ENG25519: Faster TLS 1.3 handshake using optimized X25519 and Ed25519

Jipeng Zhang[1], Junhao Huang[2,3], Lirui Zhao[1], Donglong Chen[2], Çetin Kaya Koç[1,4,5]

[1]Nanjing University of Aeronautics and Astronautics, Jiangsu, China
jp-zhang@outlook.com, lirui.zhao@outlook.com
[2]Guangdong Provincial Key Laboratory IRADS, BNU-HKBU United International College
huangjunhao@uic.edu.cn, donglongchen@uic.edu.cn
[3]Hong Kong Baptist University
[4]Iğdır University
[5]University of California Santa Barbara
cetinkoc@ucsb.edu

Artifact: `https://github.com/Ji-Peng/eng25519_artifact`

August 15, 2024

Outline

1. Motivations
2. Background
3. Optimized X/Ed25519 implementation & ENG25519
4. Conclusions

Motivations

- How can AVX-512IFMA instructions accelerate ECC?
  - Optimizing ECC using ARM NEON and AVX2 instructions has been thoroughly researched.
  - However, using AVX-512IFMA instructions remains underexplored.
- How can the optimized ECC implementation be integrated into TLS applications?
  - Few works consider integration; most focus solely on optimizing cryptographic implementations.
- How can the cold start issue of vector units be mitigated?
  - The cold start issue can cause some primitives to be up to 3.8 times slower than normal.
- To what extent can our optimized cryptographic implementation improve TLS applications?
  - It is more interesting to understand the improvements to TLS applications rather than just focusing on cryptographic primitive microbenchmarks.

Background: AVX-512 & X25519 and Ed25519

AVX-512

- 32 512-bit registers; Each 512-bit register can be divided into 32 16-bit, 16 32-bit, or 8 64-bit segments.
- AVX-512IFMA supports 52-bit multipliers, whereas AVX2 and AVX-512F only support 32-bit multipliers.

X25519 and Ed25519

- X25519, designed by Daniel J. Bernstein, is a Diffie-Hellman key exchange protocol based on Curve25519.
- Ed25519, designed by Daniel J. Bernstein et al., is an Edwards-curve digital signature algorithm.
- In 2018, RFC 8446 included X25519 and Ed25519 in the supported cipher suites for TLS 1.3.

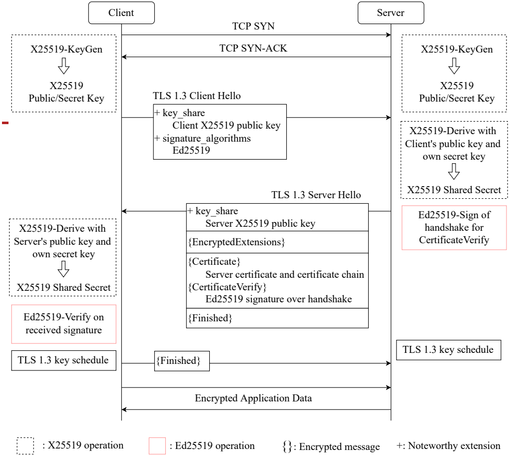
Background: TLS 1.3 handshake & DNS over TLS

TLS 1.3 handshake

- Client op; Server op.
- X25519-KeyGen/KeyGen+X25519-Derive+Ed25519-Sign+X25519-Derive+Ed25519-Verify.

DNS over TLS

- TLS handshake→DNS queries and responses over the TLS connection.



Optimized X25519 and Ed25519 implementation

Field arithmetic

- Radix-2^{51}: A field element $f = f_0 + 2^{51}f_1 + 2^{102}f_2 + 2^{153}f_3 + 2^{204}f_4$.
- 8×1-way: One subroutine performs 8 parallel independent field operations.
- We formally verified our field implementations using CryptoLine.

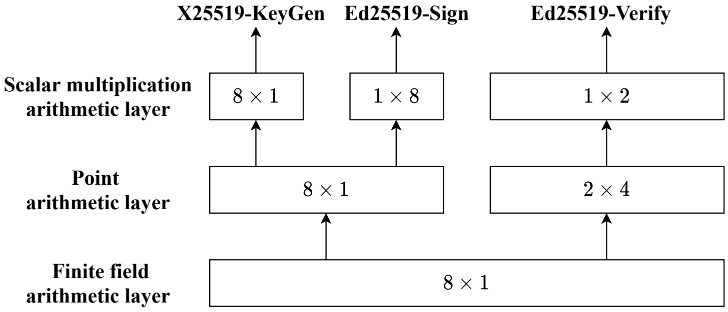


Figure: An overview of our X/Ed25519 implementation.

Optimized X25519 and Ed25519 implementation

Strategy: “finite field arithmetic” $\rightarrow$ “point arithmetic” $\rightarrow$ “scalar multiplication”

- X25519-KeyGen: $8 \times 1 \rightarrow 8 \times 1 \rightarrow 8 \times 1$
  - 12 times that of the OpenSSL implementation and 2.32 times that of Cheng et al.’s implementation.
- X25519-Derive: We don’t provide a faster X25519-Derive implementation than Hisil et al.
- Ed25519-Sign: $8 \times 1 \rightarrow 8 \times 1 \rightarrow 1 \times 8$
  - 3.79 times that of the OpenSSL implementation and 1.18 times that of Faz-Hernández et al.’s implementation.
- Ed25519-Verify: $8 \times 1 \rightarrow 2 \times 4 \rightarrow 1 \times 2$
  - 3.33 times that of the OpenSSL implementation and 1.33 times that of Faz-Hernández et al.’s implementation.

ENG25519: An OpenSSL ENGINE

- ENG25519 is based on OpenSSL ENGINE APIs, libsuola, and engntru.
- Our optimized X/Ed25519 implementations can be transparently integrated into OpenSSL and TLS applications through ENG25519.

Table: Detailed configuration of ENG25519.

Subroutine	Implementation
X25519-KeyGen Ed25519-KeyGen	Our $8 \times 1 \to 8 \times 1 \to 8 \times 1$ impl. $\textit{batch-size} = 16$
X25519-Derive	$4 \times 2 \to 1 \times 4$ impl. of Hisil et al.
Ed25519-Sign	Our $8 \times 1 \to 8 \times 1 \to 1 \times 8$ impl.
Ed25519-Verify	Our $8 \times 1 \to 2 \times 4 \to 1 \times 2$ impl.

ENG25519: How to mitigate the cold-start issue?

Code start issue

- The processor will set the upper parts of the AVX2/AVX-512 vector units to a low-power mode to save power if the units are not in use for about 675 µs, leading to a warm-up phase of approximately 14 µs (56,000 clock cycles at 4 GHz) when an AVX2/AVX-512 instruction is executed in the low-power mode.
- During the warm-up phase, the throughput of the related instructions is 4.5 times slower than usual.
- All X/Ed25519 primitives suffer from varying degrees of performance degradation; especially the X25519-KeyGen takes 3.8 times longer in the DoT scenario than in the warm-start scenario.

ENG25519: How to mitigate the cold-start issue?

We designed a heuristic auxiliary thread that performs different actions based on the application's varying load conditions.

- Low-load scenarios: It takes no action to avoid disrupting the processor's power-saving strategies.
- Medium-load: It periodically executes a vector instruction.
- High-load: The frequent cryptographic operations inherently prevent entering low-power mode.

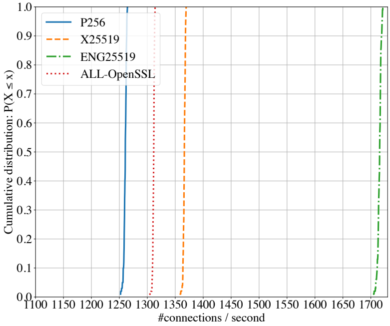
Table: Amortized CPU cycles (CC) to generate a keypair.

Batch size	Amortized CC with auxiliary thread	Amortized CC without auxiliary thread
1	10,315	28,450
4	9,107	19,388
8	9,003	14,108
16	8,980	11,406

ENG25519: Benchmark of TLS handshake

Client: tls_timer ↔ Server: OpenSSL s_server

On average, the proposed ENG25519 setting (1,707 #connections/second) enables 25% and 35% more handshakes per second than X25519 (1,366) and P256 (1,260), respectively.



ENG25519: Benchmark of DoT query

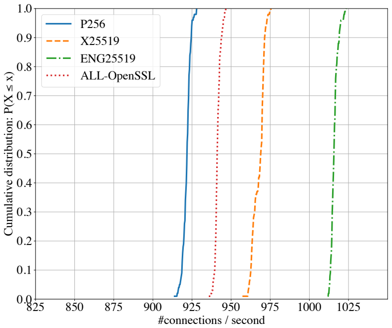
Client: dot_timer ↔ Server: unbound DoT server[1]

End-to-end experiments

- Our ENG25519 outperforms all other configurations.

Peak throughput

- Our ENG25519 configuration achieved a significant improvement, achieving 290,315 #queries/min, which represents a 41% and 24% increase over P256 (206,275) and X25519 (234,875), respectively.



[1] https://nlnetlabs.nl/projects/unbound/about/

Conclusions

- Faster X/Ed25519 implementation using AVX-512IFMA.
- Integration of optimized X/Ed25519 implementations into TLS; faster TLS 1.3 handshake; increased DNS over TLS throughput.
- Under cold start conditions, some primitives may suffer a performance degradation of up to 3.8 times. If the vector implementation does not achieve significant improvements, a reevaluation of the vector implementation versus the x64 implementation is necessary.
- Open source artifact: `https://github.com/Ji-Peng/eng25519_artifact`.

Thanks

Thanks for listening