软件定义边界（SDP）标准规范2.0

（试读本）

Software-Defined Perimeter (SDP) Specification 2.0

Software-Defined Perimeter (SDP) Specification 2.0

June TBD

The permanent and official location for Software Defined Perimeter Working Group is
https://cloudsecurityalliance.org/research/working-groups/software-defined-perimeter

0.1 Document Project Plan

Start Date	End Date	
Feb 15, 2019		Start
		Agree Outline /Assign Sections
		Revised Outlines /Assign Sections and Writing
		Writing
		Writing/Review - Extension
		Writing/Review - Extension
		External Peer Review
		Marketing Publishing

0.1.5 To Do's / Assignments

0.2 Team / Contributor Composition

Contributors	Areas of Contribution
Juanita Koilpillai jkoilpillai@waverleylabs.com	Entire Initial v2 - Initial review and reorganization of entire v1 document to start v2 SDP Component descriptions, SDP Protocol section updates, Updated diagrams, JSON edits, Onboarding example. Entire document - Made and accepted edits and minor rewrites throughout.
Jason Garbis jason.garbis@appgate.com	SDP Deployment models and Workflow table changes. SPA - broader usage section rework mTLS and IKE section
Michael Roza Michael.e.roza@gmail.com	Entire Initial v2 - Initial review and reorganization of entire v1 document to start v2, SDP Protocol section - Identification of errors, inconsistencies, and recommendations for improvement and changes to sequencing images and message text. Summary section - outline. SDP Deployment models and Workflow table changes Entire document - Made and accepted edits and minor rewrites throughout.
Bob Flores bob.flores@applicology.com	Initial review and reorganization to start v2
Junaid Islam junaid@xqmsg.com	Initial review and reorganization to start v2
Daniel Bailey dbailey@waverleylabs.com	SDP Component descriptions. SDP Protocol section and workflow. SPA clarification. Onboarding example.
Benfeng Chen bfchen@clouddeep.ai	SDP Protocol and SPA section update. Updated the SDP protocol workflow for network invisibility, as well as the cryptographic algorithms in SPA messages for security.
Eitan Bremler eitan.bremler@safe-t.com	Review of SDP architecture and components, Controller, Initiating Hosts, Accepting Hosts, Gateways, Deployment Models
Ahmed Refaey Hussein **ahmed.hussein@manhattan.edu**	SDP - SDN - NFV and cloud deployments

Acknowledgments

Version 2.0

Lead Authors

Juanita Koilpillai
Jason Garbis

Contributors

Junaid Islam
Bob Flores
Daniel Bailey
Benfeng Chen
Eitan Bremler
Michael Roza

CSA Analysts

Shamun Mahmud

Version 1.0 Contributors

Brent Bilger, Alan Boehme, Bob Flores, Zvi Guterman, Mark Hoover, Michaela Iorga, Junaid Islam, Marc Kolenko, Juanita Koilpillai, Gabor Lengyel, Gram Ludlow, Ted Schroeder and Jeff Schweitzer

CSA Analysts

Shamun Mahmud

The Software-Defined Perimeter (SDP) and Zero Trust Working Group is a Cloud Security Alliance (CSA) a research working group will advocate for and promote the adoption of Zero Trust security principles, providing practical and technically sound guidance on how organizations can and should approach this for their cloud and non-cloud environments. This group will build on and leverage the NIST Zero Trust research and approach. The group will also promote SDP as a recommended architecture for achieving Zero Trust benefits and principles. It will revise and expand the SDP specification, to capture and codify the knowledge gained from experience.
While promoting and recommending SDP, the group will take an inclusive approach to alternative security architectures and objectively support them as long as they're aligned with the Zero Trust philosophy.

Table of Contents

Acknowledgments **4**

Introduction **7**

Purpose 7

Scope 7

Audience 7

Design Objectives **8**

SDP Concept 8

SDP Architecture and Components 9

SDP Controller 10

SDP Initiating Hosts (and SDP Clients) 11

SDP Accepting Hosts (and Services) 12

SDP Gateways 12

SDP Deployment Models 13

SDP Workflow 14

Onboarding 15

Single Packet Authorization (SPA) 16

SPA Message Format 17

SPA as a Secure, Self-Contained, Connectionless Message Transmission Protocol 18

SDP in relation to SDN & NFV 18

Mutual Transport Layer Authentication Between Components 19

Device Validation 20

SDP Protocol **21**

AH-Controller Protocol 21

AH to Controller Sequence Diagram 21

a. SPA 22

b. Login Request Message 22

c. Login Response Message 22

d. Logout Request Message 23

e. Keep-Alive Message 23

f. AH Service Messages 23

g. Reserved for Private Use 23

IH-Controller Protocol 23

IH to Controller Sequence Diagram 24

a. SPA 24

b. Open TCP Connection 25

c. Login Request Message 25

d. Login Response Message 25

e. Keep-Alive Message 25
h. IH Service Message 25
i. IH Authentication Message 26
j. Reserved for Private Use 26
IH-AH Protocol 26
IH to AH Sequence Diagram 27
a. SPA 27
b. Open TCP Connection 28
k. Open Connection Request Message 28
l. Open Connection Type 28
m. Open Connection Response Message 28
n. Data Message 28
o. Connection Closed Message 28
p. User-Defined Message 29
Logging 29
Fields of a log message 29
Operations Logs 29
Security/Connection Logs 31
Summary 32
References 33

Introduction

The Software-Defined Perimeter (SDP) architecture provides the ability to dynamically deploy network security perimeter functionality where needed in order to isolate applications and services deployed and accessed on unsecured networks. SDP is designed to provide an on-demand, dynamically provisioned, trusted overlay network isolated to mitigate network-based attacks. An SDP implementation hides assets from unauthorized entities, establishes trust prior to allowing connections, and manages the system via separate control and data planes. With SDP, organizations can achieve the goals of Zero Trust, improving their security effectiveness and resiliency by moving away from traditional failed perimeter-centric security models.

Purpose

This document updates the Cloud Security Alliance (CSA) Software-Defined Perimeter (SDP) Specification. Version 1.0 was written by the Software Defined Perimeter Working Group and published in April 2014.

We believe the original specification was sound and provided a solid architectural and conceptual foundation for securing connectivity. However, it was largely silent on several areas, including SDP access authorization policies, onboarding, and securing non-person entities. In addition, in recent years, the information security industry has embraced the principles espoused in the SDP architecture, which are included in the broad trend of Zero Trust. SDP enhances Zero Trust implementations. This revised version of the SDP specification is expanded and enhanced to include additions, clarifications, and extensions.

Note that this version builds on additional documentation that the working group has published since version 1, specifically the SDP Glossary and the SDP Architecture Guide. Links to both of these documents are included in the References section of this document.

Scope

This document specifies the architectural components, interactions, and basic secure communications protocol for Software-Defined Perimeter (SDP) implementations. It focuses on the control plane interactions to enable secure connectivity into the secure perimeter and a data plane describing the enforcement of secure connectivity between initiating hosts (servers, user devices, services) and accepting hosts (servers, devices, services).

Audience

The target audiences for this document are

- Solution architects and security leaders who are deploying Zero Trust and SDP products in their enterprise
- Vendors or technology providers who are implementing Zero Trust with an SDP architecture within their products or solutions

Design Objectives

SDP aims to give network providers and application owners the ability to deploy dynamic (“software-defined”) perimeters to blacken networks and prevent unauthorized access to the services running on them. SDPs replace physical appliances with logical components that operate under the organization’s control which shrinks the logical perimeter to its bare minimum. SDPs enforce the Zero Trust principle of least privilege, providing access to resources only after device attestation and identity verification, according to access policies.

The design objective of SDP is to provide an effective and readily integrated security architecture for IPv4 and IPv6, including obfuscation and access control of the Control Plane elements and the resources protected by the perimeter, as well as the confidentiality and integrity of communication in the Data Plane from the initiating source to the control plane and the accepting source. The design includes a need-to-know access model that requires authenticated identities (users) on validated devices to sign in to the perimeter cryptographically. It also blackens assets while using public infrastructures like the Internet.

SDP’s design provides seamless integration through several layers – integrating security for users, their equipment, networks, and devices. SDP can be used to secure connections over <u>any</u> IP infrastructure, whether traditional hardware-based, a newer software-defined network (SDN), or cloud-based. SDP's core capability is that it operates largely as an encrypted overlay for any type of IP network. It normalizes security layers across heterogeneous environments, simplifying security and operations.

For cloud infrastructures, SDP integrates security at:

- the network layer where virtualization[1] provides computing, storage, and monitoring
- the transport layer where cloud APIs tie virtualized assets to resource pools and users
- the session layer where the underlying virtualized infrastructure is managed
- the network access layer where middleware manages application tiers and the application
- the application layer that provides business value to users

As a complement to SDN and NFV, SDP can secure connections over the IP infrastructure SDNs create. In fact, SDP operates largely as an encrypted overlay for any type of IP network. In this sense, it acts as a homogenizing security layer, simplifying security and operations.

Based upon feedback and lessons learned from implementations of SDP, this updated specification helps to clarify access control within the various deployment models defined in the original specification[2].

SDP provides an effective alternative for stopping all network-based and cross-domain attacks on applications and infrastructures requiring invisibility while leveraging public infrastructures like the Internet and the cloud and ‘need-to-know’ access models for secure communications. While existing cybersecurity solutions focus on securing networks and systems, SDP focuses on securing connectivity and stopping attacks such as DDoS, credential theft etc.

SDP Concept

SDP provides application and enterprise resource owners the ability to deploy perimeter functionality where

[1] See paper on SDP and Network Function Virtualization (NFV) - https://www.waverleylabs.com/resources/publications/
[2] https://cloudsecurityalliance.org/artifacts/sdp-architecture-guide-v2/

needed to:

- Securely deploy services onto networks presumed to be compromised.
- Provide remote identities with precisely controlled access to those services across untrusted networks.

SDPs replace perimeter-based and (often, physical) appliances with logical components that operate under the control of the application owner. SDPs provide access to application infrastructure only after device attestation and identity verification via the evaluation of access policies.

The principles behind SDPs are not entirely new. Multiple organizations within the Department of Defense (DoD) and Intelligence Community (IC) have implemented similar network architectures based on authentication and authorization prior to network access. Typically used in classified or “high-side” networks (as defined by the DoD), every server is hidden behind a remote access gateway appliance to which a user must authenticate before visibility of authorized services is available and access is provided. SDPs leverage the logical model used in classified networks and incorporate that model into standard workflows (Section 2.4). In addition, security leaders in the industry have been espousing many of these concepts, most notably starting with the Jericho Forum in 2004. More recently, Zero Trust, as defined by the US National Institute for Standards and Technology (NIST), incorporates these principles as well[3]

SDPs maintain the benefits of the need-to-know model described above but eliminate the disadvantages of requiring a remote access gateway appliance. SDPs require endpoints to authenticate and be authorized first before obtaining network access to protected servers and services. Then, encrypted connections are created in real-time between requesting systems and application infrastructure.

In short, SDPs cryptographically sign resources (users, devices, services) into the perimeter within which services remain hidden to all unauthorized access.

SDP Architecture and Components

In its simplest form, SDP consists of two logical components: SDP Hosts and SDP Controllers. SDP Hosts can either initiate connections or accept connections. These actions are managed by interactions with the SDP Controllers via a secure channel over a control plane. Data is communicated over a separate secure channel in the data plane. Thus, in SDPs, the control plane is separated from the data plane to enable an architecturally flexible and highly scalable system. In addition, all of the components can be redundant for scale or uptime purposes. SDP hosts (initiating hosts or accepting hosts) connect to the SDP controller, which is an appliance or process that ensures users are authenticated and authorized, devices are validated, secure communications are established, and user and management traffic on a network remain separate.

[3] See the NIST Zero Trust Architecture document - SP 800-207 https://csrc.nist.gov/publications/detail/sp/800-207/final

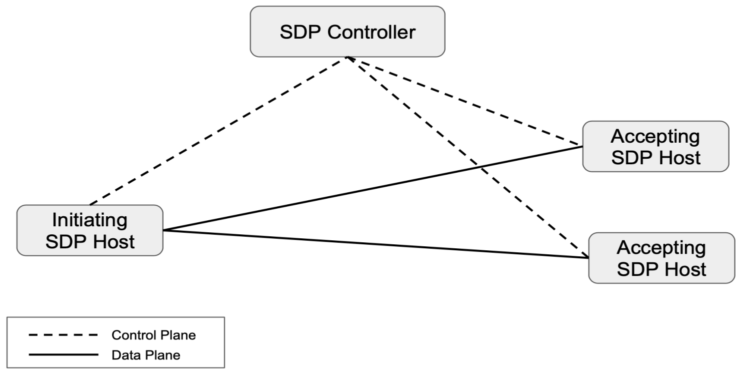


Figure 1: SDP Architecture (previously published by CSA in Software Defined Perimeter and Zero Trust)[4]

The architecture of the SDP consists of the following components:

- **SDP Controller** - this component is designed to manage all authentication and access flows. It is essentially the "brain" of the solution where access policies are defined and evaluated. It acts as a Zero Trust Policy Decision Point (PDP). The SDP Controller is also responsible for connecting to the authentication solutions (Identity providers, multi-factor authentication, etc.) to orchestrate the user's authentication and authorization.
- **Initiating Hosts (IH) on which SDP Users typically reside** - these accessing entities can be users, and NPE's (Non-Person Entities[5]) such hardware (end-user devices, or servers), network devices (for connecting networks), software applications, or services. An SDP user can use an SDP client or browser to connect.
- **Accepting Hosts (on which an SDP Services typically resides)** - these are applications or resources being accessed and protected by the SDP. In the Client-to-Gateway SDP model[6]. For example, the Accepting Host takes the form of an SDP Gateway that typically sits in front of the resources being accessed.

All of the components can be located in various locations (internet, cloud, on-premises) and can be deployed in a redundant architecture for scale or uptime purposes.

SDP is a connection-oriented protocol, securing network deployment topologies that include connections between Client to Server, Client to Gateway, Client to Server to Client, Gateway to Gateway, and Server to Server.. Details on the various deployment connection models and architecture are introduced in Figure 3 below and are detailed in the SDP Architecture Guide. It shows the multiple configurations of the SDP Gateway within each of the deployment models.

SDP Controller

The SDP Controller is a policy definition, verification, and decision mechanism (a Zero Trust Policy Decision Point) that maintains information about which users/groups from which devices have access and

[4] https://cloudsecurityalliance.org/artifacts/software-defined-perimeter-and-zero-trust/ page 6

[5] A **non-person entity** (NPE) is an **entity** with a digital identity that acts in cyberspace, but is not a **human** actor. This can include hardware devices, software applications, and information artifacts.

[6] See SDP Architecture guide for reference to the various SDP models.

permission to which organization’s applications (on-premises or in the cloud). Depending on the deployment model, it determines which SDP Hosts can communicate with each other.

Once a user (on an IH) connects to the Controller, the Controller authenticates the user and grants access only to the services (via the AHs) allowed to the user based on their context (including identity and device attributes).

In order to authenticate the user, the Controller may perform the authentication using an internal user table or connect to a third-party Identity and Access Management (IAM) service, Multi-factor Authentication (MFA), or Identity Solution (on-premises or in the cloud). Authentication is typically done based on user type and identity; for example, employees may be authenticated via an Identity Provider, while contractors may be authenticated by credentials stored in a database.

In order to authorize the user access to a service, the Controller may use an internal user-to-service mapping or connection to a third-party service such as LDAP, Active Directory, or authorization solution (on-premises or in the cloud). Authorization is typically done by user roles to the services the user can access but can be more fine-grained, based on user or device attributes, or even the actual data element or data flow that the user is authorized to access (e.g. OAuth or WebAuthN). In effect, the access policies maintained by the SDP Controller can be informed by other organizational constructs such as enterprise service directories and identity stores. In this fashion, the Controller is enforcing the type of dynamic Zero Trust policies that NIST identifies as a Zero Trust tenet.

In addition, the Controller can obtain information from external services, such as geo-location or host checking services, to further validate the user (on an IH). In addition, the controller can provide contextual information to other network components, relating to the user’s failing authentication, or accessing sensitive services.

Later in a user session, the SDP controller can leverage authentication and authorization mechanisms to force step-up authentication for the user, disconnect a user based on parameters such as session timeout, geo-location changes, or security posture changes. The SDP controller closely aligns with the Zero Trust Policy Decision Point (PDP) concept.

The SDP Controller may reside in the cloud or on-premises, depending on the SDP solution.

SDP Controllers are protected by an isolation mechanism using the single-packet authorization (SPA) protocol, or another variant , making the Controllers invisible and inaccessible to unauthorized users and devices. This mechanism may be provided by an SDP Gateway in front of a Controller, or natively within the Controller itself.

SDP Initiating Hosts (and SDP Clients)

SDP initiating Hosts (IHs) communicate with the SDP Controller in order to begin the process of accessing protected corporate resources via Accepting Hosts.

The Controller typically requires that the IH provide its information such as user identity, hardware or software inventory, device health as part of the authentication phase and before providing any access. The SDP Controller must also provide some mechanism (e.g., credentials) for IH to establish secure communications with the AH.

The IH can be in the form of a client application installed on the end-user's machine or can be in the form of a web browser. Using a client application provides deeper capabilities such as host checking (device posture checks), traffic routing, and more streamlined authentication.

One of the most important benefits of SDP client software on the Initiating Host (IH) is that the SDP client application will initiate connections to the SDP using the single packet authorization (SPA) protocol (discussed in depth later in this document).

The IH can be a human user (e.g., employee or contractor), an application (e.g., thick client), or an IOT device (e.g., remote water meter). In this latter example, the identity is a nonhuman identity but is nonetheless authenticated and authorized in a similar fashion.

SDP Accepting Hosts (and Services)

Accepting Hosts (and services) are any corporate or backend resource to which a user might wish to connect and to which the responsible enterprise needs to control access. Services can be located on-premises, in a private cloud, public cloud, etc.

An AH (as a Gateway or service) is not limited to being only a web application; it can be any TCP or UDP application - SSH, RDP, SFTP, Web, SMB, proprietary applications, fat client application, etc.

By default, all network access to an AH is blocked and can only be accessed by authenticated users. Control plane traffic to the AHs should be allowed, for example, between SDP Controllers and Gateways (depending on the deployment architecture).

SDP Gateways

An SDP Gateway provides isolation and is the component (software appliance or agent) that is the AH, acting as the frontend for the protected services and enforcing the authentication and authorization rules maintained by the SDP Controller. (Note that depending on the SDP deployment model, discussed below, the AH may be a separate component or may be part of the service being accessed)

The SDP gateway receives the control information from the SDP controller, and it accepts connections from an IH only after instructed to do so by the SDP Controller.

In conjunction with the SDP controller, the SDP Gateway provides authorized IHs (users and devices) with access to AHs (protected processes and services).

The SDP Gateway essentially works as a reverse proxy or a network firewall as it receives traffic from the IH and passes it to the protected backend service. The response from the backend service is then sent back to the IH.

The SDP gateway resides on or near the AH and can also reside on or near the IH, the SDP Controller, or both depending on the isolation and enforcement requirements. The gateway can include a firewall or access control list. The gateway can also enact monitoring, logging, and reporting on these connections.

In some deployment models, the SDP Gateway can be integrated with the AH (or service) to accomplish isolation and access enforcement for ONLY authorized users to a specific service.

In addition to the isolation mechanism, the SDP Gateway can also enforce policies and align with the Zero Trust Policy Enforcement Point (PEP) concept.

SDP Deployment Models[7]

SDP connects clients (humans or non-person entities) to resources (depicted as servers in the diagrams below). These resources may be of any type, as long as they are addressable across a network. They may be services running on physical or virtual servers, may be services running on IaaS or PaaS platforms, or may be containerized services.

This section briefly introduces the six SDP deployment models. While they have different network topologies, logically, they provide the same benefits - enforcing access to protected resources. For an introduction to these deployment models, see the SDP Architecture Guide.

Note that in the diagrams below, the blue lines indicate network connections secured by mTLS. The gray lines indicate network connections utilizing the native application protocol, which may or may not be encrypted.

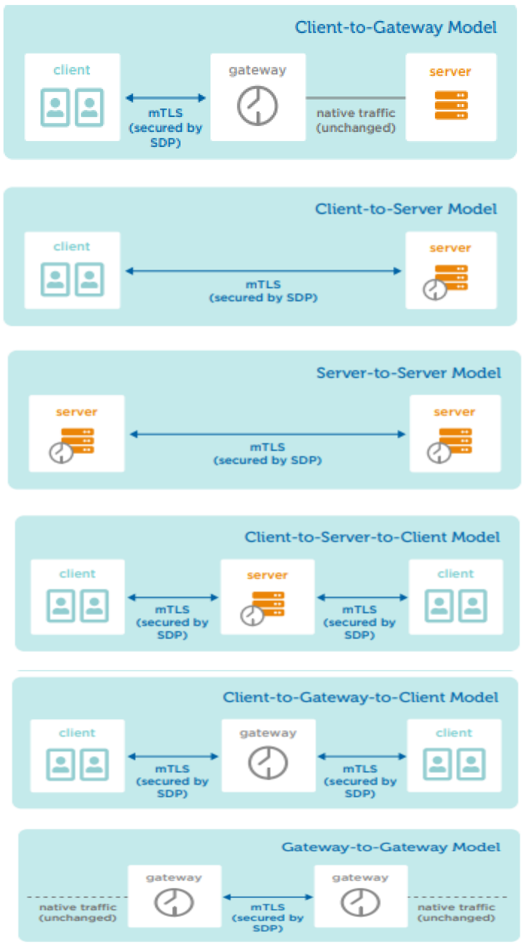


Figure 2 - SDP Deployment Models

When one or more servers must be protected behind a gateway, the connections between client/ IH and the AH (gateway) are secured regardless of underlying network topology. In this model, the Gateway is hidden and within the secure perimeter. This model doesn't require any changes to protected servers.

When an organization needs to secure connections end-to-end, this model combines a server and AH (gateway) in a single host. In this model, the server is hidden and within the secure perimeter. In this model, the server platform must be able to have the Gateway software deployed onto it.

This model is best suited for Internet of Things (IoT) and Virtual Machine (VM) environments and ensures that all connections between servers are encrypted regardless of the underlying network or IP infrastructure. In this model, all servers are hidden and within the secure perimeter.

In some instances, peer-to-peer traffic passes through an intermediary server, such as in IP phone, chat, and video conferencing services. In this model, the server is hidden and within the perimeter.

This model is a variation of Client-to-Server-to-Client, above. This model supports peer-to-peer network protocols requiring clients to connect directly to one another while enforcing SDP access policies. In this model, the gateway is hidden and within the perimeter.

The Gateway-to-Gateway model was not included in the initial publication of SDP Specification 1.0. This model is well-suited for certain Internet of Things (IoT) environments. In this model, the gateways are hidden and within the perimeter.

[7] CSA Software-Defined Perimeter ARCHITECTURE GUIDE, May 2019, pages 14 to 18

SDP Workflow

The SDP Protocol has the following general workflow (depicted in Figure 4) to secure connections between the IH and AH hosts. This workflow is representative, although specifics will vary between different implementations. Each of the steps is discussed in further detail below.

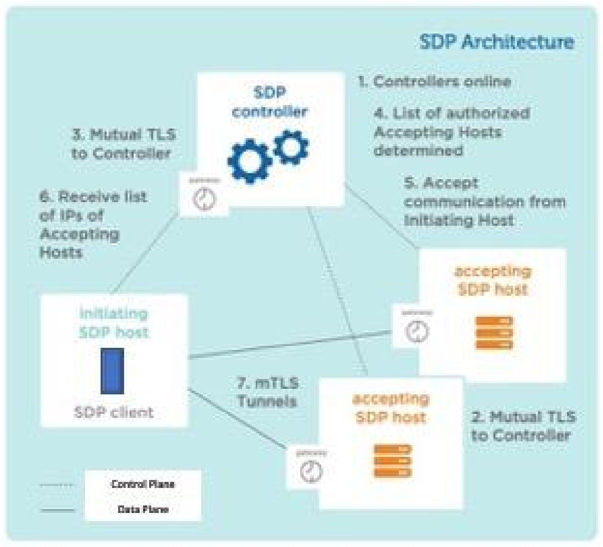

SDP Architecture
SDP controller
1. Controllers online
4. List of authorized Accepting Hosts determined
3. Mutual TLS to Controller
5. Accept communication from Initiating Host
6. Receive list of IPs of Accepting Hosts
accepting SDP host
initiating SDP host
SDP client
7. mTLS Tunnels
accepting SDP host
2. Mutual TLS to Controller
Control Plane
Data Plane

Figure 3 - SDP Workflow

Onboarding

Step	Onboarding Connection Flow
1	One or more SDP Controllers are brought online and connected to the appropriate optional authentication and authorization services (e.g., PKI Issuing Certificate Authority service, device attestation, geolocation, SAML, OpenID, OAuth, LDAP, Kerberos, multi-factor authentication, and other such services).
2	One or more AHs are brought online, depicted as SDP Gateways. These Gateways connect to and authenticate to the Controllers. However, they do not acknowledge communication from any other Host and will not respond to any non-provisioned request.
3	One or more clients on IHs are onboarded and each user (or NPE) is authenticated by the SDP Controller.

Note: The onboarding process is distinct from the user authentication process in that users are only onboarded once, but will authenticate and be authorized for each subsequent connection.

Subsequent Connection Flow

Step	Subsequent Connection Flow
1	When an onboarded IH returns online (e.g. after device reboot, or when user initiates a connection), it connects to and authenticates through the SDP Controller
2	After authenticating the IH (in some cases with its corresponding identity provider) the SDP Controllers determine a list of AHs with which the IH is authorized to communicate.

	The SDP Controller instructs the AHs to accept communication from the IH as well as any information that defines connectivity between users, devices, and services required for two-way encrypted communications.
	The SDP Controller gives the IH the list of authorized AHs as well as any optional information required for two-way encrypted communications.
3	The IH uses a single packet authorization (SPA) protocol to initiate a connection to each authorized AH that verifies the information in the SPA (for enforcement). The IH then creates a mutual TLS connection to those AHs.[8]

Onboarding

The on-boarding of SDP Controllers, one or more IHs, and AHs and users varies depending on the deployment models discussed in Figure 3. Methods could include Chef or Puppet or Terraform, or their hosting service equivalents (e.g., RightScale, AWS CloudFormation, etc.).

As described in the workflow above (Figure 3), the SDP is deployed and configured, and SDP Controllers and Accepting Hosts are brought online.

The following sequence is a sample[9] onboarding workflow for users on IHs that authenticate with an external Identity Provider (IdP).

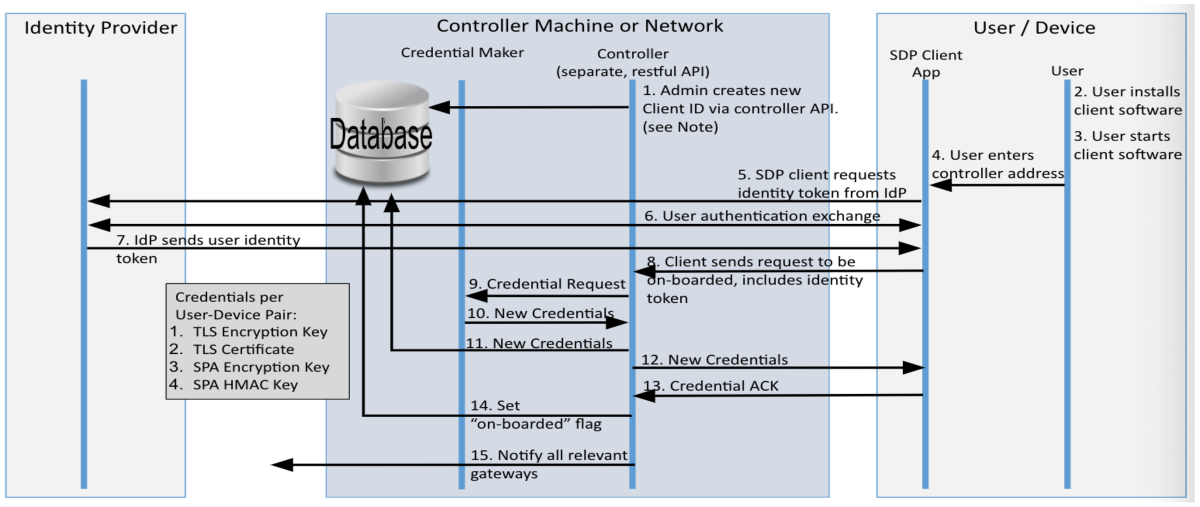

Identity Provider
Controller Machine or Network
User / Device
Credential Maker
Controller (separate, restful API)
SDP Client App
User
Database
1. Admin creates new Client ID via controller API. (see Note)
2. User installs client software
3. User starts client software
4. User enters controller address
5. SDP client requests identity token from IdP
6. User authentication exchange
7. IdP sends user identity token
8. Client sends request to be on-boarded, includes identity token
9. Credential Request
10. New Credentials
11. New Credentials
12. New Credentials
13. Credential ACK
14. Set "on-boarded" flag
15. Notify all relevant gateways
Credentials per User-Device Pair:
1. TLS Encryption Key
2. TLS Certificate
3. SPA Encryption Key
4. SPA HMAC Key

Figure 5 - Onboarding (IdP Scenario)

Note - This example uses Identify Provider information for the onboarding users. Onboarding systems could include multiple authentication factors, such as MFA and validation of device attributes (e.g., enterprise-deployed certificates or endpoint management software) as part of the onboarding process.

In this sample workflow, the IH uses a client (SDP Client in the figure) to negotiate with the Identity Provider (IdP). Note that for some SDP models, the SDP Controller will have a trust relationship with the IdP, for example, to validate a SAML token forwarded to it by the client.

[8] Steps 1 and 2 in this table may not be executed each time an IH needs to make a connection to an AH. Depending on the implementation, an IH that was previously authenticated by the Controller and still has valid access can just use SPA to connect to the AH. This model ensures that the IH can still connect in the event that the Controller is offline.
[9] Based upon SDP Open Source Reference Implementation - https://www.waverleylabs.com/open-source-sdp/

Single Packet Authorization (SPA)

One of the core principles of SDP is that resources must be inaccessible to unauthorized entities. This is enforced by requiring that entities must be cryptographically authorized to connect with any SDP component. This improves the security and resiliency of SDPs-- unauthorized entities are not able to establish a network connection with an SDP component, and therefore cannot attempt to exploit a vulnerability, brute force a login attempt, or utilize stolen credentials. This is in stark contrast to traditional remote-access solutions, such as VPNs, which are exposed to all malicious actors on the internet.

Validation of an incoming SPA packet is computationally lightweight, improving the resiliency of SDP systems against DDoS attacks (as mentioned in the SDP Cloud Security Alliance (CSA), SDP as a DDoS Defense Mechanism whitepaper).

The mechanism that SDP uses for this is Single Packet Authorization (SPA). SPA is based on an RFC 4226 HMAC-based One-Time Password "HOTP"[10], which is included in the SPA packet as described below.

Single Packet Authorization (SPA) initiates communication for all of the following: IH-Controller, AH-Controller, and IH-AH. The SPA packet initiates using either UDP or TCP protocols depending on the chosen implementation.

UDP-based SPA provides the following security benefits to the SPA-protected server:

- **Blackens the server**: The server will not respond to any attempted connections from any remote system until they have provided an authentic SPA that is valid for that SDP system. Specifically, the host will not respond to a TCP SYN, thereby avoiding the disclosure of any information to a potential attacker.

- **Mitigates Denial of Service attacks on TLS**: Internet-facing servers running the HTTPS protocol are highly susceptible to Denial-of-Service (DoS) attacks. SPA mitigates these attacks because it allows the server to reject unauthorized connection attempts before incurring the overhead of establishing a TCP or TLS connection and therefore allowing authorized connections during and in spite of DoS attacks.

- **Attack detection**: The first packet to an AH from any other host must be a SPA packet. If an AH receives any other packet, it should be viewed as an attack. Therefore, the SPA enables the SDP to determine an attack based on a single malicious packet.

TCP-based SPA obtains some of these benefits, to a lesser degree than UDP-based SPA. Specifically, an SDP component using TCP SPA will expose an open port to all remote (and potentially malicious) users, so the server will not be blackened. The server will also be partially subject to a DDoS attack -- it will likely permit the establishment of a TCP connection from any remote IP address, and then perform SPA validation prior to creating a TLS connection. A TCP connection is much less resource-intensive than a TLS connection, but it does consume server resources and puts the server at some level of risk of DDoS. Finally, using SPA over TCP will permit the server to detect an attack based on an invalid SPA packet, but only after the TCP

[10] See https://tools.ietf.org/html/rfc4226

connection is established, therefore consuming server resources.

Note that the recommended SPA message format, below, has been updated since v1 of the SDP specification, to improve the security and resiliency of the protocol.

SPA Message Format

While SPA message formats may differ between SDP implementations, all SDP systems must support SPA as the mechanism for initiating connections between components. Note that the use of SPA requires that SPA packet creators and recipients have a shared root of trust, as each SPA packet requires a shared secret in order to construct a valid SPA packet. The establishment of this root of trust -- namely, how the shared secret is securely communicated to SDP components - is implementation-dependent, and is outside the scope of this specification. Typically this information is included in the onboarding process for IHs and AHs.
DRAFT May 9 for validation

ClientID	32-bit numeric identifier, assigned per user-device pair. This field is optional, and used for SPA schemes that distinguish on a per-client basis
Nonce	16-bit random data field prevents replay attack by avoiding SPA packet reuse
Timestamp	Prevents servicing outdated SPA packets, by ensuring a short time period of validity (example 15 to 30 seconds).
Source IP Address	The *publicly visible* IP address of the initiating host. This is included so that the Accepting Host does not rely on the source IP address in the packet header, which is easily modified en route. The IH must be able to obtain the IP address for use by the AH as the origination of packets.
Message Type	This field is optional - it may be used to inform the recipient what type of message to expect from the IH after the connection is established.
Message String	This field is optional and will be dependent on the Message Type field. For example, this field could be used to specify the services that an IH will be requesting if known at connection time.
HOTP	This hashed one-time-password is generated by an algorithm such as RFC 4226, based on a shared secret. The use of an OTP is required in SPA packets for authenticity; other OTP algorithms can be substituted with the overarching goal of providing authenticity of the SPA packet.
Counter	The counter is a 64-bit unsigned integer intended to be synchronized between communicating pairs. In RFC 4226, this is done via a "look-ahead window" (because the typical use case for RFC 4226 is a hardware OTP token). However, for the SDP protocol, the counter can be sent in the SDP packet obviating the need for a look-ahead window and the potential for the communicating pair to be out of sync. Note that the counter does not need to be kept secret, however AHs should have mechanisms in place to avoid malicious use of very large counters, potentially denying service to (legitimate) IHs sending lower counter values. This field is optional, depending on the OTP algorithm chosen.
HMAC	Calculated over all fields above. Algorithm choices are SHA256 (recommended), SHA384, SHA512, and SM3. The HMAC is calculated using a shared (secret) seed. The HMAC is calculated over all prior fields of the message and then used by the AH to verify message integrity. The HMAC validation is computationally lightweight, and therefore can be used to provide resiliency against DoS attacks. Any SPA packets with invalid HMACs will be immediately discarded.

Figure 6: SPA Message Scheme

Note that other SPA options may include additional encryption, for example using the IH's private key (for non-repudiation), or the AH's public key (for confidentiality). However, asymmetric encryption is computationally expensive, and should only be used after a lighter-weight validation mechanism (such as a simple HMAC), in order to keep the AH resilient to DoS attacks.

SPA as a Secure, Self-Contained, Connectionless Message Transmission Protocol

One interesting "side" use case for SPA within an SDP system is the ability to use SPA packets as the means for actually transmitting data from a remote element. Because the SPA packet is based on a shared secret, the recipient can trust that the data contained within it has been issued by a valid SDP client.

If the SPA seed is unique to a given client (identified by the ClientID in the SPA packet), then the SPA *Message String* field can be used to transmit meaningful data by the client. This doesn't require any further processing or policy evaluation, and doesn't require the establishment of a TCP or TLS connection.

This could be useful for a set of distributed IoT sensors, for example, which need to transmit small amounts of data regularly . Embedding the data within a SPA packet permits these devices to accomplish this without incurring the overhead of establishing a TCP and TLS connection. Of course, this mechanism has some downsides. The recipient (accepting host) must be expecting this data, and because this is a unidirectional transmission, the sender receives no verification that the data was actually received with the 'fire and forget' SPA packet transmission. Nonetheless, this could be a useful way to apply SPA for some environments as long as the data is not significant and/or critical.

SDP in relation to SDN & NFV

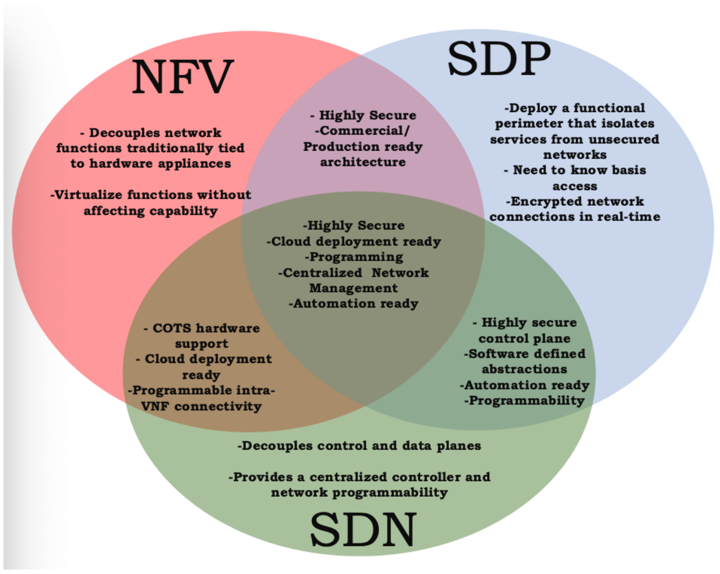
In cloud computing environments, Software Defined Network (SDN) and Network Function Virtualization (NFV) technologies efficiently address challenges[11] both at the IH environment (frontend) and AH environment (backend) while providing the benefits of scaling on-demand, pay-as-you-go, and providing resources as services. SDNs and NFVs pave the way for adopting and orchestrating heterogeneous network routing for better utilization of resources (wireless and computing resources). The IH environment challenges are related to the communication aspects of the access layer of the wireless mobile network or edge network, whereas the AH environment challenges are related to the network functions such as routing, switching, security, accounting and billing, and other such operations required for the functioning of network routing.

One of the main challenges in the NFV is resource exhaustion. The software that uses a particular physical server's resources intensively may exhaust those resources and hence affect VM availability. This condition occurs because the shared environment in a physical server magnifies the severity of resource contention, especially when multiple VMs are running the same resource-intensive software at the same time. This problem can be addressed by using SDP in which the SDP controller defines and implements a standard operating procedure that detects VMs that are throttled due to resource exhaustion—similar to Denial of

[11] J. Singh, A. Refaey and J. Koilpillai, "Adoption of the Software-Defined Perimeter (SDP) Architecture for Infrastructure as a Service," in Canadian Journal of Electrical and Computer Engineering, vol. 43, no. 4, pp. 357-363, Fall 2020, doi: 10.1109/CJECE.2020.3005316.

service—and puts a remedy in place dynamically. Another common risk in the NVF is account or service hijacking through the self-service portal or cloud management console, in which access to the portal or console increases exposure to risks such as account or service hijacking through more administrative privileges than are typically granted to end-users . In this case, the SDP shows a vital role in eliminating this risk and using administrative controls selectively, based on users' roles and functions.

There is widespread adoption of Software-Defined Networks (SDN) by Cloud Service Providers to simplify network management. The main challenges of SDNs are how to provide proper authentication, access control, data privacy, and data integrity among others for the API-driven orchestration of network routing. Herein, the Software Defined Perimeter (SDP) can provide orchestration of connections that restricts network access and connections between objects on the SDN-enabled network infrastructures.
There are several potential benefits as a result of the integration between SDPs and SDNs. In particular, it provides a completely scalable and managed security solution.


NFV
- Decouples network functions traditionally tied to hardware appliances
-Virtualize functions without affecting capability
SDP
-Deploy a functional perimeter that isolates services from unsecured networks
- Need to know basis access
-Encrypted network connections in real-time
- Highly Secure
-Commercial/ Production ready architecture
-Highly Secure
-Cloud deployment ready
-Programming
-Centralized Network Management
-Automation ready
- COTS hardware support
- Cloud deployment ready
-Programmable intra-VNF connectivity
- Highly secure control plane
-Software defined abstractions
-Automation ready
-Programmability
-Decouples control and data planes
-Provides a centralized controller and network programmability
SDN

In short, consider the Software-Defined Network (SDN) and the Network Function Virtualization (NFV) as two sides in a network virtualization triangle, the Software-Defined Perimeter (SDP) completes the missing piece of this triangle. Even though, both SDN and SDP operate in the networking arena and have similar names, the SDN can be considered as the brain which orchestrates network operations, while the SPD introduces reliable network connectivity with zero trust concepts without significant obstruction.

Mutual Transport Layer Authentication Between Components

SDPs require multiple layers of validation throughout the connection establishment process. The first step is SPA, as described above. The next step, which is the subject of this section, is the requirement for mutual authentication as part of the establishment of a secure, encrypted connection between distributed SDP components. (Additional steps, including device and user validation, are discussed below).

After components have utilized SPA to be validated and authorized, connections between the primary components in an SDP system should use TLS or Internet Key Exchange (IKE), with mutual authentication, to validate the device as an authorized member of SDPs. Specifically, the connections between the Initiating Host and Accepting Hosts (IH-AH) and the Initiating Host to Controller (IH-Controller) **must** use mTLS, while the connection and the Controller-Gateway connection **should** use mTLS.

All weak cipher suites and all suites that do not support mutual authentication are ill-advised and not secure. This ensures that the components each contain a valid private key, issued by a trusted authority, significantly reducing the likelihood of a Man-in-the-Middle attack[12].

The root certificate for these components must be an enterprise PKI system or an SDP-specific CA. It must not rely on pre-issued or implicitly trusted certificates associated with consumer browsers, which have been subject to impersonation attacks whereby an attacker has forged a certificate from a compromised certificate authority. SDPs should implement a scheme to ensure that revoked certificates can be detected efficiently, such as using OCSP[13] or another mechanism.

Any SDP must be resilient to potential attacks - such as a MITM TLS downgrade protocol attack (which mTLS avoids), as well as the potential for vulnerabilities associated with long-lived access tokens from authentication systems such as SAML or OpenID Connect.

Device Validation

Mutual transport layer authentication proves that the device requesting access to the SDP possesses a private key that has not expired and that has not been revoked, but it does not prove that the key has not been stolen. The objective of Device Validation is to prove that the proper device holds the private key and that the software running on the device can be trusted. In the simplest form of SDP described in this document, the Controller is assumed to be a trusted device (because it exists in the most controlled environment) and the IHs and AHs must validate it. SDP Gateways that aren't also acting as accepting hosts are also assumed to be trusted components as they are under the control of the enterprise operating the SDPs.

User devices, on the other hand, require validation. Device Validation mitigates credential theft and the resultant impersonation attacks. The user's device which has been authenticated with the SPA message is allowed to connect to the SDP controller via the SDP gateway. This process ensures that an attacker will not be able to access services on or behind the AH, even if the user is in possession of the correct private keys for the connection. All packets are dropped from the user's device unless authenticated and authorized via SDP, thus preventing any incoming packets from all unauthorized user devices. Device validation protocols are enterprise and product-specific, and as such are beyond the scope of the SDP specification

SDP systems must support the ability to integrate with enterprise device management / endpoint management systems, and include their device posture checks into a device validation process. In addition, SDP systems should support the ability to perform local device posture checks, in environments where the SDP system has software running locally on the user device. For example, an SDP client could validate that a user's device contains a valid certificate issued by the enterprise. Or, it could validate that the device is running an approved Anti-Virus component. Note that these aspects are related to the overall Zero Trust approach that SDP supports -- using device information as additional context for making access policy decisions. Also note that "devices" here can also refer to servers, which can and should be validated, in many of the same ways as user devices.

[12] https://wott.io/blog/tutorials/2019/09/09/what-is-mtls
[13] https://tools.ietf.org/html/rfc6960

SDP Protocol

The SDP protocol defined here comprises four sections: AH-Controller Protocol, IH-Controller Protocol, IH-AH Protocol, and Logging, which are explained in detail below. The SDP protocol shown here illustrates the types of communications between SDP components, but it is not normative - the different SDP deployment models will necessitate different interactions and messages. The goal here is to show a representative system that works but not to prescribe a fixed or standardized message format .

AH-Controller Protocol

Whatever network protocol is used for the communication between AH and Controller, and the type of network connection, the following elements are essential:

1. Initiating request message - SPA
2. Secure message channel - secure TCP, VPN, etc. channel
3. Timeout function - for credential validity
4. Agreed format for authentication - MFA, etc.
5. Specific resource list to which the request is permissioned - list of protected AHs
6. Allow or deny response that satisfies the Zero Trust architecture of SDP - reject invalid SPA
7. Enables messages to be encrypted - secure TCP, VPN handshake
8. Enables validation of messages by a current HMAC function (Hashed Message Authentication Code) or similar construct - SPA

AH to Controller Sequence Diagram

The sequence diagram for the protocol for the AH to connect to the Controller is as follows. A UDP-based SPA packet is the first packet in order to ensure that the SDP Controller is protected from unauthorized access.

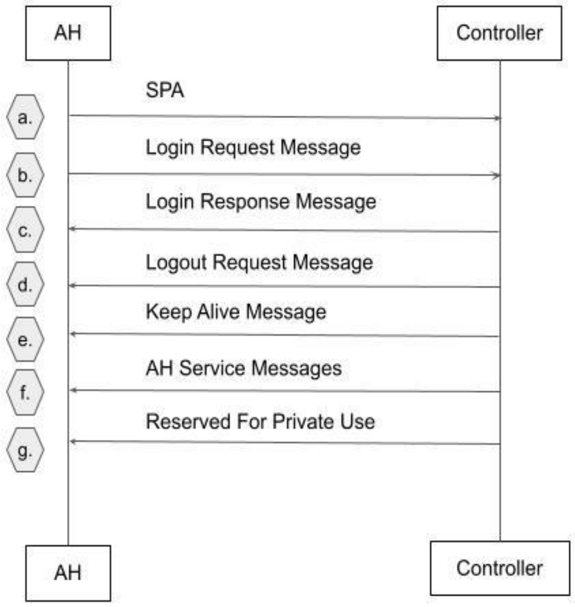


Figure 7: Accepting Host connects to the Controller

The following subsections define the various messages and formats passed between the AH and Controller. The basic protocol is of the form:

a. SPA

The request message is sent by the AH to the Controller to request a connection.

0x00	No command-specific data

The JSON specification for this as an example is:

Format	Example

{"credentials":
  ["spa_encryption_key":<64 bit>,
   "spa_hmac_key": <64 bit>,
   "tls_key": <file contents>,
   "tls_cert" <file contents>
  ]
}
	{"credentials":
	    ["spa_encryption_key":"aldskf…",
	     "spa_hmac_key": "asldjf…",
	     "tls_key": "tls_key",
	     "tls_cert": "tls_cert"
	    ]
	  }

b. Login Request Message

The request message is sent by the AH to the Controller to indicate that it is available and is able to accept other messages from the Controller.

0x00	No command-specific data

c. Login Response Message

The login response message is sent by the Controller to indicate whether the login request was successful and, if successful, to provide the AH Session ID.

0x01	Status Code (16 bits)	AH Session ID (256 bits)

d. Logout Request Message

The logout request message is sent by the AH to the Controller to indicate that it is no longer available and is not able to accept other messages from the Controller. There is no response.

`0x02`	`No command-specific data`

e. Keep-Alive Message

The Keep-Alive message is sent by either the AH or the Controller to indicate that it is still active.

`0x03`	`No command-specific data`

f. AH Service Messages

The services message is sent by the Controller to indicate to the AH that set of services that this AH is protecting.

`0x04`	`JSON formatted array of Services`

The JSON specification for the data plane is:

ormat	Example

`{“services”:`
` [“port”: <Server port>,`
`  “id”: <32-bit Service ID>,`
`  “address”: <Server IP>,`
`  “name”: <service name>`
`  “session” : <session tag >`
` ]`
`}`
	`{“services”:`
	` [“port”: “32843”,`
	`  “id”: “123445678”,`
	`  “address”: “100.100.100.100”,`
	`  “name”: “SharePoint”,`
	`  “session”: “HTTPS”`
	` ]`
	`}`

Note that this could be used to describe TCP or UDP-based services, or even services using other protocols (e.g. ICMP).

g. Reserved for Private Use

This command (0xff) is reserved for any non-standard messages between the AH and the Controller.

`0xff`	`User-specified`

IH-Controller Protocol

The Initiating Host Controller Protocol operates on the network routing and packet delivery, and implementation details depend on the type of transport (e.g., TCP guaranteed delivery or UDP fire and forget). Whatever network protocol is used for the communication between IH and Controller, and the type of network connection, the following elements are essential:

1. Initiating request message - SPA
2. Secure message channel - secure TCP, VPN, etc. channel
3. Specific authorized resource list - list of protected AHs
4. Enables messages to be encrypted - secure TCP, VPN handshake
5. Enables validation of messages by a current HMAC function (Hashed Message Authentication Code) or similar construct - SPA

IH to Controller Sequence Diagram

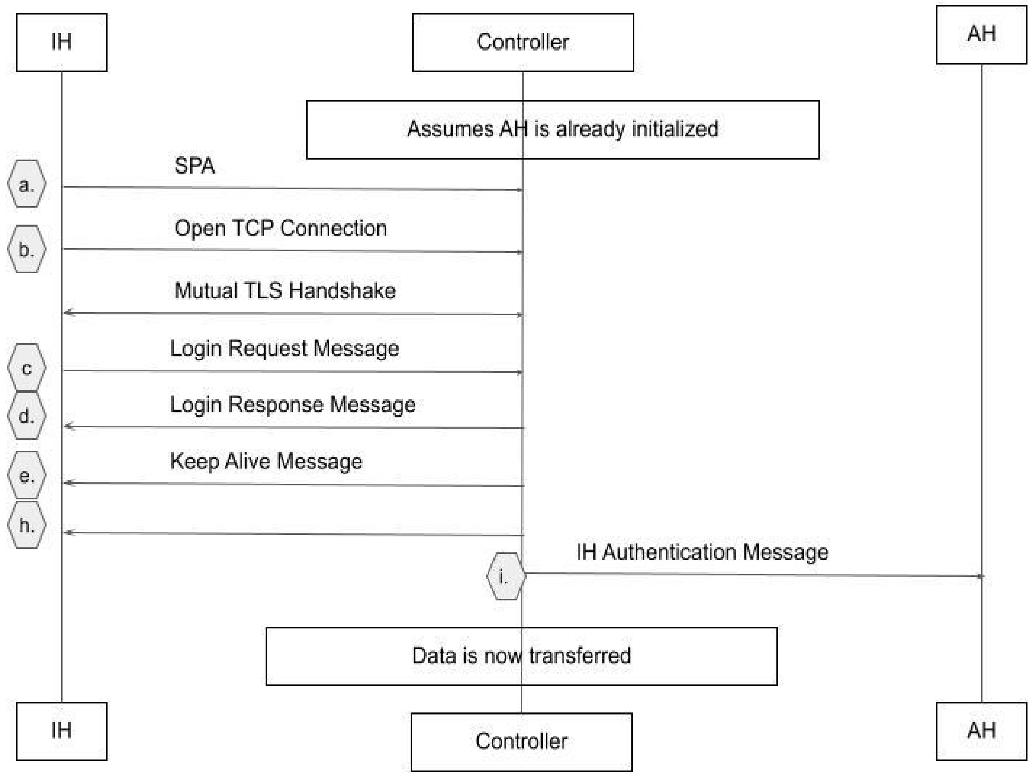


Figure 8: Initiating Host connects to the SDP Controller

The following subsections define the various messages and their formats that are passed between the IH and the Controller. The basic protocol is of the form:

`Command (8-bit)`	`Command specific data of command-specific length`

a. SPA

The request message is sent by the AH to the Controller to request a connection.

0x00	No command-specific data

b. Open TCP Connection

The mutual TLS initiation sequence.

c. Login Request Message

The login request message is sent by the IH to the Controller to indicate that it is available and would like to be part of the SDP.

0x00	No command-specific data

d. Login Response Message

The login response message is sent by the Controller to indicate whether the login request was successful and, if successful, to provide the IH Session ID.

0x01	Status Code (16 bits)	IH Session ID (256 bits)

e. Keep-Alive Message

The Keep-Alive message is sent by either the IH or the Controller to indicate that it is still active.

0x03	No command-specific data

h. IH Service Message

The services message is sent by the Controller to indicate to the IH the list of available services and the IP addresses or hostnames of the AHs protecting them.

0x06	JSON formatted array of Services

The JSON specification is:

Format	Example
{"services": [{"address" : <AH IP>, "id": <32-bit Service ID>, "name": <service name>, “type” : <service type> }] }	{"services": [{"address" : “200.200.200.200”, "id": “123445678”, "name": “SharePoint”, “type” : “HTTPS” }] }

i. IH Authentication Message

The role of the Accepting Host is to ensure that an authentication request is validated prior to allowing access to the list of protected resources. The IH Authenticated Message is sent by the Controller to the AH to indicate to the AH that a new IH has been validated and that the AH should allow access to this IH for the specified services.

0x05	JSON formatted array of IH information

The JSON specification is:

Format	Example
{“IH Authenticators”: “IH”: <IH/Device Pair>, “sid”: <32-bit IH Session ID>, “seed”: <32-bit SPA seed>, “counter”: <32-bit SPA Counter> [“id”: <32-bit service ID>,] }	{“IH Authenticators”: “IH”: “IH/DeviceID”, “sid”: “4562”, “seed”: “###”, “counter”: ‘####’, [“id”: “123445678”] }

j. Reserved for Private Use

This command (0xff) is reserved for any non-standard messages between the IH and the Controller.

0xff	User-specified

IH-AH Protocol

The Initiating Host to Accepting Host Protocol operates on the network routing and packet delivery. Implementation details depend on the type of transport (e.g., TCP guaranteed delivery or UDP fire and forget). Whatever network protocol is used for the communication between IH and AH, and the type of network connection, the following elements are essential:

1. Initiating request message - SPA
2. Secure message channel - secure TCP, VPN, etc. channel created dynamically to the service
3. Timeout - duration the channel persists or keep-alive

It is important to note that the connection is established dynamically to the service only after the IH is validated. Until then the service is kept hidden by the AH.

IH to AH Sequence Diagram

A sample sequence diagram for the protocol between the IH and the AH is shown below. This sequence diagram only describes the message sequence associated with the initial login. The messages sent when an IH connects to a Controller are shown in the AH-Controller Sequence Diagram.

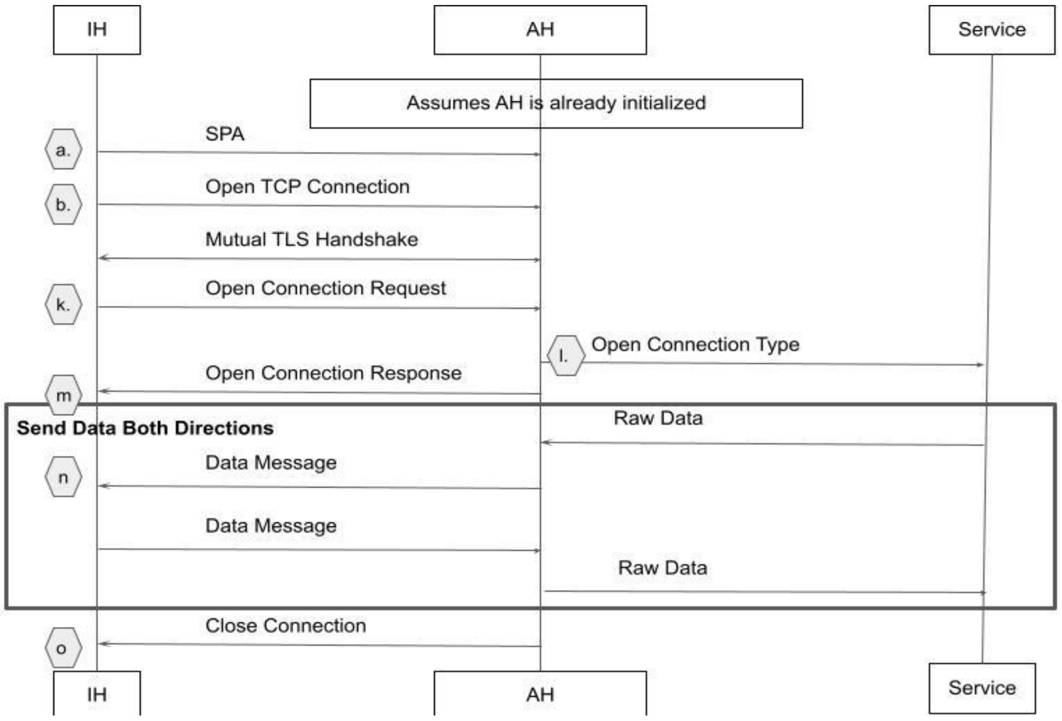


Figure 9: IH connects to an AH and then sends data to a Service

The following subsections define the various messages and formats passed between the IH and the AH. The basic protocol is of the form:

Command (8-bit)	Command specific data of command-specific length

a. SPA

The request message is sent by the AH to the Controller to request a connection.

0x00	No command-specific data

b. Open TCP Connection

The mutual TLS initiation sequence.

k. Open Connection Request Message

The IH sends the open request message to the AH to indicate that it would like to open a connection to a particular Service.

0x07	Mux ID (64 bits)

l. Open Connection Type

The AH sends the open connection type message to the Service to initiate a data connection.

0x07	Mux ID (64 bits)

m. Open Connection Response Message

The open-response message is sent by the AH to the IH to indicate whether the open request was successful.

0x08	Status Code (16 bits)	Mux ID (64 bits)

n. Data Message

The data message is sent by either the IH or the AH. It is used to push data on an open connection. There is no response.

0x09	Data Length (16 bits)	Mux ID (64 bits)

o. Connection Closed Message

The connection closed message is sent by either the IH or the AH. It is used to either indicate that a connection has been closed by the AH or that the IH is requesting a connection be closed. There is no response.

`0x0A`	`Mux ID (64 bits)`

p. User-Defined Message

This command (0xff) is reserved for any non-standard messages between the IH and the AH.

`0xff`	`User-specified`

Logging

Creating logs to determine the availability and performance of services and the security of the server is a requirement of all systems and for a Zero Trust implementation.

Fields of a log message

All logs will have the following fields.

Field Name	**Meaning**
`time`	time at which log record was generated
`name`	human-readable name for the event. Note: do not include any mutable data nuggets, such as usernames, IP addresses, hostnames, etc. That information is contained in the additional fields of the log record already and we don't want to repeat them.
`severity`	a severity for this event ranging from debug to critical (see below)
`deviceAddress`	the IP address of the machine generating the log record

Operations Logs

The following is a list of operational use cases or activities that need to be logged.

The signature_id is an identifier, making it possible to identify the type of event. The third column contains the additional fields that need to be logged for the specific log messages.

activity	signature_id	data/information to log
component startup, shutdown, restart (e.g., Controller coming up, Host restart)	*ops:startup* *ops:shutdown* *ops:restart*	***reason*** indicating why a restart or shutdown has occurred ***component*** indicating what component was affected
connection between components (Controller, IH, AH, 3rd party component, DB) up, down, reconnect	*ops:conn:up* *ops:conn:down* *ops:conn:reconnect*	***src*** origin of the connection as an ip address as seen by the reporting entity ***dst*** destination of the connection as an ip address as seen by the reporting entity ***reconnect_count*** how many times reconnect was attempted ***reason*** indicating why the communication went **down**

Here is a quick scenario that outlines a complete outage, showing what log entries are written where. In this scenario, we assume that a Controller goes down:

The controller goes down [no log, a failing component is not able to log]

IH tries to reconnect to Controller n times
ops:conn:reconnect log messages

After n times, the client declares the connection to the Controller to be down and it looks for a new Controller
ops:conn:down log message, with a severity of the *error*

IH connects to the newly found Controller
ops:conn:up log message

If no more Controllers are available
ops:conn:down log message, with a severity of *critical*

Another similar case would be a client that goes down without warning (e.g., laptop being closed). In that case, the Controller would detect a failing connection, as well as the AH. Each of them would log a

ops:conn:down log message, with a severity of the *error*

Here is an example of how a complete user login (IH connecting to AH) looks:

IH connects to Controller
ops:conn:up log message

IH mutually authenticates to Controller

sec:auth log message

IH connects to AH
ops:conn:up log message

IH mutually authenticates to AH
sec:auth log message

Security/Connection Logs

The security logs are core to the SDP and are also going to be important in a broader context to detect larger-scale infrastructure attacks. Therefore, these logs are very useful if forwarded to a SIEM product and required for a Zero Trust implementation.

The signature_id is an identifier, making it possible to identify the type of event. The third column contains the additional fields that need to be logged for the specific signature_id.

activity	**signature_id**	**data/information to log**
AH login success	*sec:login*	**src** the IP address of the AH as seen by the Controller **AH Session ID** the session ID of the AH
AH login failure	*sec:login_failure*	**src** the IP address of the AH as seen by the Controller **AH Session ID** the session ID of the AH
IH login success	*sec:login*	**src** the IP address of the IH as seen by the Controller **IH Session ID** the session ID of the IH
IH login failure	*sec:login_failure*	**src** the IP address of the IH as seen by the Controller **IH Session ID** the session ID of the IH
component authentication (e.g., IH -> Controller)	*sec:connection*	**IH Session ID** the session ID of the IH **AH Session ID** the session ID of the AH
denied inbound connection	*sec:fw:denied*	***src*** source of the attempted connection ***dst*** destination of the attempted connection

Summary

Version 2.0 of the Software-Defined Perimeter (SDP) Specification has been a long time in coming. Version 1.0 was published seven years ago in April 2014.

The version 2.0 specification expands and modifies (clarifications and extensions) the following areas:

- SDP Relationship to Cloud Layers
- Onboarding workflows
- Non-Person Entities
- Secure Message Channel

Additionally, we have provided enhanced sequence diagrams and explanation of connections and messages in the following 3 SDP sub-protocols:

- AH to Controller
- IH to Controller
- IH to AH

Some of the topics that the SDP WG is considering for future research and publication are as follows:
- SDP's role in granting ZT Network access in a Secure Access Service Edge (SASE) security model
- SDP in the Cloud for consumers and SDP for Zero Trust
- Device Validation
- Describe transport layer protocols supporting role in Zero Trust
- Access via Policy decision point (PDP) and corresponding policy enforcement point (PEP)

SDP is an effective Zero Trust implementation.[14] This specification will encourage a Zero Trust paradigm shift for securing applications particularly in the cloud where Service Level Agreements (SLAs) favor the service providers, by giving consumers of cloud services more visibility into their security while developing cloud and/or hybrid applications.

SDP is also proven to secure internal Enterprises[1], Infrastructure as a Service offerings[2], Network Function Virtualization[3], Software-Defined Networking[4], and IoT applications[5].

[1] P. Kumar, Abdallah Moubayed, Ahmed Refaey, Abdallah Shami, and Juanita Koilpillai "Performance Analysis of SDP for Secure Internal Enterprises," IEEE Wireless Communications and Networking Conference (WCNC), 1-6, 2019.
[2] J. Singh, A. Refaey and J. Koilpillai, "Adoption of the Software-Defined Perimeter (SDP) Architecture for Infrastructure as a Service," in Canadian Journal of Electrical and Computer Engineering, vol. 43, no. 4, pp. 357-363, Fall 2020, DOI: 10.1109/CJECE.2020.3005316.
[3] J. Singh, A. Refaey and A. Shami, "Multilevel Security Framework for NFV Based on Software Defined Perimeter," in IEEE Network, vol. 34, no. 5, pp. 114-119, September/October 2020, doi: 10.1109/MNET.011.1900563.
[4] Ahmed Sallam, Ahmed Refaey, and Abdallah Shami, " On the Security of SDN: A Completed Secure and Scalable Framework Using the Software-Defined Perimeter," IEEE Access, Accepted, August 2019. (Impact factor: 4.098).
[5] Refaey, A.; Sallam, A.; Shami, A.: 'On IoT applications: a proposed SDP framework for MQTT', Electronics Letters, 2019, 55, (22), p. 1201-1203, DOI: 10.1049/el.2019.2334IET Digital Library, https://digital-library.theiet.org/content/journals/10.1049/el.2019.2334

[14] https://www.helpnetsecurity.com/2020/05/29/sdp-zero-trust/

References

Cloud Security Alliance (CSA), SDP Architecture Guide version 2.0, published May 2019, available @ https://cloudsecurityalliance.org/artifacts/sdp-architecture-guide-v2/

Cloud Security Alliance (CSA), SDP Glossary, published June 2018, available @ https://downloads.cloudsecurityalliance.org/assets/research/sdp/SDP-glossary.pdf

Cloud Security Alliance (CSA), SDP as a DDoS Defense Mechanism, published October 2019, available @ https://cloudsecurityalliance.org/artifacts/software-defined-perimeter-as-a-ddos-prevention-mechanism/

Waverley Labs, SDP Center, Open Source Reference Implementation (funded by DHS), available @ http://sdpcenter.com/test-sdp/

Cloud Security Alliance (CSA) Software-Defined Perimeter (SDP) Specification 1.0, published April 2014, available@ https://cloudsecurityalliance.org/artifacts/sdp-specification-v1-0/

Zero Trust Security: An Enterprise Guide, by Jason Garbis and Jerry W. Chapman, Apress, 2021, available @ https://www.apress.com/us/book/9781484267011

Zero Trust Networks: Building Secure Systems in Untrusted Networks, by Evan Gilman and Doug Barth, O'Reilly, 2017, available @ https://www.oreilly.com/library/view/zero-trust-networks/9781491962183/