What's next for the PSP Security Protocol

Panel
Netdevconf 0x1a
Rome, July 2026

(Picture: Palazzo della Civiltà Italiana, down the road from us)

What is PSP

A secure network protocol, like IPSec

Optimized for large-scale datacenter deployments

What is PSP

A secure network protocol, like IPSec

Optimized for large-scale datacenter deployments

Netdevconf 0x18: PSP Crypto Protocol and HW Offload

Why Now

- Linux kernel support from v6.18
  - and in active development: netkit, packetdrill, ..
- HW from Broadcom, Intel, NVIDIA
- Widely deployed at Google
  - intra- and inter datacenter, cloud tunneling, AI fabrics
- Open Compute Project v1 spec

SPEC, NW, OCP PSP Architecture Specification V1.0.0

OCP PSP Architecture Specification

Contributor: Google, Broadcom Inc, Intel, Meta, NVIDIA

Family: Networking

Project: IT Infrastructure > Networking

Release Year: 2026

BROADCOM
Google
intel
Meta
NVIDIA

https://www.opencompute.org/documents/ocp-psp-architecture-specification-final-v1-pdf

June 23, 2026

PSP quick recap

Scalable

- 10M+ flows, 100K+ flows/sec change
- All Rx keys derived from one device key: O(1) storage

Simple: few algorithms, few features

- No Authentication Header (AH)
- Only AES-GCM and AES-GMAC
- No Replay Protection -> L4 (TCP) already has

Datacenter optimized

- UDP encapsulated for load balancing
- Plaintext headers for telemetry (AEAD)
- HW Timestamp based IV for latency measurement

Standardize through Open Compute Project

Standard: Codify existing and new interoperable HW
Open: Convert from github.com/google

Community: Bring together invested industry
Consensus: As process for revisions

Ecosystem: Avoid fragmentation
Compatibility: Stay backward compatible as we extend

Previous Work: OCP NIC Core Features spec

Changes from Google to OCP Spec

- **Clarify**: Fix ambiguity in the text
- **Complete**: Address feature gaps
- **Expand**: Support new features (if consensus)

- **v1**: Existing Google spec + minor clarifications only
- **v2+**: Complete + expand: iterate

Panel: ideas for future work

- Multi-user security
- Cluster-mode key derivation (KDF)
- Key exchange
- Test coverage
- Confidential Compute
- SR-IOV
- FIPS 140-3 certification
- New algorithms: ChaCha20-Poly1305
- Wire format extensions: optional fields
- …

PSP Testing

PSP Support in Packetdrill

- Support PSP encap in .pkt language
- Support netlink ops in .pkt language
- Use “wire mode”, packets traverse the driver/device layer
- Do a key exchange with packetdrill wireserver
- Wireserver uses openssl to encrypt PSP packets with key from Linux stack under test

```
1   // PSP data is allowed in the ofo-queue during TX key insertion
2
3   --psp_udp_port=1000
4
5   // Initialize connection
6       0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
7      +0 setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
8      +0 bind(3, ..., ...) = 0
9      +0 listen(3, 1) = 0
10
11    +.1 < S 0:0(0) win 50000 <mss 1000,nop,wscale 0>
12     +0 > S. 0:0(0) ack 1 <mss MSS,nop,wscale 8>
13    +.1 < . 1:1(0) ack 1 win 50000
14     +0 accept(3, ..., ...) = 4
15
16     +0 psp_rx_assoc(4, [7]) = 0
17
18   +0.1 < encap psp spi 7 P. 101:501(400) ack 1 win 50000
19     +0 >  . 1:1(0) ack 1
20
21     +0 psp_tx_assoc(4, 23) = 0
22
23     +0 < encap psp spi 7 P. 1:101(100) ack 1 win 50000
24     +0 > encap psp spi 23 . 1:1(0) ack 501
25     +0 recv(4, ..., 1000, 0) = 500
26
27  // PSP encapsulated FIN handshake
28    +.1 close(4) = 0
29     +0 > encap psp spi 23 F. 1:1(0) ack 501
30    +.1 < encap psp spi 7 F. 501:501(0) ack 2 win 50000
31  +.001 > encap psp spi 23 . 2:2(0) ack 502
```

PSP netlink op

PSP encap syntax

Packetdrill: PSP test coverage

Very useful testing scenarios close to connection setup and key exchange:

- Retransmission of data queued to socket before PSP tx state is set
- Accepting PSP or cleartext data until PSP tx state is finalized
- Drop cleartext data after upgrade
- ofo queue handling
  - PSP data allowed during upgrade
  - Cleartext present will cause netlink tx assoc op to fail
- TCP Timewait sockets with PSP state

Planned: Support for tso.py and gro.py kernel selftests

Extend TCP segmentation and coalescing tests to work with PSP:

- selftests/drivers/net/hw/tso.py:
  - Do a PSP key exchange up front. The rest of the test should work the same.
  - The test cases that use tunnel drivers may have to wait for more kernel support. __ip6_tnl_rcv() scrubs skb extensions.
- selftests/drivers/net/gro.py
  - This probably looks a lot like packetdrill:
  - Pull an encryption key to use from the device we are testing
  - Give that key to the remote side, and do PSP encap and encryption with OpenSSL.
  - By the time the packet socket on the rx side sees the packets, they will be decrypted and decapsulated (and hopefully coalesced!), so test assertion logic should be the same.

Google

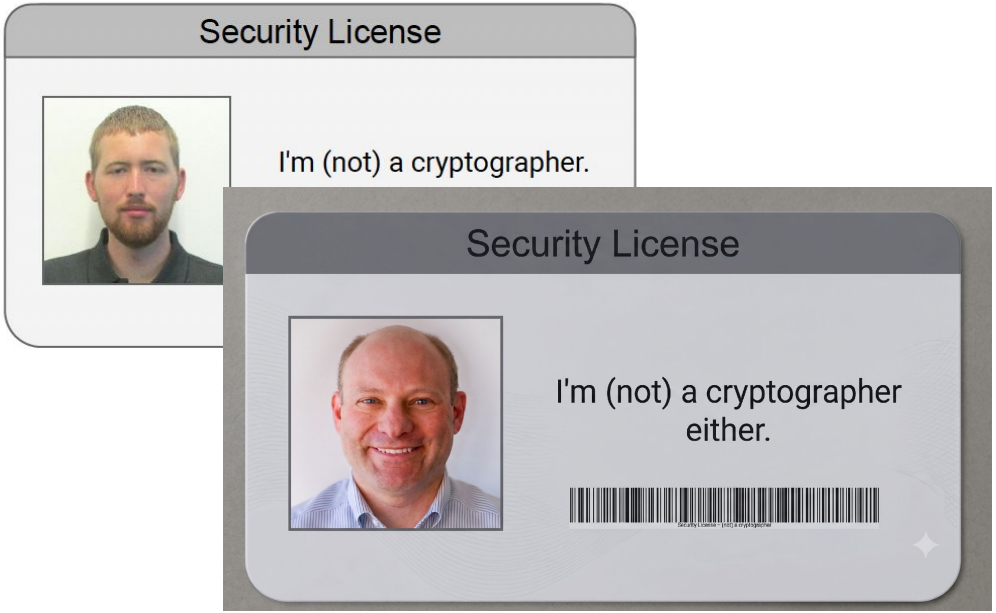
- Certification: Upgrade to FIPS 140-3 (sec 4.2.1)
- Optional integrity protection of L3 fields (daddr)
- Optional headers
  - Define TLV encoding?
  - Opt hdrs need not be VC cookie: update PSP Header figure
    - Hdr Ext Len text in Sec 5.1.2. already updated
- Balance features against the simplicity goal of PSP
  - Collectively say no to some
  - Avoid full cross-product of all optional features
- IETF Enhanced ESP
  - EESP, IKEv2 negotiation for EESP, IKEv2 for PSP, IKEv2 Group ..

Broadcom

- Cluster-mode KDF: allow NICs to share master keys
- Multi-user security
- Native mode: without UDP encap
- Key exchange: formalize IKEv2

Multi-user security/Crypto Topics

- **Crypto Topics**
  - Rekey limits under multi-user conditions
    - TLS/QUIC provided several new security features and analysis
    - Do these apply to PSP?
- **Proposals**
  1. IV construction and randomization
  2. Structured IV
  3. Automatic key generation/Replay
- Many of these topics were discussed during the development of the UEC (Ultra Ethernet Consortium) UET TSS (transport sub-layer security) UEC UET 1.0.2 specification
  - BTW: I am the editor of the UET TSS



[Jeff Andersen - Intro to Hash-based Signatures - presented to OCP security group]

Re-key limits under – TLS/QUIC Multi-user (MU)

- **TLS/QUIC Appendix B – AEAD Algorithm Analysis** RFC 9001:
  - Analysis of the limits of the AEAD crypto algorithms using various assumptions
  - Using Confidentiality Limit from theorem 4.3
  - The Multi-user Security of GCM, Revisited: Tight Bounds for Nonce Randomization (acm.org):
- Bounds to the Encryption Limits (256b key):
  - With packet size of 4096B (2^{12})→ **2^{27} packets before rekey**
- Bound for the Integrity Limit:
  - $v \leq 2^{192/l} \rightarrow$ **2^{180}**
- **Using single user security from** Limits on Authenticated Encryption Use in TLS
  - Rekey limit is **2^{34}-2^{48}** packets depending upon assumptions
- **Take away:**
  - **If MU applies, rekeying is required more often than in a single user environment**

Counter Size (bits)	Time to Wrap at 800G with a 4KB frame
2^{27}	~5.5 seconds
2^{34}	~12 minutes
2^{48}	~134 days

Multi-user Security Paper Assumptions

THE MU SECURITY OF AE. One question is what is the best we can expect from an AE scheme in terms of its mu security. To this end, BT adapt a well-known generic key-recovery attack by Biham [6] to AEAD. First, fix N^*, A^* and M^*, and obtain their encryption with respect to u different users, which yields ciphertexts

$$C_i = \mathcal{E}_{K_i}(N^*, A^*, M^*)\,, \quad i = 1, \ldots, u\,,$$

where K_i is the key of the i-th user. The attacker's goal is to recover *at least one* of the K_i's. To do so, it makes p key-guesses K (e.g., random ones), and for each guess, computes $C = \mathcal{E}_K(N^*, A^*, M^*)$. If $C = C_i$ for some i, then $K = K_i$. It is not hard to see that the probability that this attack succeeds is roughly $u \cdot p/2^k$, where k is the key-length (e.g., $k = 128$ in GCM based on 128-bit AES). Therefore, the effort to succeed is only $p \approx 2^{k-\log(u)}$.

- **Does this apply?**
  - Premise of attack is a larger number of coordinated attackers.
  - Breaking any of the session keys is a successful attack
  - To be successful many samples using a **FIXED**: Nonce (N), Additional Authentication Data (A), and Message/Packet (M) are required.

[The Multi-user Security of GCM, Revisited: Tight Bounds for Nonce Randomization (acm.org)]():

IV Constructions from Other Standards

- **TLS/QUIC/DTLS IV**
  - The 64-bit record sequence number padded to zero in network byte order
  - To address multi-user security random mask from the session secret
    - xx_write_iv = HKDF-Expand-Label(Secret, "iv", "", iv_length)
  - The padded sequence number is XORed with static xx_write_iv and used in the cipher
- **OCP PSP 1.0**
  - Specification is vague, but global time at transmitter is discussed as the IV and used directly in the cipher
- **IPSec ESP**
  - IV = Salt(4B) not transferred +Explicit IV (8B) is used directly in the cipher
    - Explicit IV is usually implemented as a strictly increasing 64-bit counter
- **UEC UET TSS**
  - Packet TSC = Epoch (16b)||counter(48b) – counter is incremented for every packet sent
  - To address multi-user security random IV mask IVMASK is derived from the KDF or stored with key
  - IV = IVMASK XOR TSC is used in cipher

Proposal 1: Add IV Randomization to Address MU Security

- **Call to action**
  - **Cryptographers, PLEASE comment on multi-user security assumption and if they apply to PSP and rekeying/randomization solution**

- **IV Randomization Proposal for PSP 2.0**
  - Define a new cipher version to maintain backward compatibility
  - Add an additional KDF round to generate a random MASK to XOR with the wire IV
  - No additional wire bandwidth is required

Proposal 2: Structure the IV

- **The current IV text in OCP PSP 1.0 is vague**
  - Structured IV is compatible with current spec 1.0

- **Allow Structuring IV into a fixed and counter portions per** RFC 5116: An Interface and Algorithms for Authenticated Encryption (rfc-editor.org)
  - IV: epoch(16b)||counter(48b)
    - Fixed portion is an epoch counter used to address station reboot.
    - Counter portion monotonically increments for every packet:
      - Easier to manage than a time-based solution
      - Reset to zero on rekey (AN change)

Proposal 3: Automatic Rekey and Replay Protection Proposal

- **Automatic Rekey can address address Multi-user security**
  - Using a portion of the IV, the symmetric key can be generated more frequently using a KDF.
  - This improves nonce properties and key invoke limits but not provide replay protection.
  - A dedicated epoch field can be used to address station reboot.

- **Replay Protection**
  - PSP leaves this to upper layers but it is important for a robust solutions
    - Structuring the IV allows for a simple implementation at upper layers
  - Local rejection (IV based)
    - Reject anything not in the same epoch and IV is older than # packets.
  - Global Epoch Rejection
    - Reject packets not the same epoch

What Is Cluster Mode PSP?

- AI and HPC use case are typically classified as Distributed Communication Applications
- These applications have a different security model than traditional client-server applications
  - Specifically isolation between ranks within a job is not required
  - Cluster mode PSP extends PSP for these applications

Existing PSP (client-server)

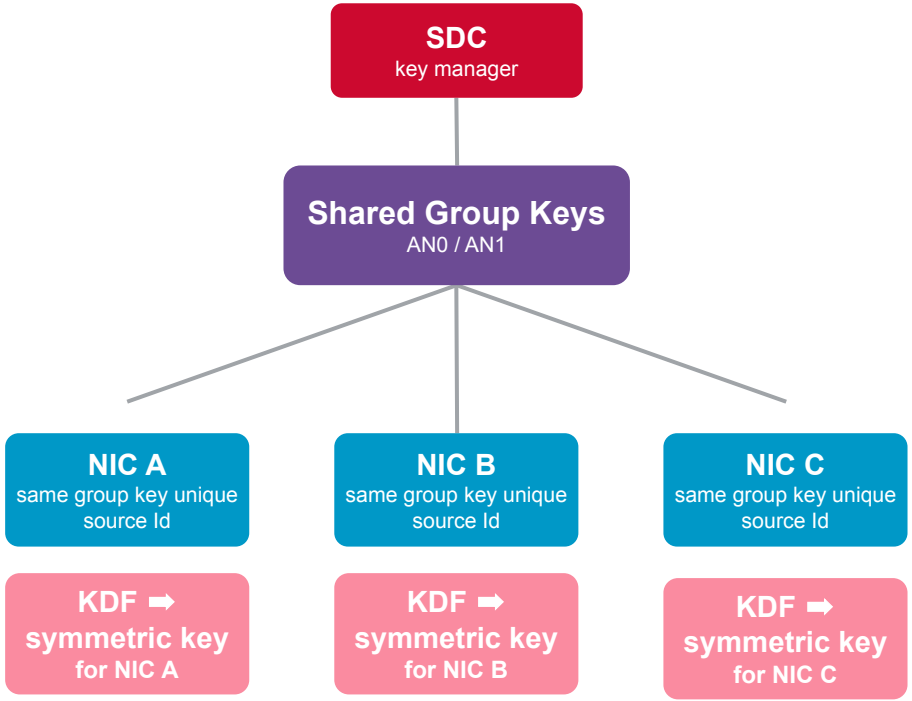
- Point-to-point security associations
- A separate derived key for every peer session
- KDF runs on the receive path only
- Each NIC holds its own private keys

Cluster Mode PSP

- Shared group key across all ranks in a job
- **KDF used to generate source specific key at both Source and Destination**
- Key storage is independent of the number of ranks
- No key-exchange handshake between peers
- Unchanged PSP header with a new version 10

Cluster Mode PSP Architecture

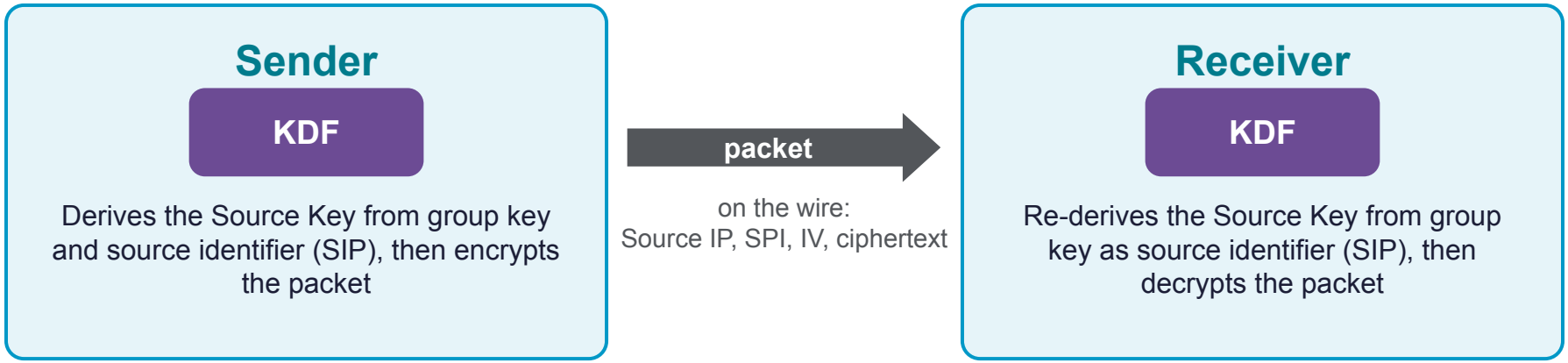
- **Secure Domain (SD):** a group of NIC endpoints that all share a group key.
  - Selected using existing SPI
- **SDC** (Secure Domain Controller) generates and distributes group keys over a secure channel.
- Each SD holds two group keys (AN0 / AN1) for hitless key rotation, identified with the upper bit of the SPI.
- Any member/rank can securely reach any other member — No per-peer setup!



Every NIC in the domain holds the same master keys.

Source Key Derivation

Both endpoints use the same KDF and the group key to derive a per source key — no handshake needed.



The identical source specific key is derived independently.
No key exchange required for new ranks/members

Miscellaneous Topics

- Key Exchange for PSP
  - In our opinion, a scale key exchange protocol is need to make PSP widely deployable
  - Ideas?
    - Extension to TLS/QUIC seems like a good choice.
- Native Encapsulation
  - The next header field in PSP header uses same encoding as IPv6 protocol field
    - Space is controlled by IANA and very difficult to add new protocols (good thing)
  - Encapsulation requires a IP protocol (usually UDP with a source port)
    - e.g. {Eth, IP, UDP, PSP(next-header=UDP), UDP, RoCE}
  - Proposal: Add a PSP namespace indicator to the PSP header and protocol field all for PSP to define native encapsulation
    - E.g. {Eth, IP, UDP, PSP (lcl-ns, next-header=RoCE), RoCE}
- PSP Version issues
  - Currently the PSP Version defines the protocol version AND the crypto algorithm
  - Break this apart and have separate header fields for Standard vs Cluster mode, Crypto algorithm, etc.

PSP workshop

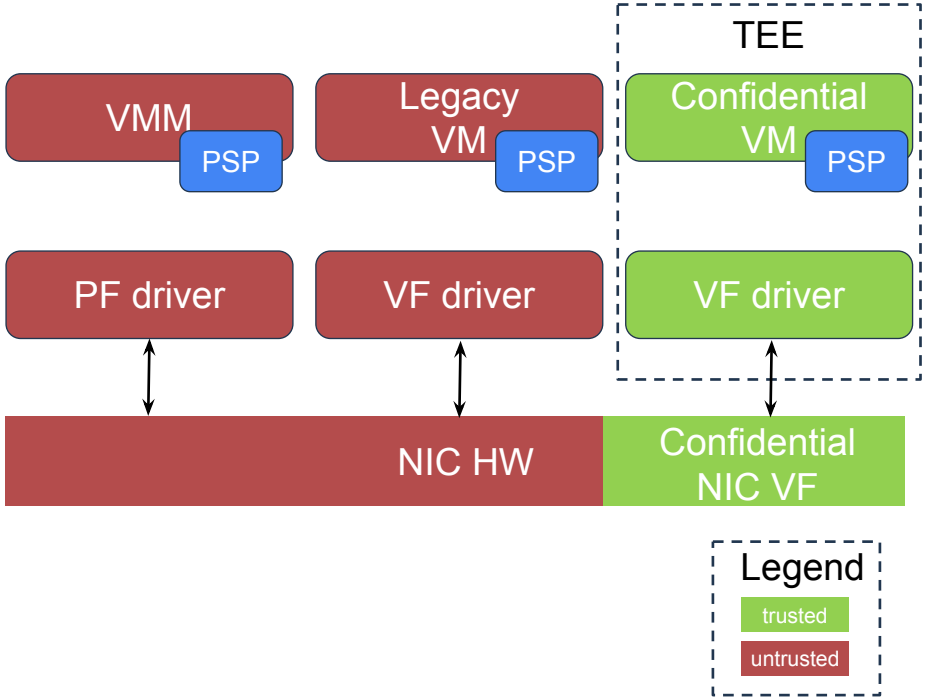
Boris Pismenny

netdev0x1A

Confidential compute virtual machines

Key PSP SRIOV use-case

- VMs must control their own encryption
- VMs trust the NIC via TDISP
- VMs distrust the hypervisor



PSP in SRIOV

Unique master key per virtual function

- All SPIs are available to each VM
- Master rekey impact only the VM
- Compatible with confidential compute
- No new APIs

Shared master key with dedicated SPI space per VM

- Limited SPI space per VM increases rekey frequency
- VMs share fate upon rekey
- Incompatible with confidential compute
- New APIs required to notify on rekey and split SPIs

Psp Evolution

1. PSP wire protocol aka dataplane.
2. On purpose control plane in not kn the spec.
3. Suggestion is to add an appendix to give an example in may be pseudocode to describe the control plane.
4. Appendix : assumptions on the device side for scalability, programmability, configurability and evolution and interoperability.
5. PSP pseudo code model, white model for getting it right in multiple hw devices.
6. Tests for conforming to a version of the spec.
7. And concepts that should be abstracted by the host side driver for live migratability.
8. Options on rate of programming vs memory foot print.

Anjali Singhai Jain

Q&A&Ideas

Backup

Changes from Google spec to v1

- Clarify IV construction uses (SPI | timestamp) concatenation
- Clarify label and context field separator in key derivation
- Clarify HdrExtLen constraints. Limit to range [0, 63].
- Add Appendix B: Example Packets
- Add section numbering

Key mgmt: scalability challenges

1. Storage: 10M flows x 256B x {tx, rx} = 5GB

1. Latency: generation, insertion, deletion tail latency

Key mgmt: stateless offloads

- Rx: derive session key at linerate
  - On-device hidden master key
  - Encrypt counter to derive session key (KDF-CM)
  - Per-packet IV: counter (SPI) plus timestamp

- Tx:
  - Key in descriptor

OCP PSP Members

Broadcom: Eric Davis, Eric Spada

Google: Stephen Anderson aka smanders, Willem de Bruijn

Intel: Anjali Singhai, Stephen Doyle

Meta: Daniel Zahka, Jakub Kicinski

Nvidia: Boris Pismenny, Saeed Mahameed