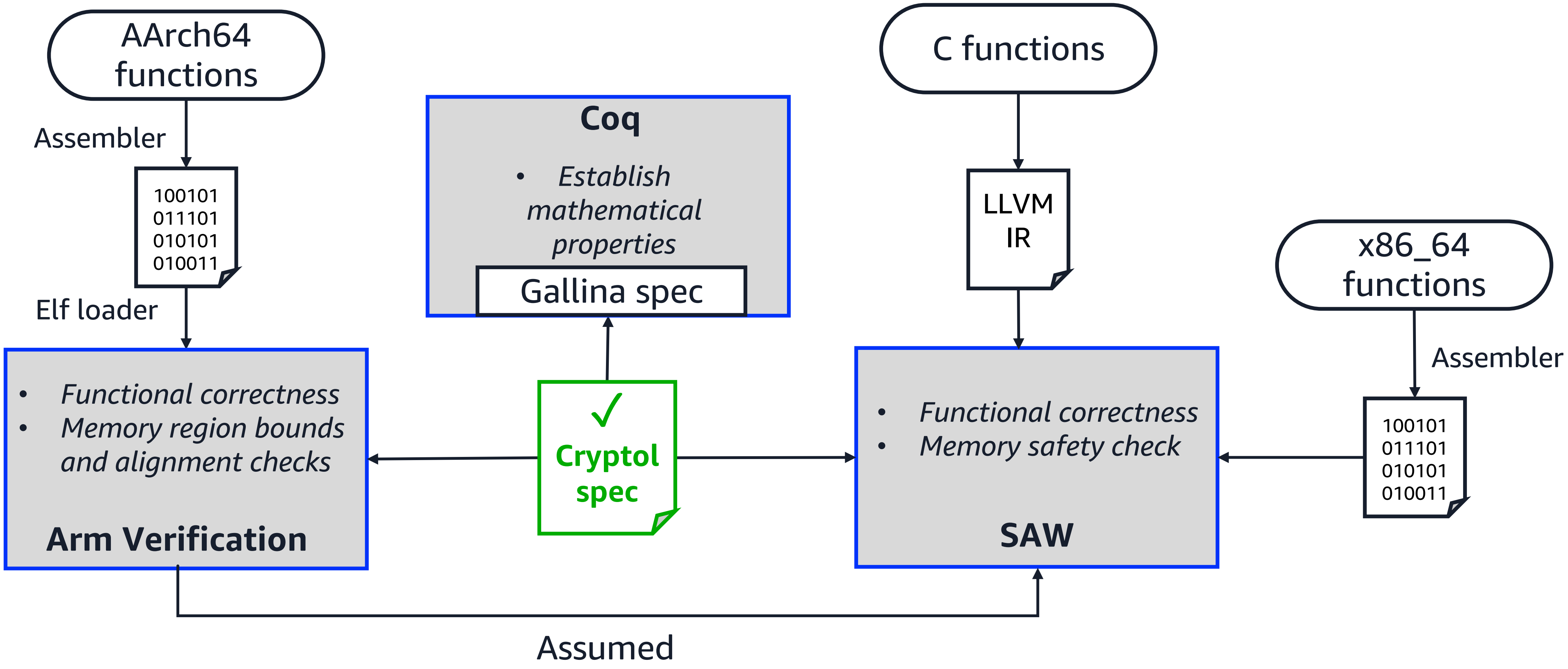
Verified up to API unbounded proof

Algorithm	Variants	Platform	Tech	LOC (approx.)	Proof Run Time
SHA-2	384, 512	SandyBridge+	SAW	1000	150s
SHA-2	384	Neoverse-n1 Neoverse-v1	SAW, Prototype Arm Verification Tool	2600	230s
HMAC	SHA-384	SandyBridge+	SAW	1000	327s
AES-KW(P)	256	SandyBridge+	SAW	700	215s
Elliptic Curve Keys and Parameters	P-384	SandyBridge+	SAW, Coq, HOL-Light	2400+20000	620s
ECDSA	P-384, SHA-384	SandyBridge+	SAW	1500	703s(~11mins)
ECDH	P-384	SandyBridge+	SAW, Coq, HOL-Light	400	423s
HKDF	HMAC-SHA384	SandyBridge+	SAW	700	220s

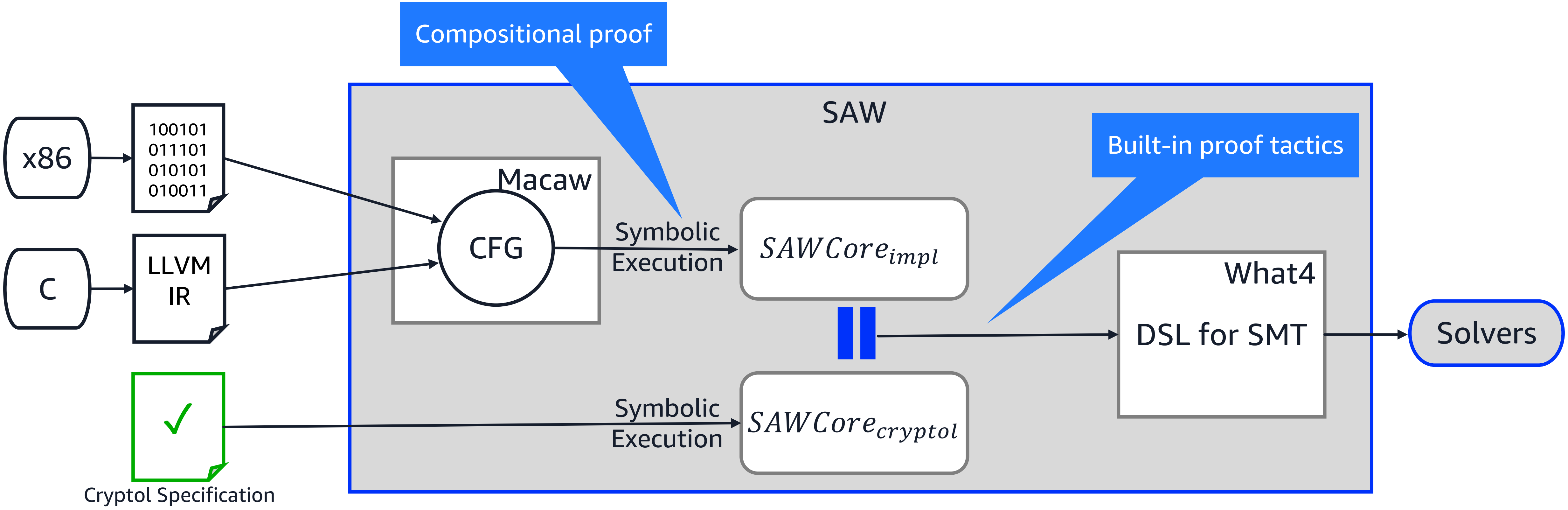
Total ~ 10,000 SAW

- SandyBridge+ : x86_64 with AES-NI, CLMUL and AVX

AWS-LC Formal Verification Workflow

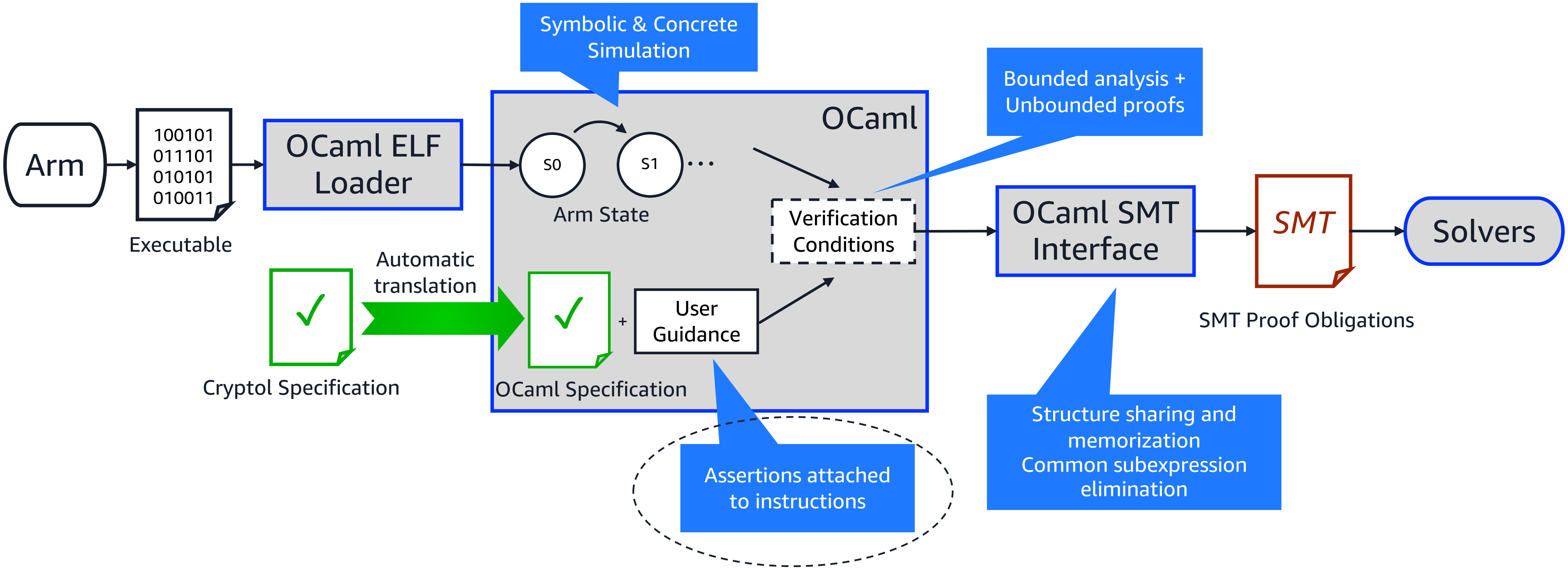


Verifying C and x86_64 using SAW



- Unbounded proofs – improved comparing to previous results
- Does not support Arm (64bit)

Verifying Arm Assembly



- Memory safety: memory access is within bounds and correctly aligned
- Implemented in OCaml, currently exploring Lean

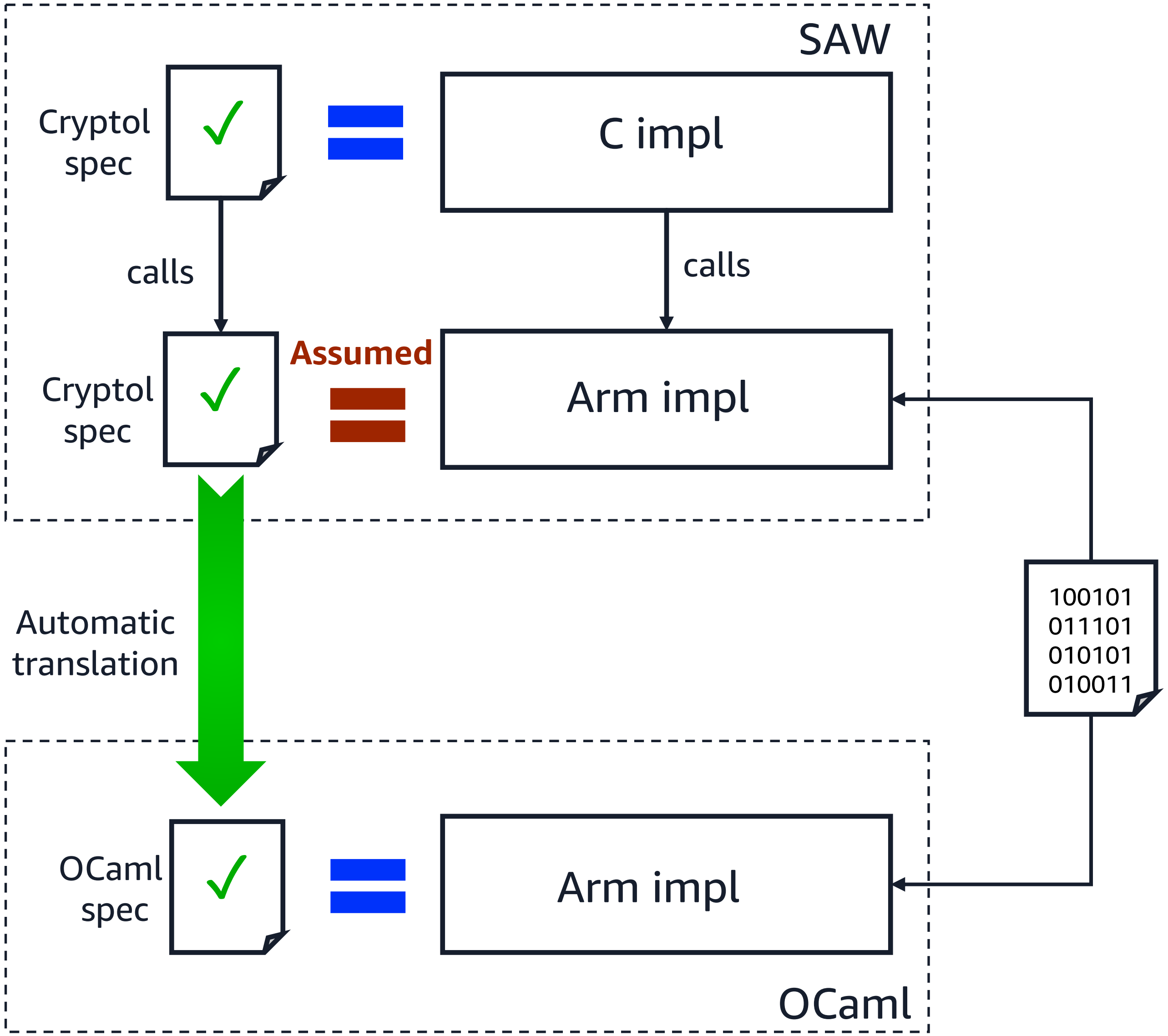
Integrating Arm Proofs with SAW

Verify C function through compositional proof

Assume correctness of assembly

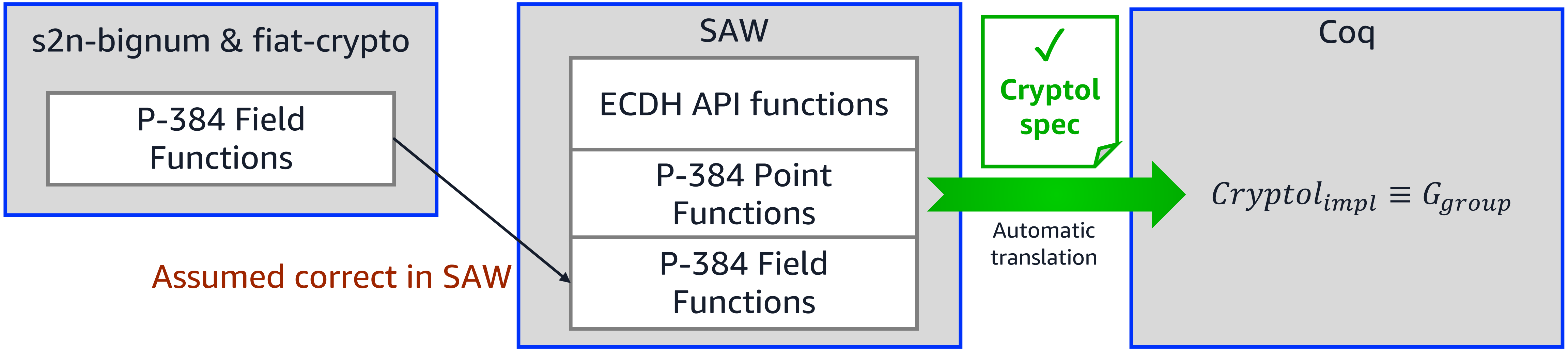
Automatic translation of Cryptol spec to OCaml

Verify Arm assembly using translated spec



Use of Coq for Mathematical Reasoning

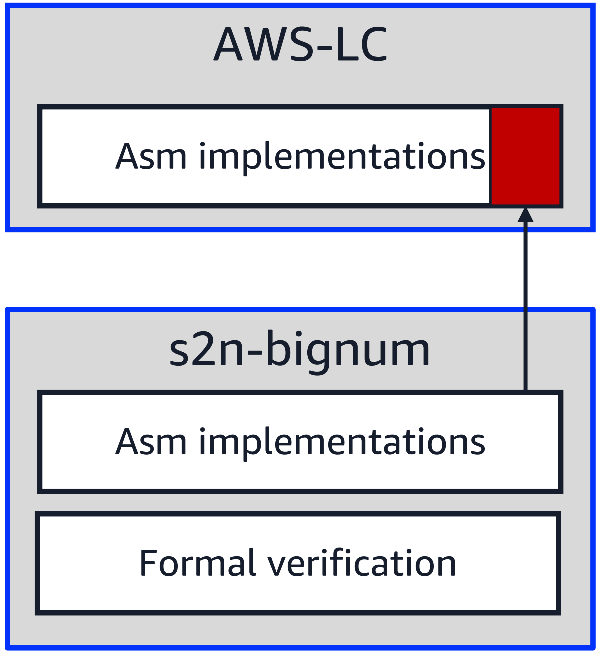
- ECDH verification workflow



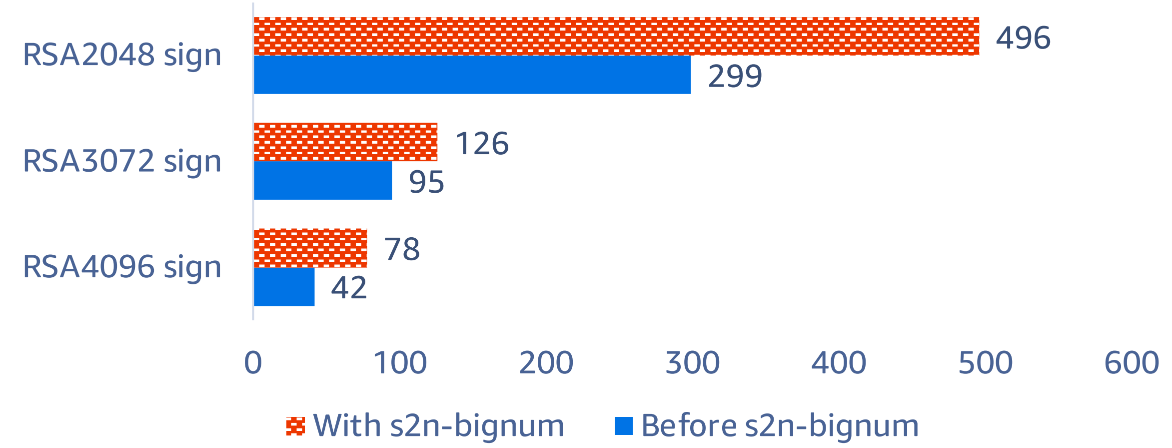
- Mathematical reasoning and induction is easier in a theorem prover
  - We want: the group multiplication used in the ECDH implementation is in the correct group of P-384 points

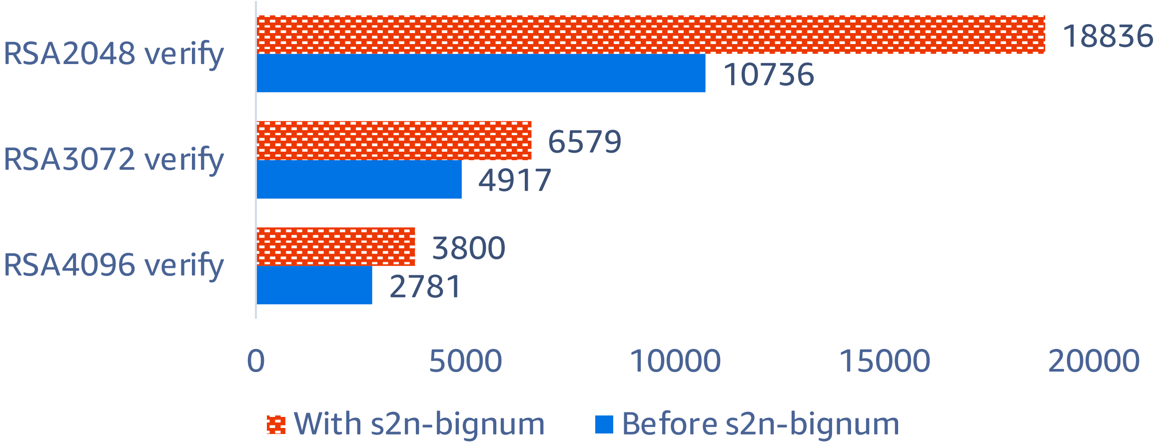
s2n-bignum

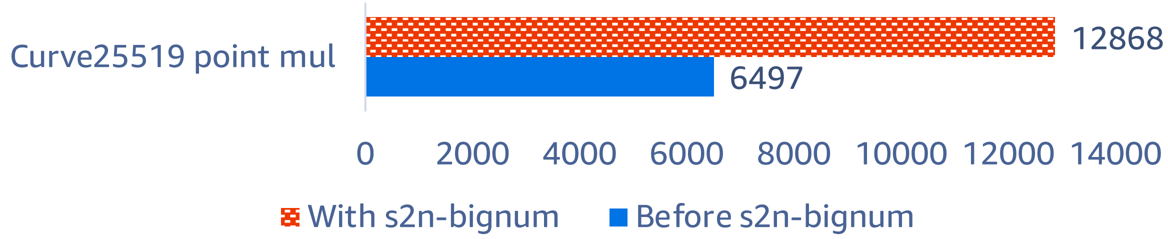
- An open-source library developed at AWS
- Efficient implementation of low-level big number operations
- Written in constant-time fashion
- Supports both x86_64 and aarch64
- Formally verified in HOL-Light



Formal verification enables fearless performance optimization







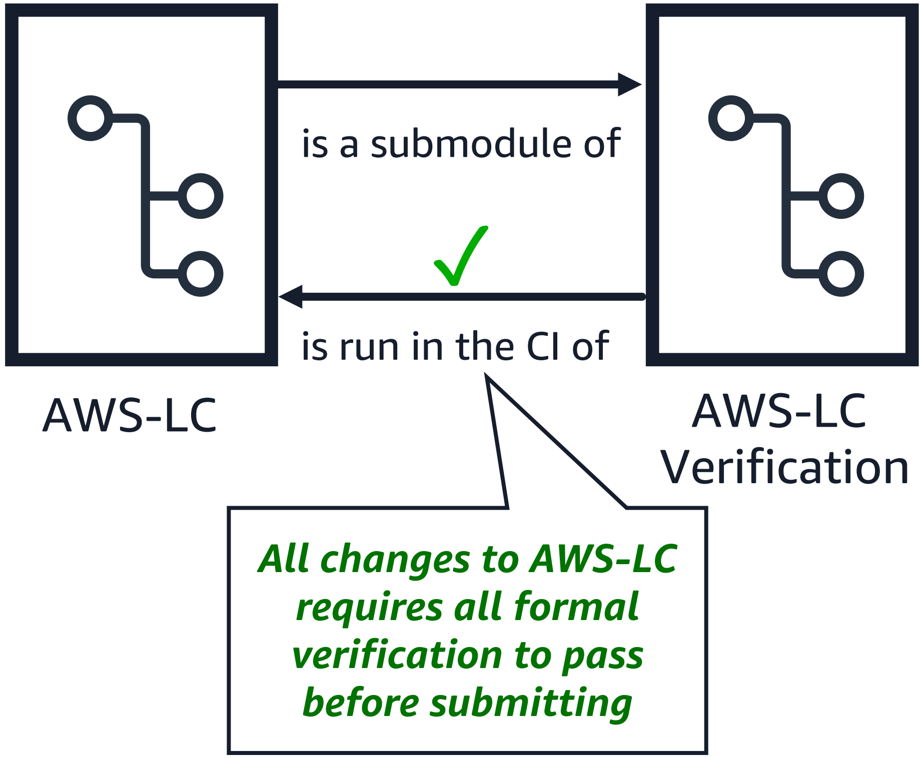
- RSA sign: 30%~80%; verify: 30%~75%
- Curve25519 point mul: 98%

- Fine tuning for the micro-architecture
- Curve25519:
  - Lenngren's X25519 optimization[2]
  - SLOTHY[3]

Note: performance (op/sec) measured on Graviton2 using benchmarking tool provided in AWS-LC

Continuous Integration and Proof Maintenance

- Formal verification needs to run relatively **fast**
- Formal verification of open-source libraries requires **continuous effort**
  - Formal proofs need to catch-up with new optimizations



- Total CI run time 30min:
  - Saw-x86_64: 17mins
  - Saw-aarch64: 2mins
  - Coq: 28mins (mostly building fiat-crypto)
  - Arm Verification: 9mins
- Requires reasonable effort for proof maintenance
  - Year 2023, around 16/616(PRs) fixes
  - LLM?

Summary and Lessons Learnt

Summary: We formally verified several critical algorithms in the open-source cryptographic library AWS-LC

- These proofs are open-source and run in the continuous integration

Lessons Learnt:

- Verifying highly-optimized cryptographic library is a challenging task that requires multiple formal techniques/tools
- Formal verification enables fearless performance optimization
- Formal verification of open-source libraries requires continuous effort

Thank you!

Yan Peng

yppe@amazon.com

Open-source cryptography @ AWS
https://aws.amazon.com/security/opensource/cryptography

AWS-LC
https://github.com/aws/aws-lc

AWS-LC-verification
https://github.com/awslabs/aws-lc-verification

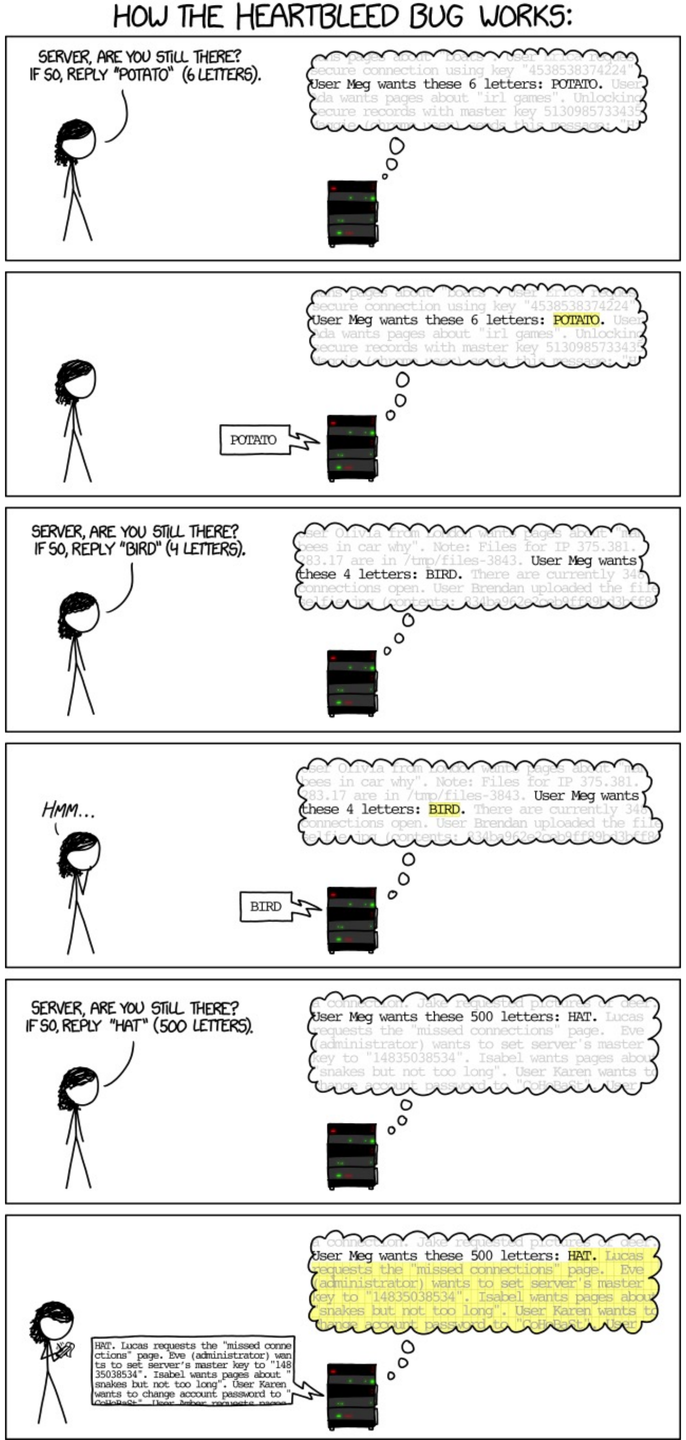
s2n-bignum
https://github.com/awslabs/s2n-bignum

References

[1] https://dwheeler.com/essays/heartbleed.html

[2] https://github.com/Emill/X25519-AArch64/blob/master/X25519_AArch64.pdf

[3] https://github.com/slothy-optimizer/slothy



Source: https://xkcd.com/1354/