OpenSSL Hardware offload Enhancement

Ping Yu

Intel NPG

Agenda

Background introduction for TLS

TLS acceleration with QuickAssist Technology

OpenSSL asynchronous acceleration framework

Enable OpenSSL asynchronous acceleration framework in Nginx

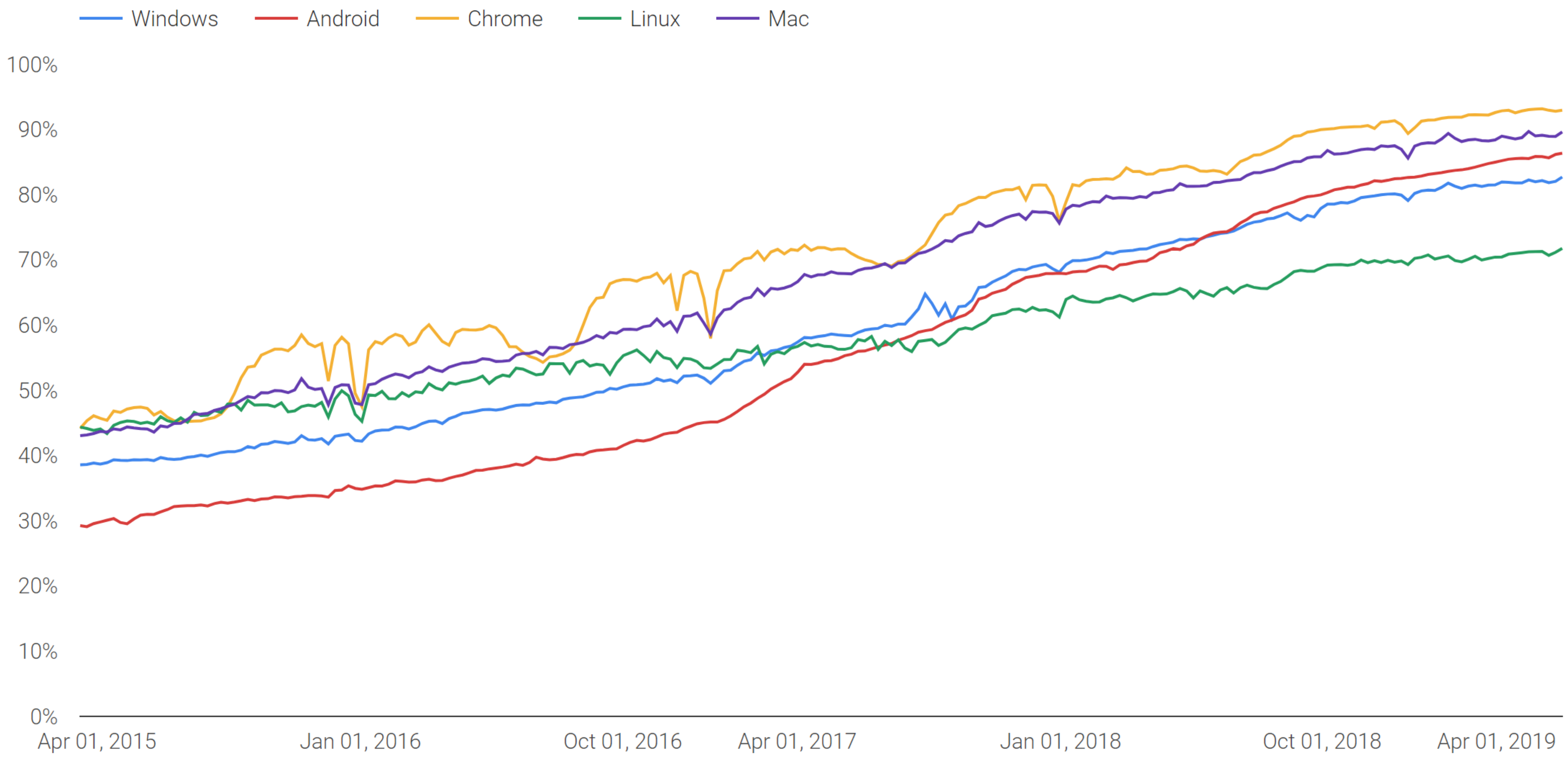
Enhance OpenSSL asynchronous framework

Enable VPP TLS asynchronous framework

User private key protection

Summary

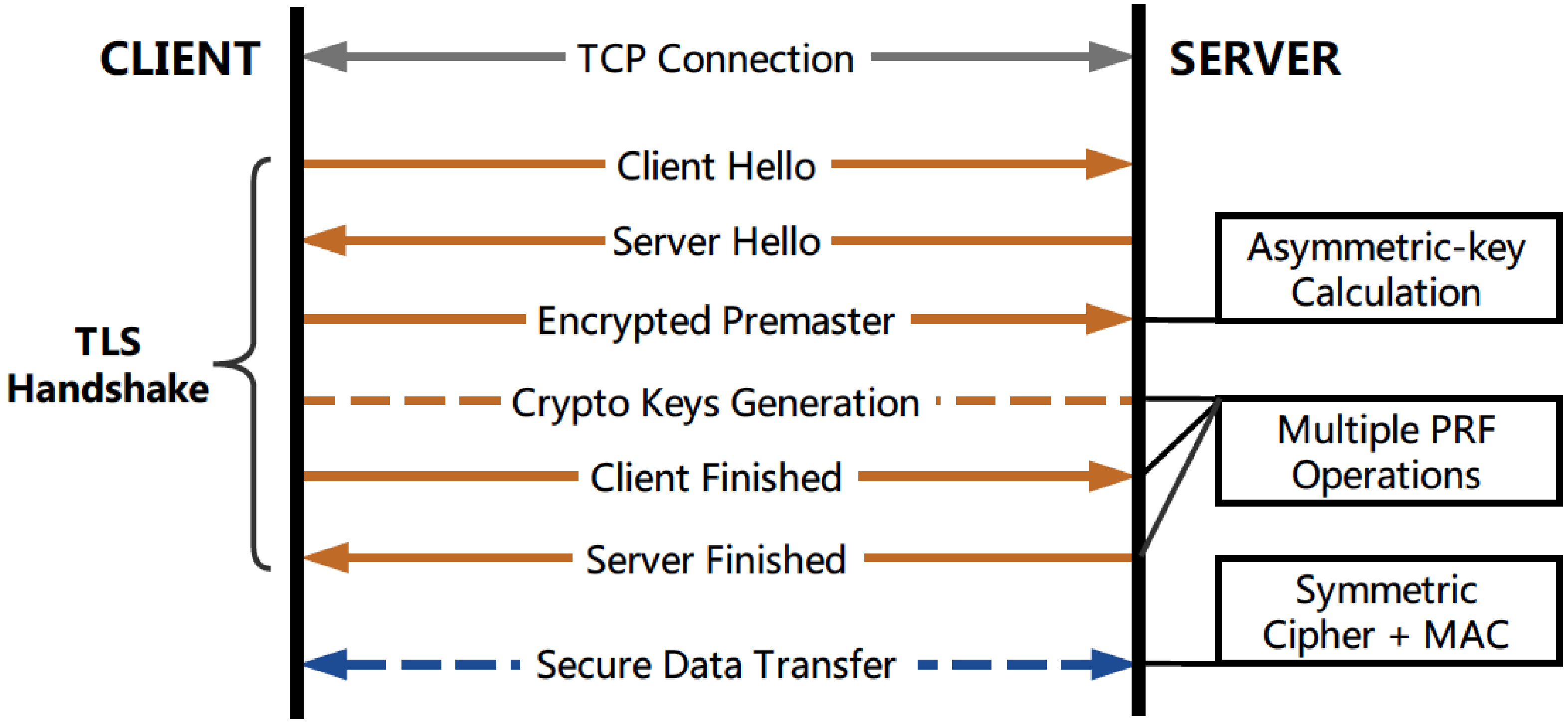
Motivation – TLS everywhere



https://transparencyreport.google.com/https/overview?hl=en

Motivation

- Computation consumption for SSL/TLS protocols
- TLS connection: (1) handshake phase and (2) secure data transfer phase



TLS Hardware Acceleration

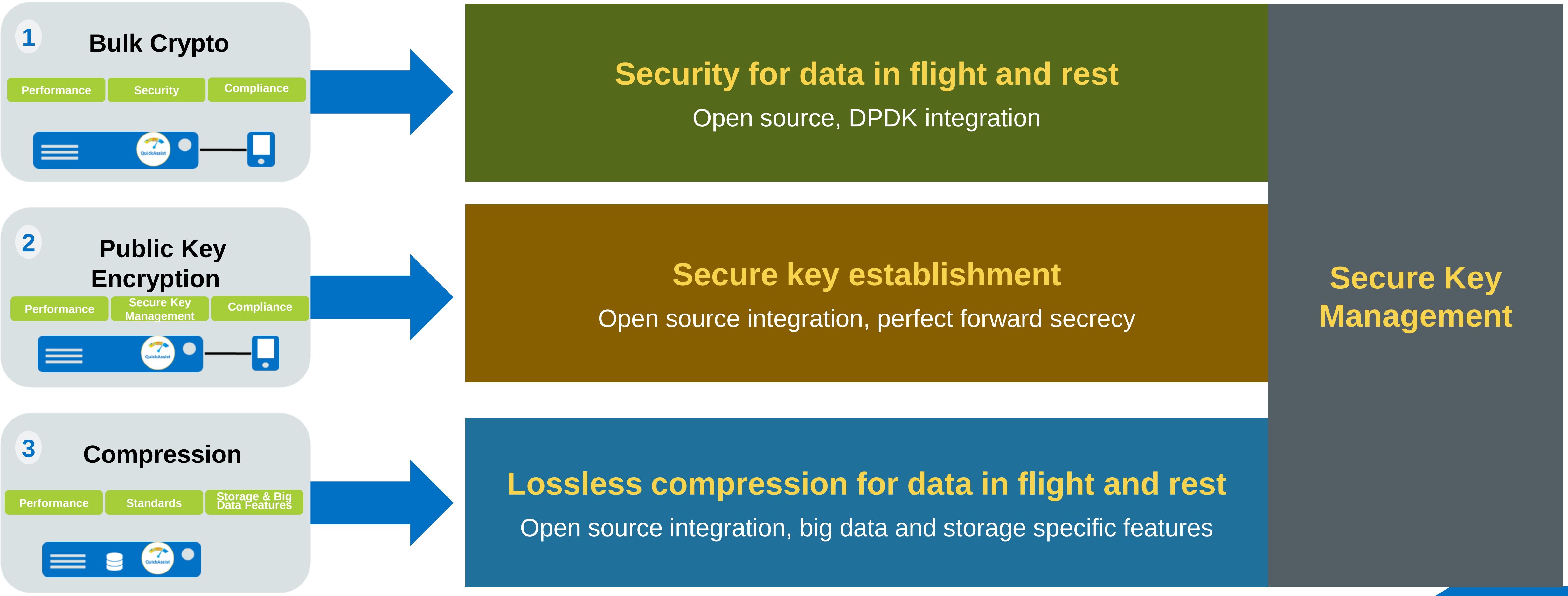
Hardware accelerators

- Specified hardware for TLS accleration
- Reduce CPU usage & Reduce Total Cost of Ownership (TCO)
- First step is to offload TLS-involved crypto jobs

Why Intel® QuickAssist Technology (QAT)?

Intel® QuickAssist Technology

Designed to optimize the use and deployment of crypto and compression hardware accelerators



1 Bulk Crypto

Performance | Security | Compliance

2 Public Key Encryption

Performance | Secure Key Management | Compliance

3 Compression

Performance | Standards | Storage & Big Data Features

Security for data in flight and rest

Open source, DPDK integration

Secure key establishment

Open source integration, perfect forward secrecy

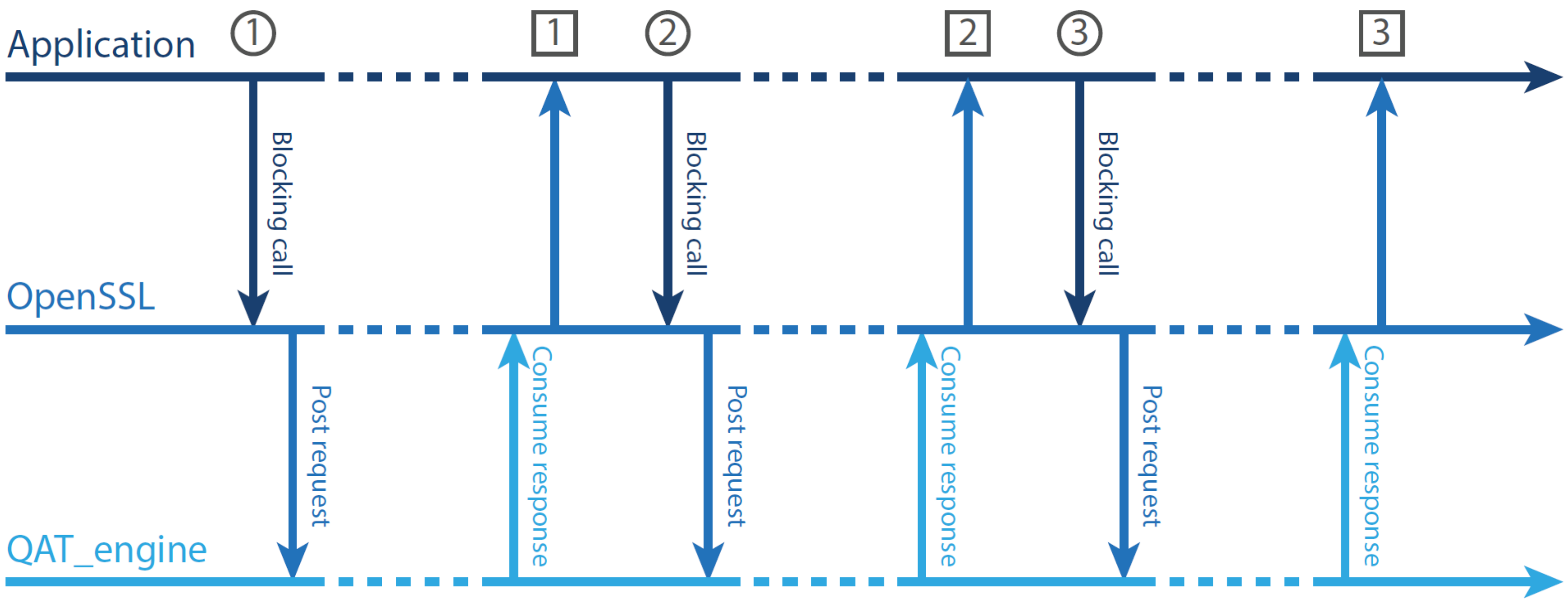
Lossless compression for data in flight and rest

Open source integration, big data and storage specific features

Secure Key Management

Background - Synchronous Mode

Synchronous Mode



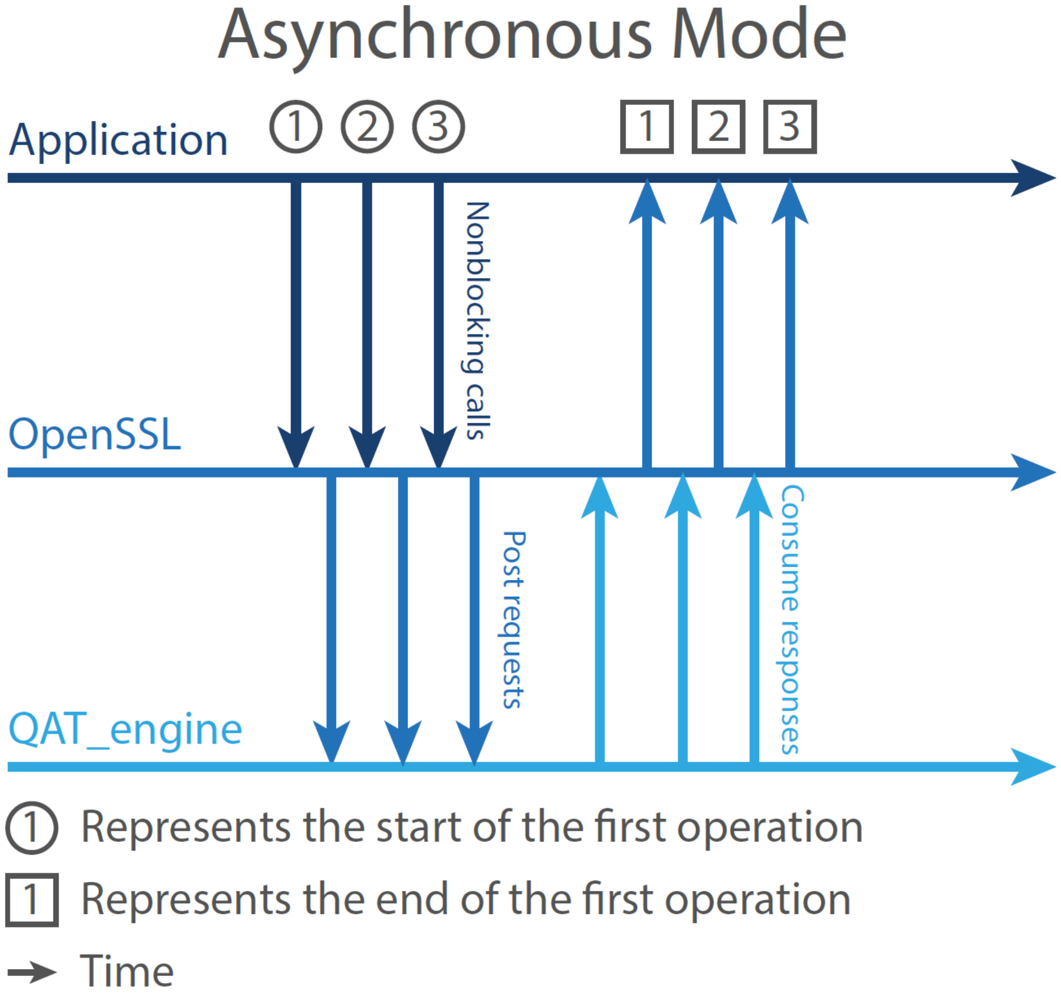
① Represents the start of the first operation

1 Represents the end of the first operation

→ Time

https://01.org/sites/default/files/downloads/intelr-quickassist-technology/intelquickassisttechnologyopensslperformance.pdf

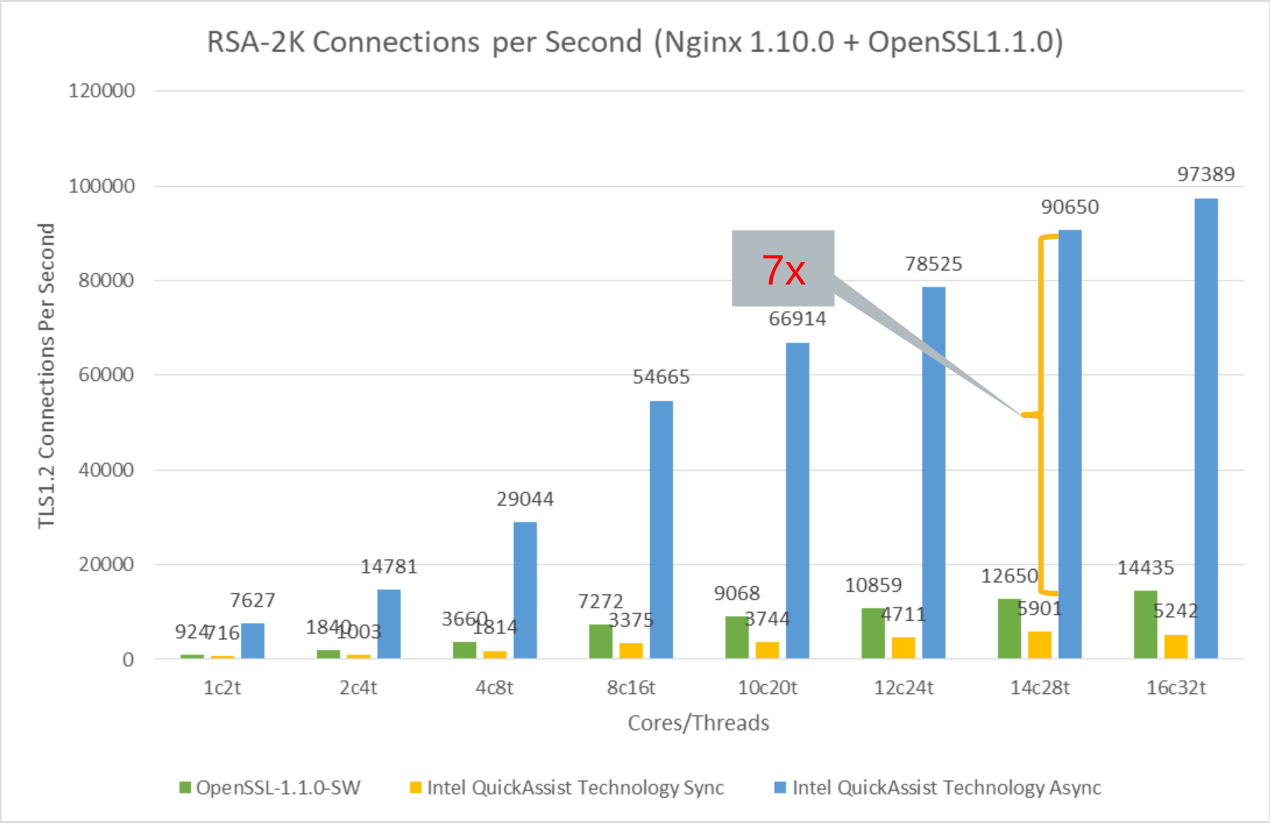
Asynchronous Mode



https://01.org/sites/default/files/downloads/intelr-quickassist-technology/intelquickassisttechnologyopensslperformance.pdf

Synchronous Mode vs Asynchronous Mode

- Performance
  - Synchronous Mode for multi-thread
  - Asynchronous Mode for single thread
- Development and Deployment
  - Synchronous Mode minimizes impact to existing software stack
  - Asynchronous Mode ecosystem

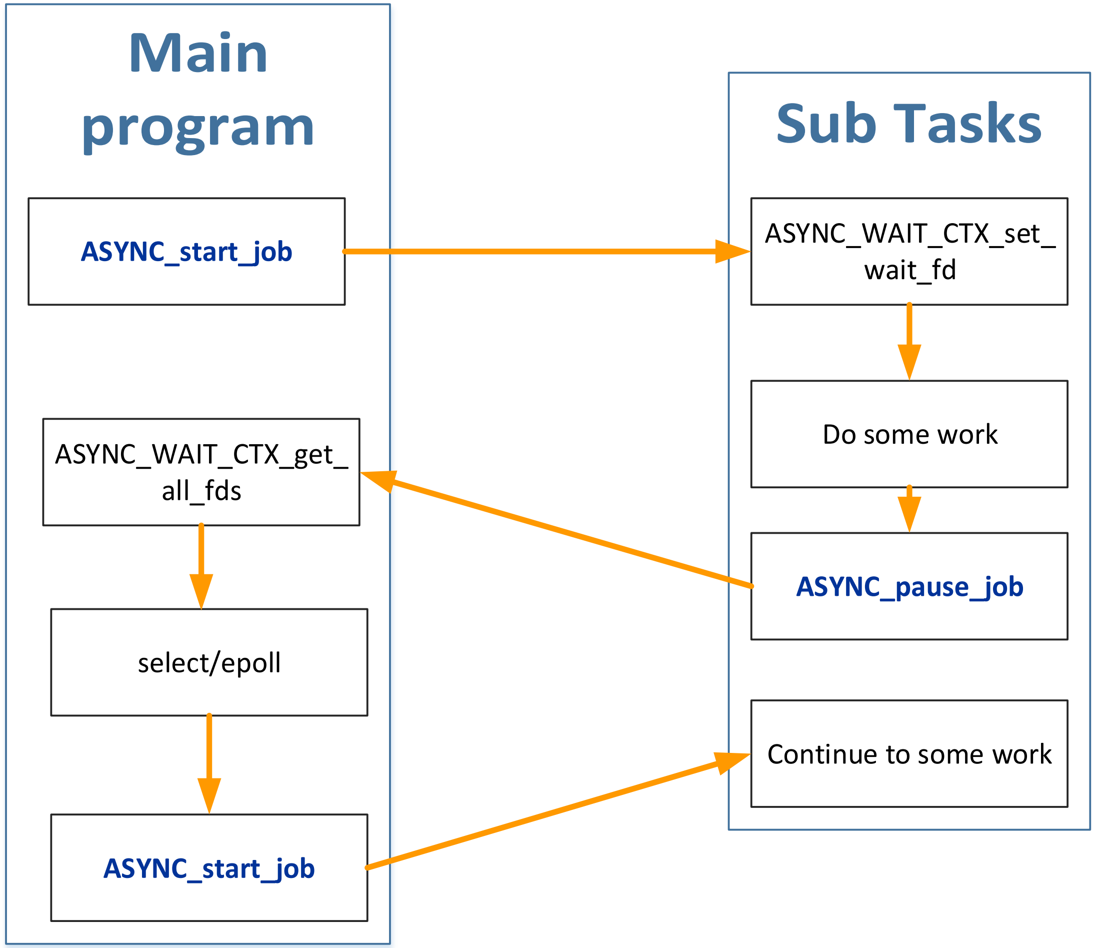


QAT boosts the performance significantly

https://01.org/sites/default/files/downloads/intelr-quickassist-technology/337003-001-intelquickassisttechnologyandopenssl-110.pdf

OpenSSL asynchronous introduction

Fiber based asynchronous mechanism



Main program

ASYNC_start_job

ASYNC_WAIT_CTX_get_all_fds

select/epoll

ASYNC_start_job

Sub Tasks

ASYNC_WAIT_CTX_set_wait_fd

Do some work

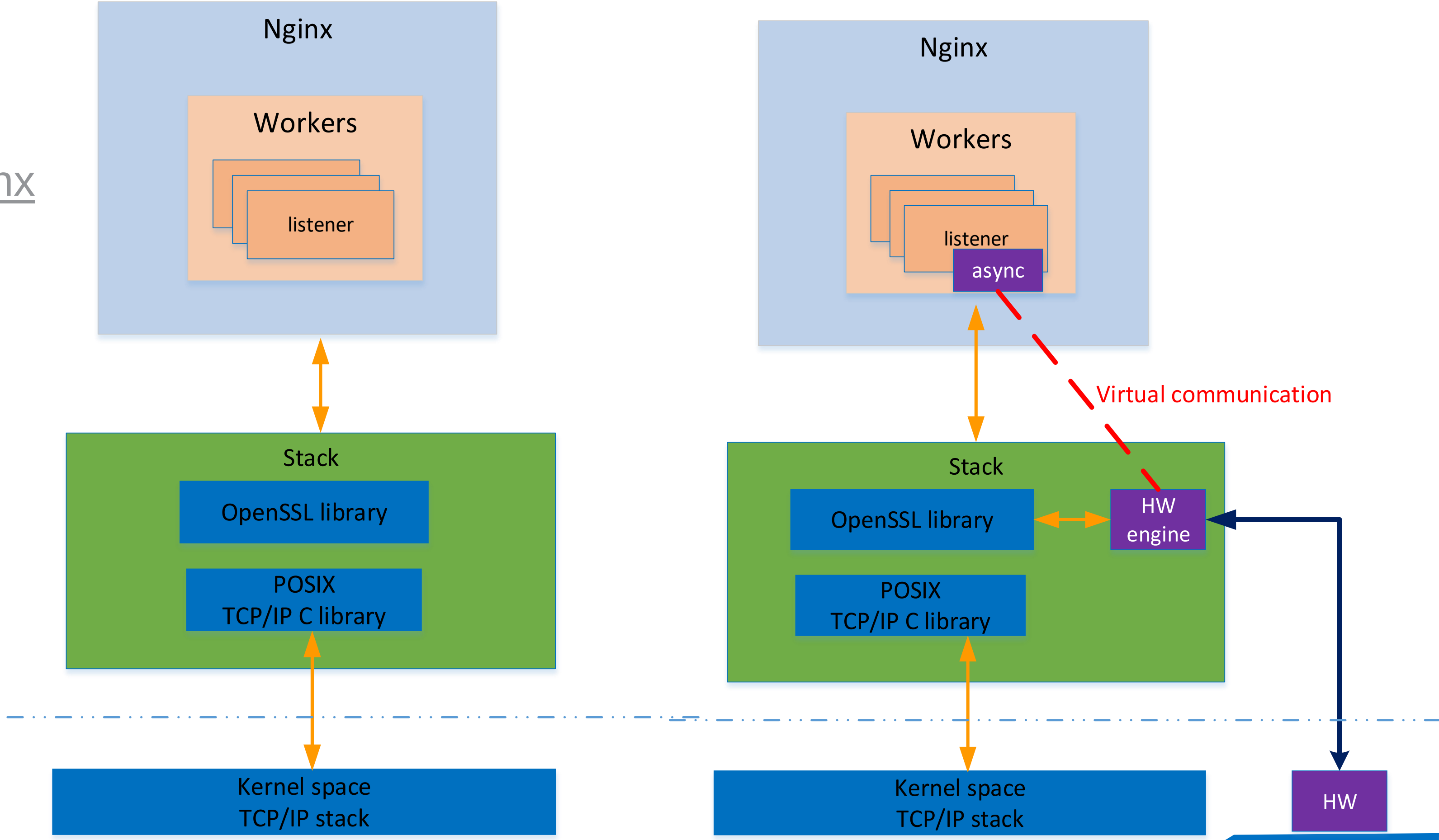
ASYNC_pause_job

Continue to some work

Apply OpenSSL asynchronous in Nginx

https://github.com/intel/asynch_mode_nginx

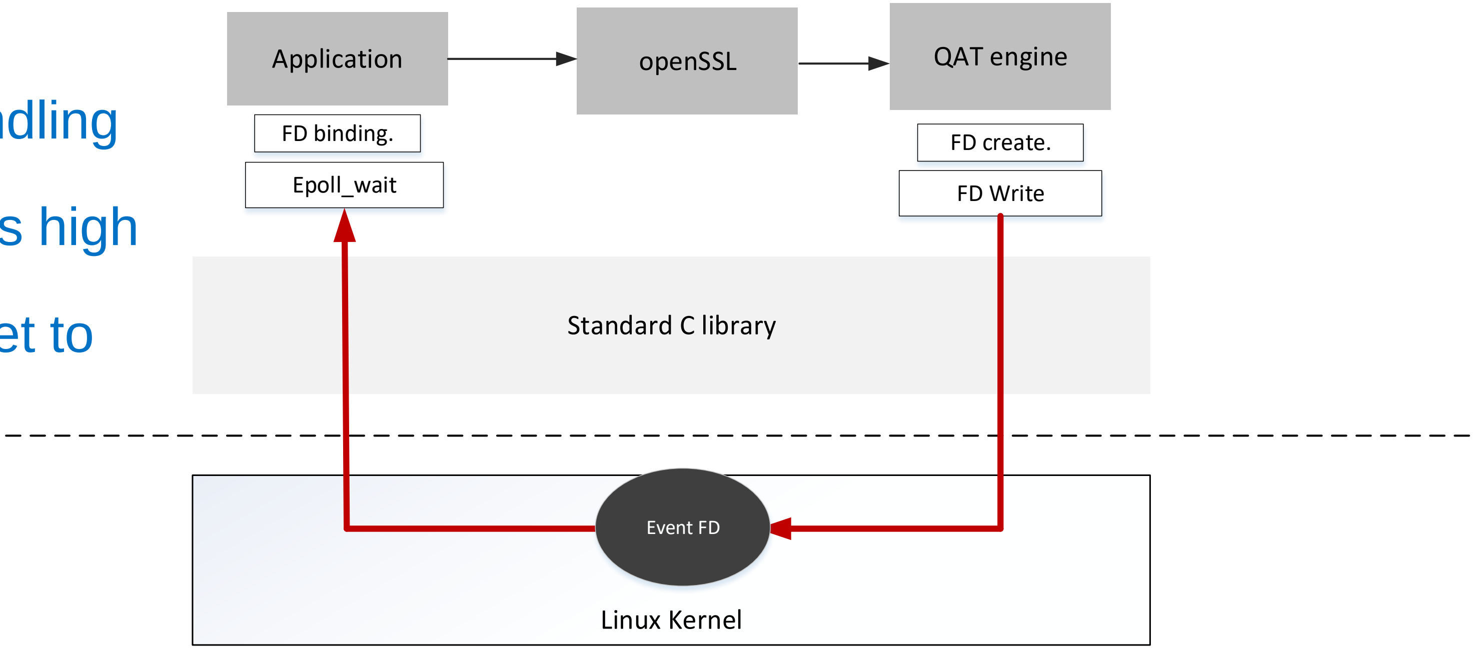
- QAT crypto request submission
- Crypto pause
- QAT response retrieval(Heuristic polling)
- Event notification
- Post processing



Event notification

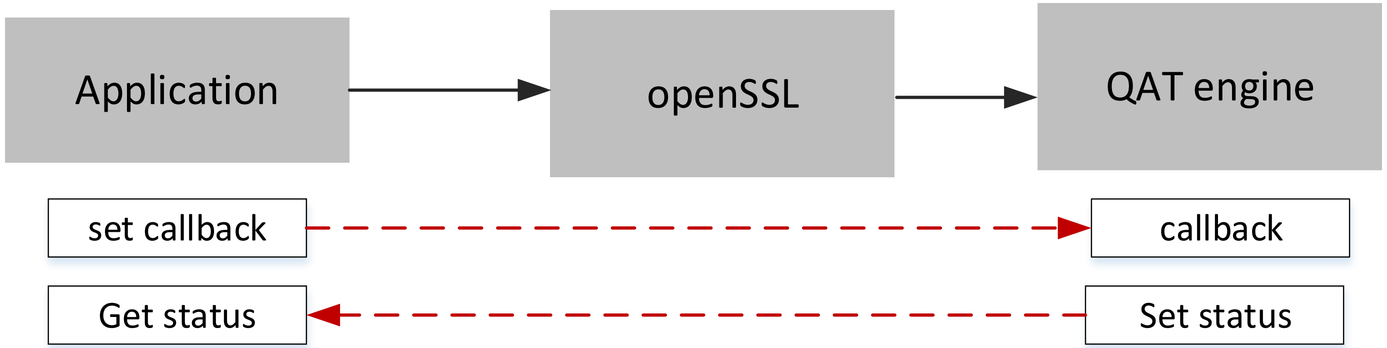
- Application and QAT engine shares the event in Kernel
- QAT engine creates event fd, and application bind it into monitoring list
- QAT engine writes events to the fd
- Application get notified and event handling
- User space and kernel space switch is high
- What if user space stack as epoll is set to

session layer function call



Enhanced callback mechanism

- Drop eventfd based solution due to the high cost
- Instead of the setup the event channel, why not call the event handler directly?
- Set the callback when the SSL session is created or even set the callback in SSL_CTX
- Allow application to set callback and get status
- Callback should be small
- OpenSSL 3.0.0 master branch



https://github.com/intel/QAT_Engine

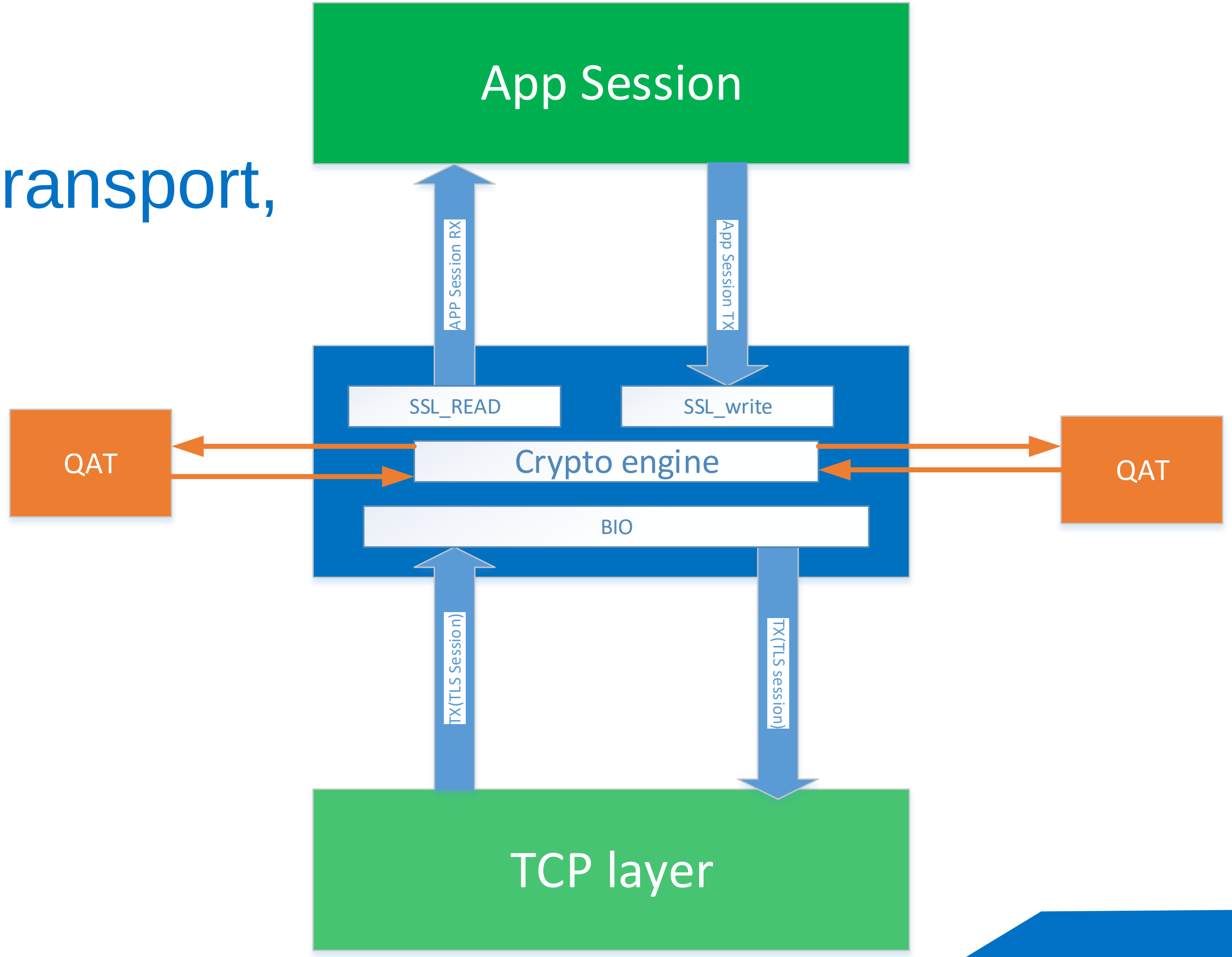
```
git clone https://gerrit.fd.io/r/vpp
```

Enable TLS support based on user space VPP stack

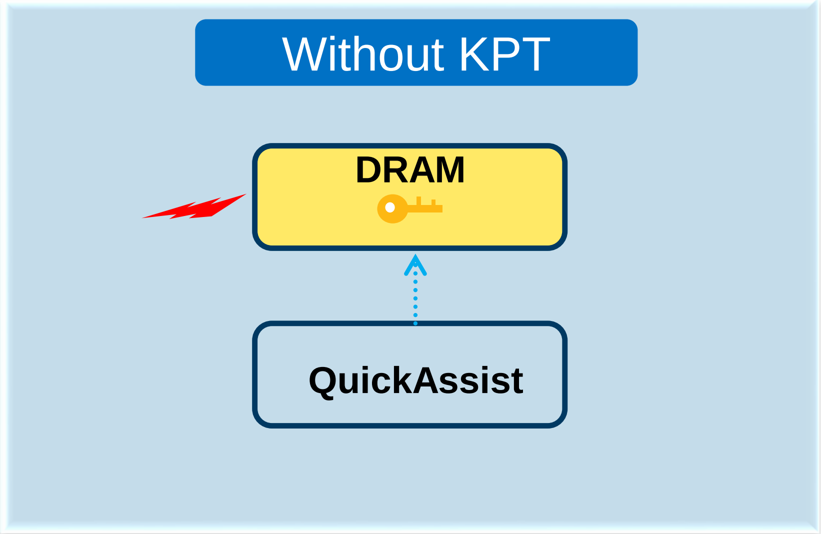
Put TLS context between session and TCP/IP transport

TLS is an application above TCP/IP transport, and it is a transport for session

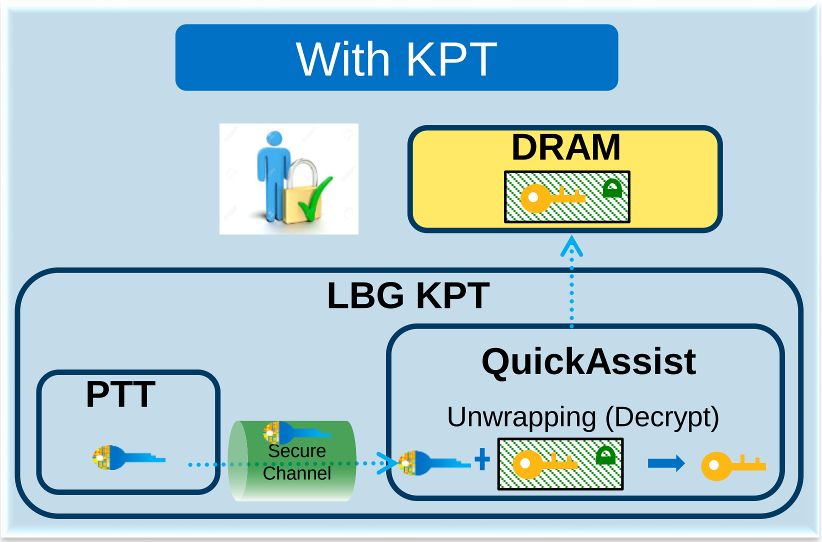
- New API
  - SSL_CTX_set_async_callback
  - SSL_CTX_set_async_callback_arg
  - SSL_set_async_callback
  - SSL_set_async_callback_arg
  - SSL_get_async_status



Key protection technology



- Private key exposes in clear text
- Key is not protected and unsafe
- Subject to attacks



- Private key is wrapped (encrypted)
- Key is protected and safe
- **NOT** subject to attacks

Private **K**ey (PK) **S**ymmetric **W**rapping **K**ey (SWK) **W**rapped **P**rivate **K**ey (WPK)

PTT: **Pl**atform **T**rust **T**echnology (firmware implementation of TPM 2.0)

Thank list

Brian Will (Intel)

Linsell Steven (Intel)

Mattcaswell (OpenSSL)

Paul Yang (Baishan Cloud)

Kinsella Ray (Intel)

Hongjun Ni(Intel)

Xiaokang Hu(Shanghai Jiaotong University)

Jokul Li(Intel)

Florin Coras (Cisco)

John DiGiglio(Intel)

Changzheng Wei (Alibaba)

intel®