Software-Defined Perimeter (SDP) Specification v2.0

cloud security alliance®

Acknowledgments

Lead Authors

Jason Garbis
Juanita Koilpillai

Contributors

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

Special Thanks

Larry Hughes

Reviewers

Alistair Cockeram
Takahiro Ono
T Prasad
Nya Murray
Michael Rash

CSA Analyst

Shamun Mahmud

CSA Global Staff

Claire Lehnert

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 Analyst

The Software-Defined Perimeter (SDP) and Zero Trust Working Group is a Cloud Security Alliance (CSA) research working group that advocates for and promotes the adoption of Zero Trust security principles. It provides practical and sound guidance on how to apply Zero Trust to cloud and non-cloud environments. This group will build on and leverage the NIST Zero Trust research and methodologies. It will recommend SDP as a fundamental architecture for applying Zero Trust principles. It will revise and expand the SDP specification, to capture and codify a wealth of knowledge based on experience. Lastly, the SDP group recognizes the need for an inclusive approach to SDP. As such, it also supports alternative architectures so long as they align with Zero Trust principles.

Dedication

This paper is dedicated to the memory of Juanita Koilpillai, a security leader, influencer, mentor, and friend. Juanita was a luminary in her field and broke many glass ceilings on her path to becoming founder and chief executive at two successful start-ups. In return, Juanita shared her knowledge by mentoring young women with careers in technology. She played a prominent mentorship role in IEEE's Women In Engineering (IEEE WIE) Organization. She also created a philanthropic fund in memory of her father and founded MERGE, a non-profit to support local youth projects. She was instrumental in the authorship of this as well as several other CSA, IEEE and NIST publications.

Table of Contents

Acknowledgments 3
Dedication 4
Introduction 7
  Purpose 7
  Scope 7
  Audience 7
SDP Design 8
  SDP Concept 9
  SDP Architecture and Components 10
    SDP Controller 11
    SDP Initiating Hosts (and SDP Clients) 12
    SDP Accepting Hosts (AH) 12
  SDP Deployment Models 13
  SDP Workflows 15
    Controller Onboarding Workflows 15
    Accepting Host (AH) Onboarding Workflow 16
    Initiating Host (IH) Onboarding Workflow 16
    Access Workflow 17
  IH Onboarding Workflow Example 18
  Single Packet Authorization (SPA) 19
    SPA Message Format 20
    SPA as a Secure, Self-Contained, Connectionless Message Transmission Protocol 21
    Alternatives to SPA 22
  Mutual Transport Layer Authentication Between Components 22
  Device Validation 23
  SDP and IoT Devices 24
  Access Policies 25
SDP Protocol 26
  AH-Controller Protocol 27
    AH to Controller Sequence Diagram 27
    a. SPA 27
    b. Open connection and establish mutually authenticated communications 27
    c. Login (SDP Join) Request Message 28

d. Login (SDP Join) Response Message 28
e. Logout Request Message 29
f. Keep-Alive Message 29
g. AH Service Messages 29
h. Reserved for Private Use 30
IH-Controller Protocol 30
IH to Controller Sequence Diagram 30
a. SPA 31
b. Open connection and establish mutually authenticated communications 31
c. Login (SDP Join) Request Message 31
d. Login Response Message 31
e. Keep-Alive Message 32
f. IH Services Message 32
g. IH Authenticated Message 33
h. Logout Request Message 34
z. Reserved for Private Use 34
IH-AH Protocol 34
IH to AH Sequence Diagram 34
a. SPA 35
b. Open connection and establish mutually authenticated communications 35
c. Open Connection Request Message 36
d. Open Appropriate Connection Type 36
e. Open Connection Response Message 36
f. Data Message 36
g. Connection Closed Message 37
z. User-Defined Message 37
Logging 37
Fields of a log message 37
Operations Logs 37
Security/Connection Logs 38
Summary 40
References 41
Appendix A: SDP, SDN & NFV 43
Appendix B: OSI / SDP Component Mapping 44

Introduction

The Software-Defined Perimeter (SDP) architecture provides the ability to dynamically and flexibly deploy network security perimeter functionality to isolate applications and services on unsecured networks. SDP is designed to provide an isolated, on-demand, and dynamically provisioned trusted overlay that mitigates network-based attacks from both inside and outside the enterprise-controlled network. An SDP implementation hides assets from unauthorized entities, establishes trust before allowing connections, and manages the system via separate control and data planes. With SDP, enterprises can achieve the goals of Zero Trust, thereby improving their security effectiveness and resiliency, by moving away from traditional (and largely ineffective) perimeter-centric models.

Purpose

This specification updates the Cloud Security Alliance (CSA) Software-Defined Perimeter (SDP) Version 1.0 specification published by the Software Defined Perimeter Working Group (SDP WG) in April 2014.

Although that original specification was sound, it did not sufficiently address several topics, including component onboarding, and securing non-person entities[1]. In addition, since the release of SDP v1, the industry has enthusiastically embraced its principles, and included them in what we now call *Zero Trust*. This revised version of the SDP specification is expanded and enhanced to include additions, clarifications, and extensions, as well as reflecting the current state of the industry around Zero Trust.

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

Scope

This specification encompasses the architectural components, interactions, and basic security communications protocol for SDP. It focuses on the control plane that enables secure connectivity within the security perimeter, and the data plane that enforces secure connectivity between *initiating hosts* and *accepting hosts*, be they servers, devices, or services.

Audience

The target audiences for this specification are

- solution architects and security leaders that are deploying Zero Trust and SDP products in their enterprises
- vendors and technology providers that are implementing Zero Trust with SDP architectures in their products

1 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.

SDP Design

SDP aims to give enterprise security architects, network providers and application owners the ability to

- deploy dynamic "software-defined" perimeters
- hide networks and resources
- prevent unauthorized access to the services running on them
- enforce an identity-centric access policy model

SDPs replace physical appliances with logical components that operate under the organization's control, regardless of where they are physically deployed, thereby minimizing the logical perimeter. SDPs enforce the Zero Trust principles of enforcing least privilege access, assuming breach, and "trust but verify", authorizing policy-based access to resources only after attestation and identity verification.

SDP's design objective is to provide an effective and easily integrated security architecture for IPv4 and IPv6 networks. It includes obfuscation of and access control to control plane elements. It provides for the confidentiality and integrity of communications across the data plane. It also includes a need-to-know access model that requires authenticated identities (users and non-person entities) on validated devices to cryptographically sign in to the perimeter.

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 any IP infrastructure, whether traditional hardware-based, software-defined network (SDN), or cloud-based. SDP's mutually authenticated tunnel is effectively an encrypted overlay, which can be deployed atop any type of IP network. As a result, SDP normalizes security layers across heterogeneous environments, simplifying networks, security and operations.

For cloud infrastructures, SDP integrates security at five of the seven OSI layers, namely:

- The **network layer**, where virtualization[2] 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 **presentation layer**, where middleware manages application tiers and the application.
- The **application layer**, that provides business value to users.

The OSI layers where SDP is not integrated are the data link and physical layers. The TCP/IP model refers to these combined layers as the Network Layer.

See Appendix B for additional information on SDP and the layered network models.

As a complement to SDN and Network Function Virtualization (NFV), SDP can secure connections over the IP infrastructure that SDNs create.

2 See paper on SDP and Network Function Virtualization (NFV) - https://www.waverleylabs.com/resources/publications/

Based upon prior feedback and lessons learned from implementations of SDP, this updated specification helps to clarify access control issues within the various deployment models defined in the original specification, and elaborated upon in the SDP Architecture Guide[3].

SDP provides a demonstrably better approach to prevent, detect, and respond to network and cross-domain attacks on applications and infrastructure. It does so by minimizing the attack surface and enforcing the principle of least privilege, even when users access resources over untrusted public networks like the Internet.

While traditional cybersecurity solutions focus on securing networks and systems, SDP focuses on securing assets from an identity-centric perspective. This shift away from perimeter-centric security makes enterprises more resilient to attacks such as DDoS, credential theft, and ransomware on private, enterprise-controlled resources.

SDP Concept

SDP focuses on protecting the critical organizational resources instead of the organizational perimeter. It enables the definition and enforcement of risk-based, dynamic, identity-centric, and contextually-aware access policies, across all layers of the network. SDP provides a basis for defining and enforcing access policies in terms that are meaningful to business, application, and network stakeholders – often for the first time within an organization.

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

- securely deploy services onto networks presumed to be compromised (i.e., "assume breach").
- provide identities with finely-tuned access to services across untrusted networks. One such use case would be VPN replacement.

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

The principles behind SDP are not entirely new. Multiple organizations within the Department of Defense (DoD) and Intelligence Community (IC) have implemented such network architectures based on authentication and authorization prior to accessing the network. Typically used in classified or "high-side" networks (such as defined by the DoD), every server is hidden behind a remote access gateway appliance to which a user must authenticate before authorized services are visible and access is provided. SDPs leverage the logical model used in classified networks and incorporated into standard workflows. Security leaders have been espousing many of these concepts for years, most notably starting at 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[4]

3 https://cloudsecurityalliance.org/artifacts/sdp-architecture-guide-v2/

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

SDPs maintain the benefits of the need-to-know (least-privilege) model described above but eliminate the disadvantages of requiring a remote access gateway appliance. In fact, SDP is designed to apply access controls for *all* users, not just for *remote* users. 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 summary, SDPs securely authenticate resources (users, devices, and services) into a perimeter in which services remain obfuscated to unauthorized resources.

SDP Architecture and Components

In its simplest form, SDP consists of two logical components: SDP Hosts and SDP Controllers. SDP Hosts, which may be full-stack hosts or lighter-weight services, can either initiate or accept connections. These actions are managed by interactions with the SDP Controller via a secure channel over a *control plane*. Data is communicated over a separate secure channel in the *data plane*. 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 availability purposes. SDP hosts (Initiating or Accepting) communicate with the SDP controller, which is an appliance or server process that ensures users are authenticated and authorized, devices are validated, secure communications are established, and data and control traffic on a network remain separate.

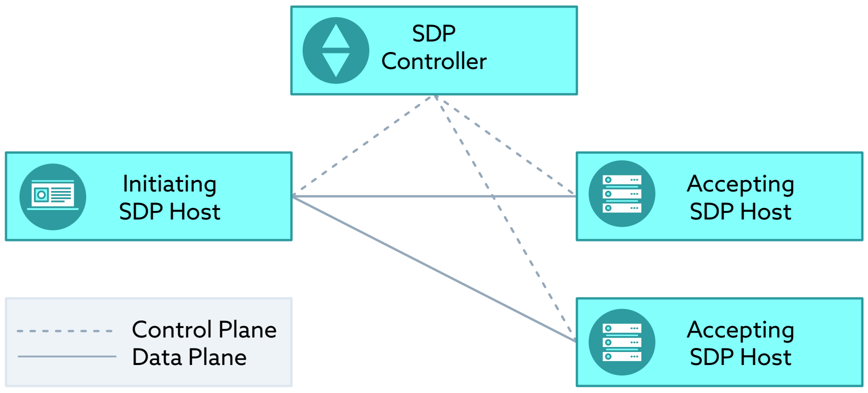


Figure 1: SDP Architecture (previously published by CSA in Software Defined Perimeter and Zero Trust)[5]

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 wherein access policies are defined and evaluated. It acts as a Zero Trust Policy Decision Point (PDP). The SDP Controller is responsible for communicating with enterprise authentication solutions (e.g., identity providers, multi-factor authentication services) to orchestrate authentication and authorization. It is a control point for visibility and reporting of connections allowed by the defined access policies.
- **Initiating Hosts (IH)** - These accessing entities can be user devices or NPE's (Non-Person Entities) such as hardware (e.g., end-user devices, or servers), network devices (for

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

connecting networks), software applications, and services. An SDP user can use an SDP client or browser to initiate.

- **Accepting Hosts (AH)** - These entities are logical components that front applications, services, and resources being accessed and protected by the SDP. The AH acts as a Zero Trust Policy Enforcement Point (PEP). A PEP may be implemented in SDP-specific software or hardware. It allows or disallows network traffic to a target service (which may be an application, a lightweight service, or a resource) based on instructions from the SDP controller. Logically, the AH may be co-resident with the target service, or separated by a network.

These SDP components can be located on-prem or in the cloud, and can be deployed redundantly for scale or availability purposes.

We now examine each component below.

SDP Controller

The SDP Controller is a policy definition, verification, and decision mechanism–a Zero Trust Policy Decision Point–that maintains information about which identities (e.g., users and groups) from which devices should have access to an organization's services (on-premises or in the cloud). 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 which includes 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 (on-premises or in the cloud), and may enforce Multi-factor Authentication (MFA). Authentication is typically 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, or using Identity Federation.

To authorize a user access to a service, the Controller may use an internal user-to-service mapping and policy model, or a third-party service such as LDAP, Active Directory, or other authorization solution (on-premises or in the cloud). Authorization is typically determined by user roles and fine-grained information, based on user or device attributes, or the actual data element or data flow that the user is authorized to access. 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 enables 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 validation 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.

The SDP controller, which closely aligns with the Zero Trust PDP concept, may reside in the cloud or on-premises, depending on the SDP architecture configuration.

SDP Controllers are protected by an isolation mechanism using the Single Packet Authorization (SPA) protocol, making them invisible and inaccessible to unauthorized users and devices. This mechanism may be provided by an SDP Gateway fronting a Controller, or natively within the Controller itself.

SDP Initiating Hosts (and SDP Clients)

SDP 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 information such as user identity, hardware or software inventory, and device health during the authentication phase. The Controller must also provide a mechanism (e.g., credentials) for the 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 a web browser. Using a client application provides richer capabilities such as host checking (device posture checks), traffic routing, and more streamlined authentication.

One of the most important roles of the Initiating Host (IH) is to initiate connections using SPA, as discussed later in this document). SPA packets may be generated by browser-based SDP clients in some implementations. The IH can be a human user's device (e.g., employee or contractor computer, or mobile device), an application (e.g., thick client), or an Internet of Things (IoT) device (e.g., remote water meter). In this latter example, the identity is a nonhuman identity which is nonetheless authenticated and authorized. See the section on *SDP and IoT Devices* for further discussion.

SDP Accepting Hosts (AH)

Accepting Hosts (AH) are the SDP policy enforcement points (PEPs) that control access to any resource (or service) to which an identity might need to connect, and to which the responsible enterprise needs to hide and control access. AHs can be located on-premises, in a private cloud, public cloud, etc.

A service protected by an AH is not limited to being only a web application; it can be any TCP- or UDP-based application, e.g., SSH, RDP, SFTP, SMB, or proprietary applications accessed by fat clients.

By default, all network access to an AH is blocked, and it can only be accessed by authenticated and authorized entities.

As stated above, the AH may be co-resident with the target service, or separated from it via a network. These models are depicted below in Figure 2

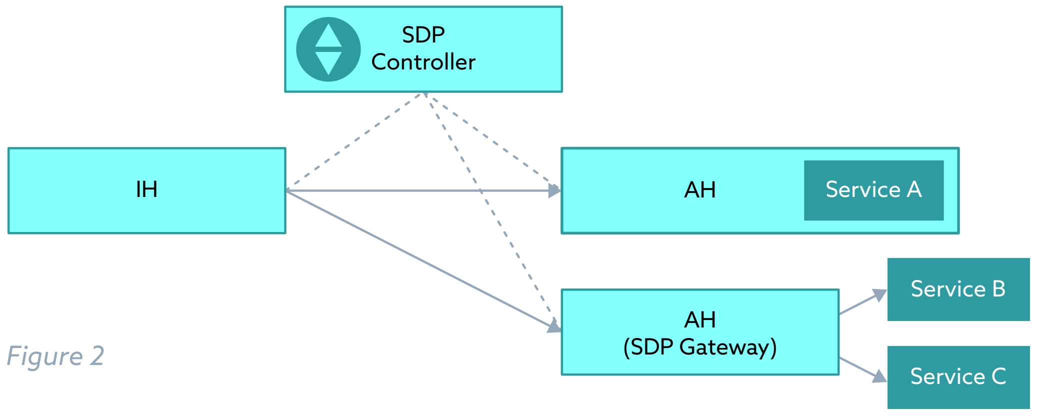


Figure 2

AHs receive control information from the SDP Controller, and accept connections from an IH only after being instructed to do so by the Controller. Using the control information received from the SDP controller, the AH provides authorized IHs (users and devices) access to protected services.

The AH acts as an exchange for secure communication or access 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.

SDP is a logically connection-oriented protocol, securing multiple network deployment topologies. 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 multiple configurations of SDP components in several deployment models.

SDP Deployment Models

SDP connects clients (humans or non-person entities) to resources depicted as servers in the diagrams that follow. 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,services running on IaaS or PaaS platforms, or containerized services.

This section briefly introduces the six SDP deployment models. While they use different network topologies, they logically provide the same benefit of enforcing access to protected resources. For a complete introduction to these deployment models, see the SDP Architecture Guide[6].

Note that in the diagrams below, the blue lines indicate network connections secured by mutually authenticated cryptography protocols, such as mTLS or Internet Key Exchange (IKE), that provide countermeasures to MITM attacks. The gray lines indicate the network connections utilizing the native application protocol, which may or may not be encrypted. The diagrams omit the SDP Controller for simplicity.

Several of the deployment models use SDP Gateways. An SDP Gateway is a software agent that acts as an SDP Accepting Host, and provides isolation and access controls for the protected services.

6 For a complete description, see the CSA Software-Defined Perimeter Architecture Guide, May 2019, pages 14 to 18

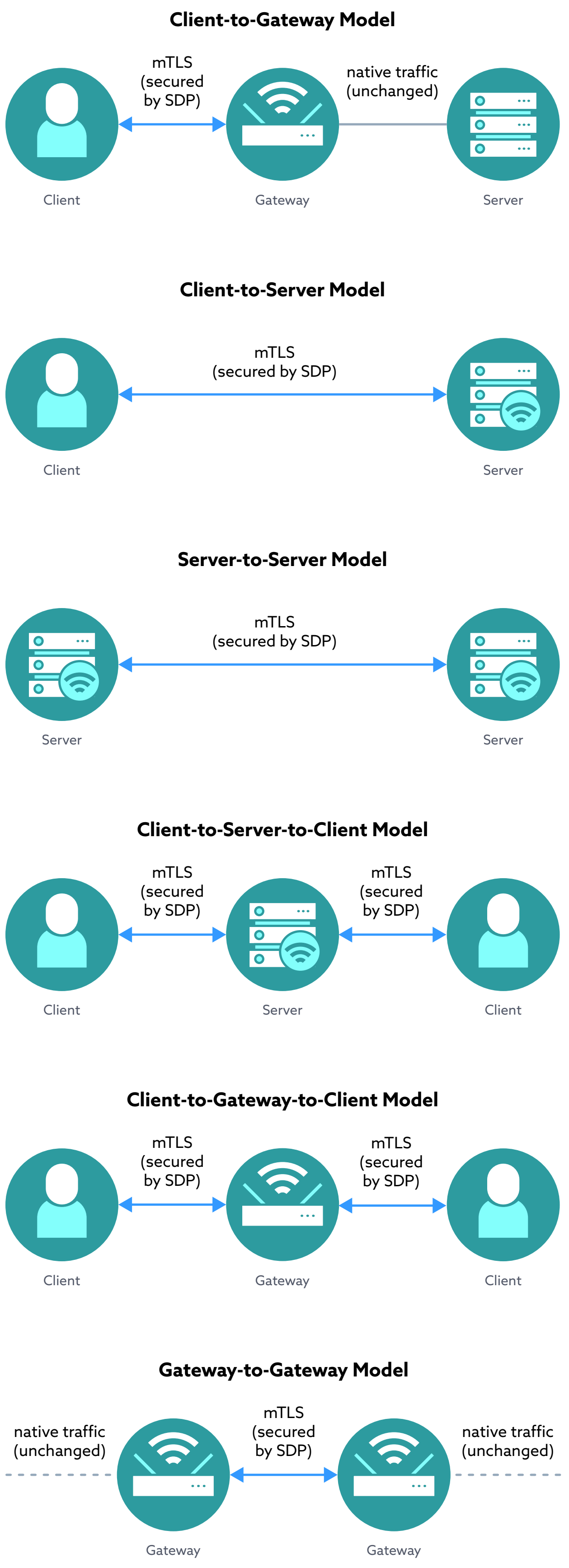


Figure 3 - SDP Deployment Models

When one or more servers must be protected behind an SDP Gateway, the connections between the IH and the AH (Gateway) are secured regardless of underlying network topology. In the Client-to-Gateway model, the Gateway is remotely accessible but hidden, and provides a secure perimeter. This model does not require any changes to protected servers.

When an organization needs to secure communications end-to-end, the Client-to-Server model combines the service and the AH in a single host. In this model, the server is hidden within the secure perimeter. In this model, the server must have SDP software to act as an AH.

The Server-to-Server model ensures that all connections between servers are encrypted regardless of the underlying network. In this model, all servers are hidden within the secure perimeter.

In some instances, peer-to-peer traffic passes through an intermediary server, such as with VOIP, chat, and video conferencing services. In the Client-to-Server-to-Client model, the server is hidden within the secure perimeter.

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

The Gateway-to-Gateway model was not included in the SDP Specification 1.0. This model is well-suited for certain IoT environments. In this model, the gateways are hidden within the perimeter.

SDP Workflows

In general, SDP component workflows are typically split into two separate types: Onboarding Workflows (separate flows for each component) and an Access Workflow (coordinated among multiple components). By definition, each SDP component has a single onboarding workflow, followed by participation in multiple access workflows.

These workflows are representative, since specifics will vary between different implementations and across the different SDP deployment models.

Controller Onboarding Workflows

Every SDP system requires one or more Controllers. At least one Controller must be available at all times for any onboarding flows to succeed. Some implementations also require an active Controller for access workflows to succeed. Controllers must be network-accessible from all locations in which other SDP components operate; thus they are typically globally accessible from the Internet, but only for authorized users/devices.

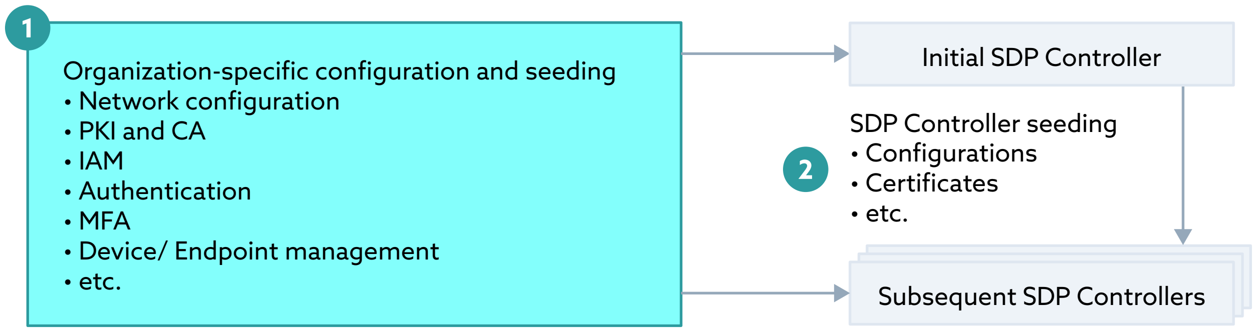


Figure 4 - Controller Onboarding Workflow

This workflow is depicted in Figure 4: An initial (primary) SDP Controller is brought into service 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). The SDP Controller will run continuously and indefinitely, for immediate availability to any other SDP component. Optionally, subsequent Controllers are brought online, and onboarded with a combination of the same organization-specific configuration, as well as seeding (configuration) information from the initial Controller.

Many SDP implementations will support deploying a cluster of Controllers for load balancing and redundancy purposes. Each SDP implementation must support a mechanism by which subsequent Controllers are connected to other Controllers in its cluster, and any appropriate state is shared or accessible. This mechanism is implementation-dependent, and outside the scope of this specification.

Accepting Host (AH) Onboarding Workflow

Every SDP system requires one or more AHs. They may be deployed using any of the SDP Deployment Models depicted above. That is, they may be standalone Gateways, or deployed as part of a server (resource/workload).

AHs may be long-lived or ephemeral–both are acceptable in SDP implementations. Standalone Gateway AHs may be long-lived, such as when an AH system runs for months or years. However, they may alternatively be short-lived, such as within a dynamic cluster of Gateways that expands or contracts based on load.

AHs deployed within an individual server (workload) may also be long-lived or short-lived. In this case, their lifespan will be tied to that of the server instance they are part of. Server instances may be long-lived, such as a traditional Web or Application server, or short-lived, such as part of a DevOps infrastructure.

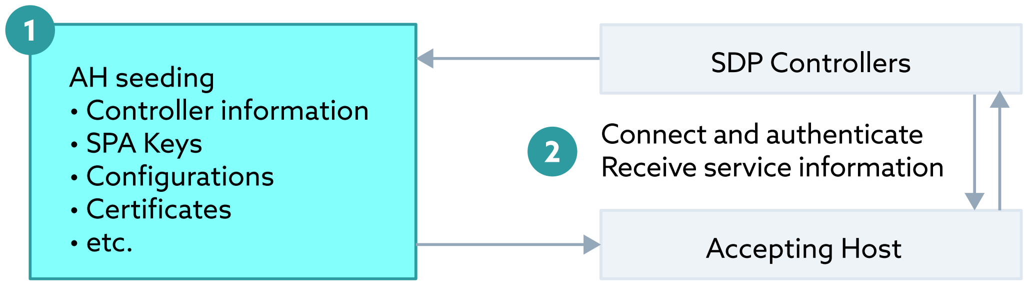


Figure 5 - Accepting Host Onboarding Workflow

The AH onboarding workflow is depicted in Figure 5: When the AHs are brought into service, they must connect and authenticate to one or more Controllers in their SDP. Once onboarded, they are ready to receive SPA packets and process access from authorized IHs.

Each SDP implementation must support a mechanism by which AHs are configured to connect to the Controllers in its cluster. This mechanism is implementation-dependent, and outside the scope of this specification. Likewise, many SDP implementations will support implementation-specific deployment of a cluster of AHs for load balancing and redundancy purposes, most commonly with one of the Gateway deployment models.

Initiating Host (IH) Onboarding Workflow

Initiating Hosts (IHs) may be user devices, or systems without a human user (such as an IOT device, or a server acting as an IH). Like all components in an SDP system, IHs need to be onboarded, which is depicted in Figure 6. During this workflow, they are configured with initial information needed to connect to the Controller. This will include network information (hostnames or IP addresses), as well as any necessary shared secrets (e.g., SPA keys, certificates). The mechanism by which this occurs is implementation-dependent, and outside the scope of this specification. Typically, this involves using an enterprise identity management provider, and for user devices, user authentication is often a step within the IH onboarding workflow.

IHs only need to be onboarded once, after which they can initiate the *access workflow*.

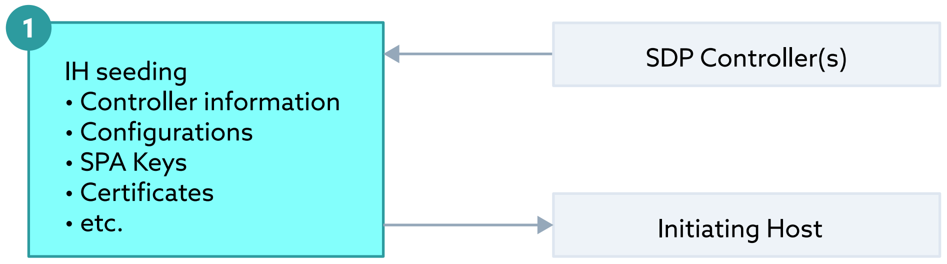


Figure 6 - Initiating Host Onboarding Workflow

Access Workflow

IHs initiate the *access workflow* to connect to workloads (resources) protected by SDP systems, following the steps shown in Table 1, and in FIgure 7. The access flow involves all components of the SDP: IH, Controller, and AH.

Step	Access Workflow
1	When an onboarded IH returns online (e.g., after device reboot, or when the 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 Controller determines a list of services (via AHs) with which the IH is authorized to communicate. The Controller does not yet communicate this list to the IH; this must wait until after step 3.
3	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.
4	The SDP Controller gives the IH the list of authorized AHs and services as well as any optional information required for two-way encrypted communications.
5	The IH uses a 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.

Table 1: Access Workflow

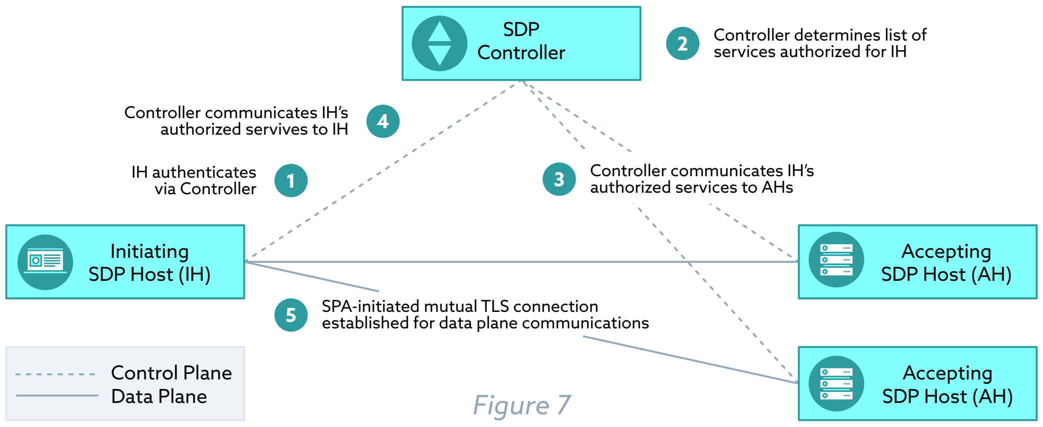

SDP Controller
2 Controller determines list of services authorized for IH
4 Controller communicates IH's authorized servives to IH
1 IH authenticates via Controller
3 Controller communicates IH's authorized services to AHs
Initiating SDP Host (IH)
Accepting SDP Host (AH)
5 SPA-initiated mutual TLS connection established for data plane communications
Accepting SDP Host (AH)
Control Plane
Data Plane

Figure 7

IH Onboarding Workflow Example

The onboarding of SDP Controllers, IHs,AHs, and users varies depending on the deployment models shown in Figure 2, as well as implementation specifics. The establishment and operation of an SDP system may be conducted via an Application Programming Interface (API) or an administrative UI.

As described in the workflows above, the SDP system is deployed and configured, and SDP Controllers and AHs are brought online.

The following sequence is an example[7] onboarding workflow for users of IHs that authenticate with an external Identity Provider (IdP).

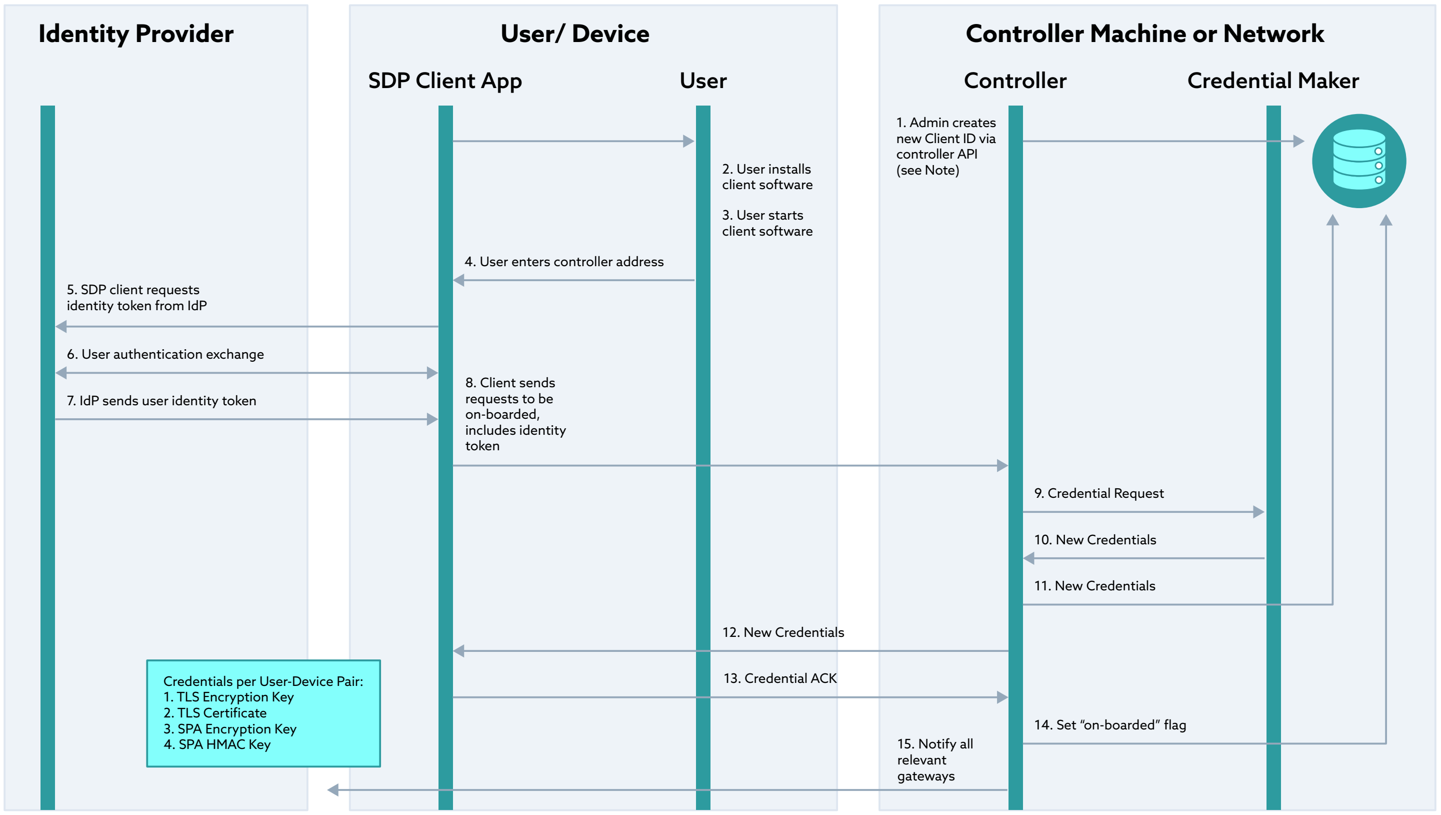

Identity Provider
User/ Device
SDP Client App
User
Controller Machine or Network
Controller
Credential Maker
1. Admin creates new Client ID via controller API (see Note)
2. User installs client software
3. User starts client software
4. User enters controller address
5. SDP client requests identity token from IdP
6. User authentication exchange
7. IdP sends user identity token
8. Client sends requests 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 8 - Onboarding (IdP Scenario)

7 Based upon SDP Open Source Reference Implementation - https://www.waverleylabs.com/open-source-sdp/

Note - This example uses Identity 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. This example also omits the implementation-dependent distribution of key material (such as the SPA key, or other shared secrets) to the nodes. For example, the open source SDP implementation generates this key material, and then relies on the enterprise to securely transmit it to components of the SDP in some secure fashion.

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.

Single Packet Authorization (SPA)

One of the core principles of SDP is that not just workloads (resources), but the SDP infrastructure itself must be inaccessible to unauthorized entities. This is enforced by requiring that entities 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 user credentials. This is in stark contrast to traditional remote-access solutions, such as VPNs, which are too often exposed to all malicious actors on the internet.[8]

The mechanism that SDP uses for this is Single Packet Authorization (SPA). The SDP version of SPA is based on an RFC 4226 HMAC-based One-Time Password "HOTP"[9], which is included in the SPA packet as described below.

Note that while the SDP specification is oriented around the use of SPA to secure access to Controllers and AHs, there are credible alternatives that can work with appropriate architectures. See the *Alternatives to SPA* section below.

The key principles that SPA achieves within SDP are the following:

- **Cloaks SDP system components:** Neither Controllers nor AHs will respond to any attempted connections from a remote system until they have provided an authentic SPA packet that is valid for that SDP system. Specifically, in the case of UDP-based SPA, the host will not respond to a TCP SYN, thereby avoiding the disclosure of any information to a potential attacker. (A default-drop firewall is an example of how this could be implemented). This is true both for standalone AHs (SDP Gateways) as well as AHs that are logically part of a server/workload.
- **Mitigates Denial of Service attacks on TLS:** Internet-facing servers running the HTTPS (which uses TLS) protocol are highly susceptible to Denial-of-Service (DoS) or Distributed Denial-of-Service (DDoS) attacks. SPA mitigates these attacks because it allows the server to quickly reject unauthorized connection attempts before incurring the overhead of

8 to be fair, there are some commercial and open source VPN-like solutions that do hide the infrastructure. Wireguard is one open such example.

9 See https://tools.ietf.org/html/rfc4226

establishing a TCP or TLS connection, and therefore allowing authorized connections during and in spite of DoS attacks.
- **Attack detection:** The first packet to a Controller or 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.

Note that, by design, 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, *SDP as a DDoS Defense Mechanism* whitepaper[10]).

SPA initiates communication for 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 all the security benefits listed above to the SPA-protected server. 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 cloaked. 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 TCP SPA, TCP will permit the server to detect an attack based on an invalid SPA packet, but only after the TCP connection is established, thereby consuming server resources.

Another consideration for minimizing service exposure is DNS enumeration of the AHs and Controllers. Connection directly via IP or via only a private DNS service may reduce service visibility to attackers. That is, the very presence of public DNS entries for SDP components - such as `controller1.sdp.mycompany.com` - may itself represent an attack surface for a malicious actor, for example, via a volumetric DDoS attack on the SDP infrastructure.

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 should 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 shared secrets in order to construct a valid SPA packet. Establishing this root of trust -- namely, how the shared secret is securely communicated to SDP components – is implementation-dependent, and outside the scope of this specification. Typically this information is included in the onboarding process for IHs and AHs.

10 https://cloudsecurityalliance.org/artifacts/software-defined-perimeter-as-a-ddos-prevention-mechanism/

ClientID	256-bit numeric identifier, assigned per user-device pair. This field is used to distinguish the user, device, or logical group that is sending the packet.
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 (for example, 15 to 30 seconds). This also provides a mechanism to reduce the replay-detection caching required for the recipient.
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 (hashed OTP) is generated by an algorithm as described by RFC 4226, based on a shared secret. The use of an OTP is required in SPA packets to establish authenticity; other OTP algorithms can be substituted with the overarching goal of providing authenticity of the SPA packet.
HMAC	Calculated over all fields above. Algorithm choices are SHA256 (recommended), SHA384, SHA512, SM3, Equihash, or other efficient and robust algorithms. 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 are immediately discarded.

Table 2: 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 by the recipient 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 not just as a means for initiating a connection, but as the means for 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 Message String field can be used to transmit meaningful data by the client. This does not require any further processing or policy evaluation, and does not require establishing a TCP or TLS connection.

This can 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 (AH) 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 not critical.

Alternatives to SPA

The SDP architecture is designed with flexibility in mind, and this revision of the specification embraces the diversity of approaches and deployment models that may be created. For example, SDP defines six deployment models, each of which may lend itself to solving different problems, and be implemented in different ways, with different tradeoffs. SPA is a sound and secure mechanism for achieving the key SDP principles of cloaking the infrastructure, providing resiliency against DDoS attacks, and improving the ability to detect attacks. SPA also has the advantage of making SDP a self-contained system; once seeded, IHs can securely and reliably establish encrypted communications with Controllers and AHs, without relying on any external systems.

However, SPA is not the only mechanism by which a system can achieve these goals. There are alternative approaches that deliver the principles defined here, and which can therefore be part of a system that supports SDP and Zero Trust principles. For example, a SPA-less SDP system could utilize a globally accessible enterprise identity provider, with a control channel to the SDPs' Controllers and AHs. In this model, an IH authentication to the identity provider would trigger a control plane message to the SDP Controllers and AHs, informing them of the IH's successful authentication, and to expect an immediate incoming connection from the IH's public IP address. The IH would then be able to establish a TCP connection to the Controller and AHs, which would be expecting this. The TCP connection would, of course, be followed by (mutually-authenticated TLS (mTLS) and other layers of security as defined below.

Every system and architecture has its own tradeoffs, of course. As long as the approach achieves the three principles noted above, a system using an alternative to SPA can be part of a sound and valuable SDP system.

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 become validated and authorized, connections between the primary components in an SDP system must use TLS or other alternatives such as 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), the Initiating Host to Controller (IH-Controller), and Controller-AH connection must use mutual authentication. Note that TLS is supported over TCP as well as UDP (in which case it's referred to as Datagram Transport Layer Security, or DTLS). Both are acceptable for SDP systems.

Weak cipher suites or protocols that do not support mutual authentication are ill-advised and not secure. Strong ciphers and protocols ensure that the components each contain a valid private key, issued by a trusted authority, significantly reducing the likelihood of a Man-in-the-Middle (MITM) attack[11] by verifying signed certificates in both directions.

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 Online Certificate Status Protocol (OCSP)[12] or other mechanisms.

Whichever transport layer approach an SDP system uses, it must be resilient to potential attacks - such as a MITM TLS downgrade protocol attack (which mTLS avoids).

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 validate that the device meets security or configuration requirements.

The objective of Device Validation is to prove that the device meets security requirements. Note that in enterprise environments, Controllers are assumed to be trusted devices because they exist in the most controlled environments. SDP Gateways are also assumed to be trusted components if they are under the control of the enterprise operating the SDPs. That is, these components run in controlled and managed environments, which must be subject to enterprise change management controls and deployment processes, and must be deliberately onboarded into the SDP system. For Controller and Gateway components, Device Validation should be considered optional.

User devices (IHs), on the other hand, require Device Validation. Device Validation mitigates credential theft and the resultant impersonation attacks.[13] User devices are allowed to connect to the SDP Controller only after they successfully meet Device Validation and overall security policy requirements. This process ensures that an attacker will not be able to access services on or behind the AH, even if the attacker is in possession of the correct private keys for mTLS authentication.

11 https://wott.io/blog/tutorials/2019/09/09/what-is-mtls

12 https://tools.ietf.org/html/rfc6960

13 The private keys and SPA secrets should be securely stored and rotated on a regular basis. This is an area of interest for future work.

Packets are dropped from any device unless authenticated and authorized via the SDP, thus the process prevents any incoming packets from unauthorized devices. Device Validation protocols and details are enterprise- and product-specific, and as such are beyond the scope of the SDP specification, but are an area of interest for future work.

SDP systems must support the ability to integrate with enterprise device management / endpoint management systems, and include their device security posture checks into a Device Validation process. In addition, SDP systems should support the ability to perform local device security posture checks in environments where the SDP has software running locally on the user device. For example, an SDP client could verify that a user device contains a valid certificate issued by the enterprise, which may be used for mTLS authentication. 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.

SDP and IoT Devices

Two of the core Zero Trust tenets from NIST 800-207 state that a system must secure access for "all data sources and computing services,", and "all communication,"[14] which must therefore include IoT devices. We consider IoT broadly, to include devices that exist in most IT environments (such as printers, IP cameras, and VOIP phones), as well as more specialized devices such as environmental or medical sensors, or industrial control systems.

If these devices are open, and can support the installation and operation of arbitrary and fully functional software to act as a normal SDP Initiating Host, they can be treated no differently than a typical host operating as an IH. Our discussion here is restricted to those devices which fall into one of three categories -- they are closed (non-extensible) devices on which enterprises cannot install additional software, they are general-purpose computers (such as Arduinos), but have limited compute/memory/storage capacity, or they are fully-functional general purpose systems onto which the enterprise has chosen not to deploy SDP client software. In all cases, these are networked devices, and access to and from them should be included in an SDP architecture and policy model.

In order to be included in an SDP system, non-extensible devices need an "upstream" network device (a "broker" or a "connector") to capture IoT device network traffic, and broker it into the SDP system. The SDP broker acts as the logical IH on behalf of the IoT devices. Note that this broker should be able to control traffic both to and from the IoT devices.

Limited capacity devices can run some variant of SDP software on them, but cannot and should not act as a full-featured IH. SDP is open to alternative models for communications from,and potentially to, these devices. For example, an array of low-power, low-capacity environmental sensors could run a lightweight SDP client that only generates valid SPA packets to securely transmit the data. Not needing to establish a TLS connection or to authenticate a very computationally lightweight way to securely[15] transport data.

14 See NIST Special Publication 800-207, Zero Trust Architecture, Page 6

15 This provides message integrity but not privacy since the data isn't encrypted in transport (although it could be encrypted

These ideas are good starting points for future work around Zero Trust and IoT, and we look forward to additional research and publications in this area.

Access Policies

Zero Trust is firmly centered on the idea of *access policies* controlling access to protected resources -- recall the key Zero Trust concepts of the *Policy Decision Point* and the *Policy Enforcement Point*. This is explicitly called out in one of the core tenets of the NIST *Zero Trust Architecture* document: "Access to resources is determined by dynamic policy—including the observable state of client identity, application/service, and the requesting asset—and may include other behavioral and environmental attributes."[16] And the Cybersecurity & Infrastructure Security Agency (CISA) Zero Trust Maturity Model discusses how policies shift from being static at the *Traditional* maturity level, to using cross-domains inputs and outputs at the Advanced level, to being "dynamic policies based on automated/observed triggers"[17] at the *Optimal* level.

However, defining an effective, yet broadly applicable policy model framework and associated vocabulary is difficult, given the wide variety of Zero Trust architectures and enterprise requirements. For example, SDP alone has six different deployment models, each with slightly different enforcement capabilities. The result of this breadth is that early Zero Trust documents were often silent on the topic of policies; for example, version 1 of this SDP specification did not discuss or define a policy model. Newer Zero Trust models provide us with some guidance on how to approach policy.

NIST 800-207 introduces the concept of a *Policy Engine* evaluating a *Trust Algorithm*, depicted at a conceptual level in Figure 9.

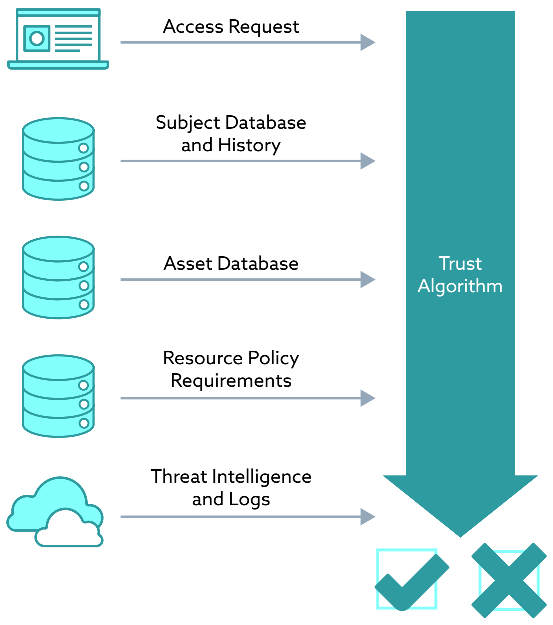

Access Request
Subject Database and History
Asset Database
Resource Policy Requirements
Threat Intelligence and Logs
Trust Algorithm

Figure 9: Trust Algorithm figure from NIST 800-207

NIST 800-207 touches on a policy model where it discusses resource policy requirements:

> "This set of policies complements the user ID and attributes database [SP800-63] and defines the minimal requirements for access to the resource. Requirements may include authentication assurance levels, such as MFA network location (e.g., deny access from overseas IP addresses), data sensitivity, and requests for asset configuration. These requirements should be developed by both the data custodian (i.e., those responsible for the data) and those responsible for the business processes that utilize the data (i.e., those responsible for the mission)."

with a separately distributed key).

16 See NIST Special Publication 800-207, Zero Trust Architecture, Page 6

17 US Cybersecurity and Infrastructure Security Agency Cybersecurity Division, Zero Trust Maturity Model, Pre-decisional Draft, June 2021, Version 1.0, Page 5.

The CISA Zero Trust Maturity Model does not directly address policy, but does so indirectly by giving examples of system capabilities at different maturity levels that are expressible in policies.

Defining a structure and vocabulary for a universally applicable Zero Trust policy model is challenging. This is inherent in the nature of Zero Trust as a security philosophy. The SDP architecture is more specific, and we think it is well-suited to a structured policy model, although there are differences across SDP deployment models. For example, consider a device posture check policy that requires verifying that the device contains a corporate-issued certificate. This is straightforward to enforce in a system with a local agent running on a user's device, but is more difficult in an agentless scenario.[18] These types of real-world constraints result in industry research and guidance documents (which by necessity must be implementation-neutral) that are limited to more generic policy aspects (e.g., ensure user devices meet enterprise security posture checks) rather than specific and detailed policy models.

A Zero Trust policy model is not in scope for this v2 SDP specification, but it is an interesting area for future work within the SDP and ZT Working Group, as well as across the industry at large. We look forward to future collaboration and publications on this topic.

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 communications using SPA between SDP components, but it is not normative -- different SDP deployment models will necessitate different interactions and messages. The goal here is to show a representative system that works, not to prescribe a fixed or standardized message format.

Note also that this protocol excludes messages or data flow for component or device onboarding, as noted previously.

Note that for clarity, the protocol described here does not specify timeouts, retries, or other error-handling mechanisms. Note also that during the IH login process, the Controller may make additional external calls, for example to an enterprise identity management system to authenticate the IH. Finally, note that the examples below depict the use of TCP and TLS connections between components. As noted previously, using a connectionless UDP-based protocol (namely, DTLS) is also acceptable, but not depicted below for clarity.

18 And may be impossible to achieve without an agent in some Operating Systems

AH-Controller Protocol

AH to Controller Sequence Diagram

The protocol sequence diagram for the AH to connect to the Controller is shown in FIgure 10. A UDP-based SPA packet is the first packet, in order to ensure that the SDP Controller is protected from unauthorized access.

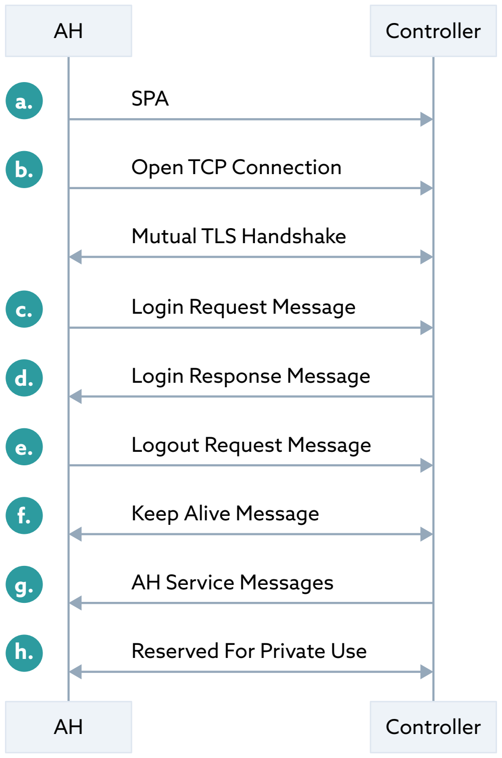


Figure 10: Accepting Host connects to the Controller

The following subsections define the various messages and formats passed between the AH and Controller. The basic message format is:

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

a. SPA

The SPA packet is sent by the AH to the Controller to request a connection, following the format discussed earlier in this document.

b. Open connection and establish mutually authenticated communications

After sending the SPA packet, the IH attempts to open a TCP connection to the Controller. If the Controller determined that the SPA packet was valid, it will permit this TCP connection to be established. This is followed by the required mutual authentication for the establishment of an mTLS connection.

In the case of UDP-based DTLS, this is a logical connection, since UDP is a connectionless protocol.

c. Login (SDP Join) Request Message

The login request message is sent by the AH to the Controller to indicate that it is operational and requesting to join the SDP as an active AH. Note that the AH login request *may* include credentials with which the AH identifies and authenticates itself to the Controller.

`0x00`	`No command-specific data`

d. Login (SDP Join) Response Message

The login response message is sent by the Controller to indicate whether the login request was successful and, if so, to provide the AH Session ID. Note that it is possible that the Controller will reject the AH's login request. This could be due to, for example, invalid credentials. Or, the Controller could be enforcing system license or scale limits, and reject this AH's attempt for those reasons. In both these cases, the connection will be denied, and the Controller will terminate the TCP connection from the AH.

`0x01`	`Status Code (16 bits)`	`AH Session ID (256 bits) and optionally, renewed SPA keys and TLS keys for the AH.`

The Status Code is implementation dependent. The AH Session ID is used within the system logs. The JSON specification for this as an example is:

Format	Example

`{ “session_id” : <256 bits>,`
`“credentials”:`
`[“spa_encryption_key”:<64 bit>,`
 	
`“spa_hmac_key”: <64 bit>,`
 	
`“tls_key”: <file contents>,`
 	
`“tls_cert” <file contents>`
 	
`]`
`}`
	`{ “session_id”: “0x1234…”`
	`“credentials”:`
	`[“spa_encryption_key”:”aldskf…”,`
	
	`“spa_hmac_key”: “asldjf…”,`
	
	`“tls_key”: “tls_key”,`
	
	`“tls_cert”: “tls_cert”`
	
	`]`
	`}`

e. Logout Request Message

The logout request message is sent by the AH to the Controller to indicate that the AH is no longer available and is not able to accept other messages from the Controller. There is no response message sent by the Controller, and the TLS and TCP connections must then be terminated by either the AH or the Controller.

`0x02`	`No command-specific data`

f. 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`

g. AH Service Messages

The services message is sent by the Controller to indicate the set of services that this AH is protecting. Note that in this example, the service is designated by a fixed IP address, which could be the case when the AH is an SDP Gateway. The service may instead be designated by a hostname, which the AH would need to resolve. If the AH is part of the host on which the service is running (see Figure 1A), it may use the `id` or `name` fields to identify the service to which the AH is controlling access rather than the IP address.

`0x04`	`JSON formatted array of Services`

The JSON specification for the data plane is:

Format	**Example**

`{“services”:`
 	
`[“port”: <Server port>, “id”: <256-bit Service ID>, “address”: <Server IP>, “name”: <service name>`
`“type” : <protocol type  tag >`
`]`
`}`
	`{“services”:`
	`[“port”: “443”,`
	`“id”: “123445678”,`
	
	`“address”: “10.2.1.123”,`
	
	`“name”: “Marketing Web App”,`
	
	`“type”: “HTTPS”`
	
	`]`
	`}`

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

h. Reserved for Private Use

This command (0xff) is reserved for 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).

IH to Controller Sequence Diagram

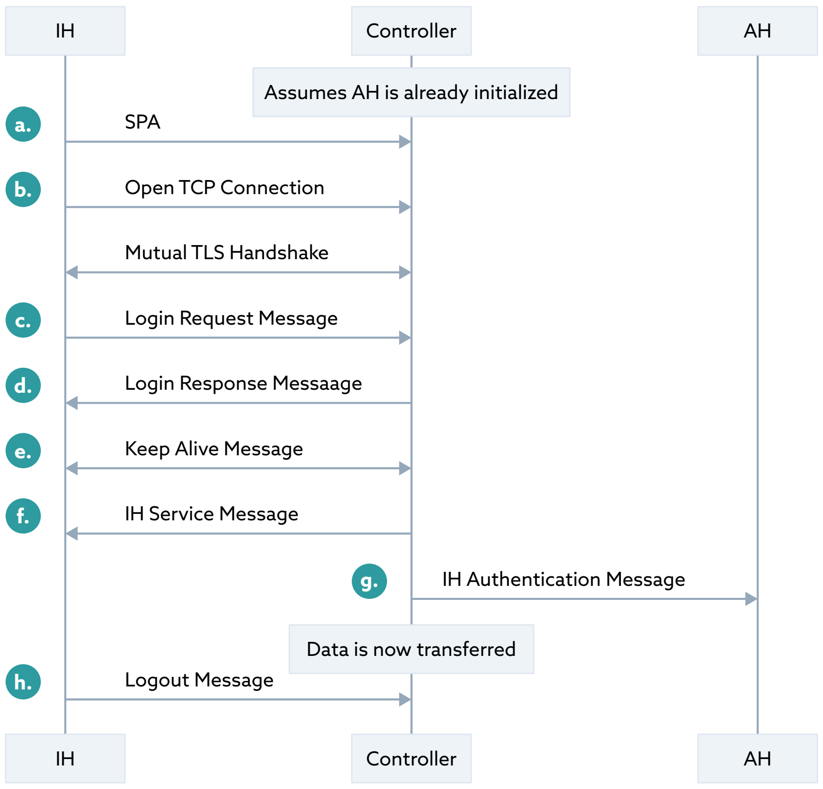


Figure 11: Initiating Host connects to the SDP Controller

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 SPA packet is sent by the IH to the Controller to request a connection, following the format discussed earlier in this document.

b. Open connection and establish mutually authenticated communications

Following its sending of the SPA packet, the IH attempts to open a TCP connection to the Controller. If the Controller determined that the SPA packet was valid, it will permit this TCP connection to be established. This will be followed by the required mutual authentication for the establishment of an mTLS connection.

In the case of UDP-based DTLS, this is a logical connection, since UDP is a connectionless protocol.

c. Login (SDP Join) 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. Note that the IH login request *may* include credentials with which the IH identifies and authenticates itself to the Controller. Recall also that this login request occurs after the onboarding flow depicted earlier in this document. This login request occurs once per session - for example, once per day when a user first turns on their device.

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. Note that it is possible that the Controller will reject the IH's login request. This could be due to, for example, invalid credentials. Or, the Controller could be enforcing system license or scale limits, and reject this IH's attempt for those reasons.

0x01	Status Code (16 bits)	IH Session ID (32 bits) and optionally, renewed SPA keys and TLS keys for the IH.

Format	Example

`{ "session_id" : <256 bits>,`
`"credentials":`
`["spa_encryption_key":<64 bit>,`
`"spa_hmac_key": <64 bit>,`
`"tls_key": <file contents>,`
`"tls_cert" <file contents>`
`]`
`}`
	`{ "session_id": "0x1234…"`
	`"credentials":`
	`["spa_encryption_key":"aldskf…",`
	`"spa_hmac_key": "asldjf…",`
	`"tls_key": "tls_key",`
	`"tls_cert": "tls_cert"`
	`]`
	`}`

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

f. IH Services Message

The services message is sent by the Controller to provide the IH with the list of available services and the IP addresses or hostnames of the AHs protecting them. This message must contain sufficient information for the IH to be able to connect to the service. Note that the hostname/IP address listed is that of the AH which is directly reachable by the IH. The actual service may be running on a different host/IP than the AH, as mentioned in the "Deployment Models" section earlier.

The Service ID is used later by the IH, to identify a target service when communicating with the AH.

0x06	JSON formatted array of Services

The JSON specification is:

Format	Example
{“services”: [{“address” : <AH IP>, “id”: <256-bit Service ID>, “name”: <service name>, “type” : <service type>, “port”: <Server port> }] }	{“services”: [{“address” : “10.2.1.34”, “id”: “123445678”, “name”: “FinanceApp”, “type” : “HTTPS”, “port” : “8443” }] }

g. IH Authenticated 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.

Note that although this message is sent from the Controller to the AH, it is initiated in response to the IH authenticating to the Controller.

0x05	JSON formatted array of IH information

The JSON specification is:

Format	Example
{“IH Authenticators”: “IH”: <IH/Device Pair>, “session_id”: <256-bit IH Session ID>, “hmac_seed”: <256-bit SPA HMAC seed for the IH>, “hotp_seed”: <256-bit SPA HOTP seed for the IH> “id”: [<array of 256-bit service IDs>] }	{“IH Authenticators”: “IH”: “IH/DeviceID”, “session_id”: “4562”, “hmac_seed”: “###”, “hotp_seed”: “###”, “id”: [“123445678”, “9012345”] }

Note that the information in this payload, sent from the Controller to the AH, needs to be sufficient for the AH to validate a future connection from the IH. In the example above, this supposes that the AH needs the IH-specific Device ID, Session ID, and SPA seed. Different implementations may approach this differently.

h. Logout Request Message

The logout request message is sent by the IH to the Controller to indicate that the IH wishes to terminate its SDP session. There is no response message sent by the Controller, and the TLS and TCP connections must then be terminated by either the IH or the Controller. Note that the IH remains onboarded, and can establish a new session again in the future.

`0x07`	`No command-specific data`

z. Reserved for Private Use

This command (0xff) is reserved for any non-standard messages between the IH and the Controller. This message isn't shown in figure 11.

`0xff`	`User-specified`

IH-AH Protocol

The Initiating Host to Accepting Host Protocol utilizes network routing and packet delivery. Implementation details depend on the type of transport (e.g., TCP guaranteed delivery or UDP fire and forget).

It is important to note that the connection is established dynamically through the AH to the service only after the IH has connected with a valid SPA packet, established mutually authenticated communications, and the AH verifies that the IH is authorized to access the requested service. 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 in Figure 12.

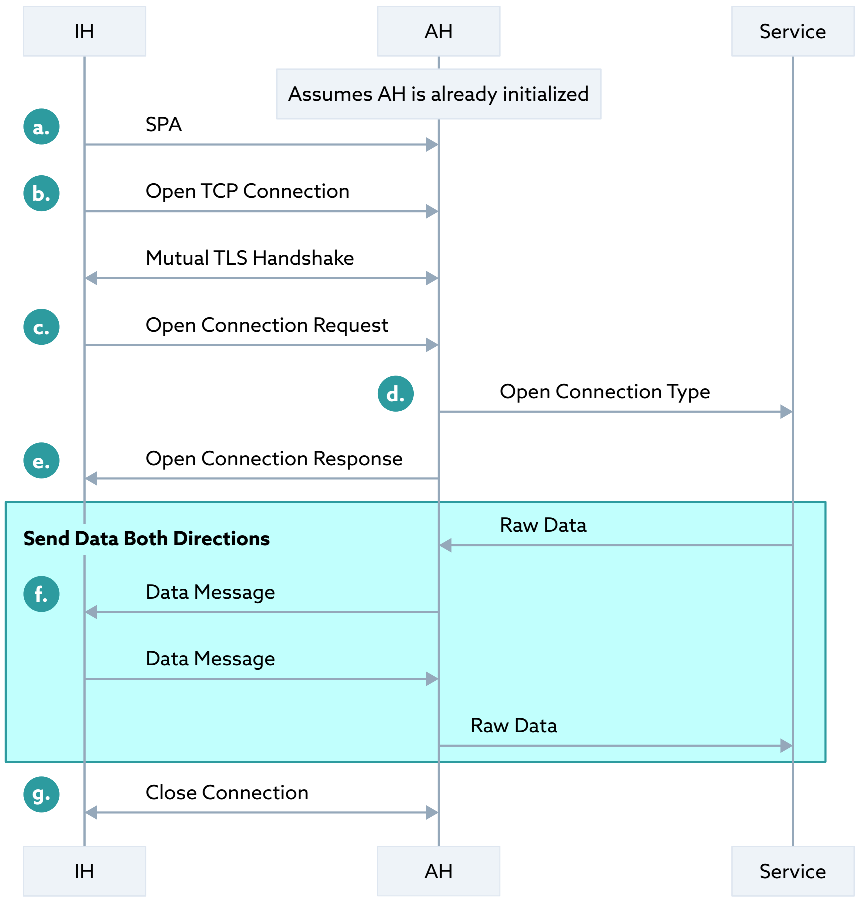


Figure 12: IH connects to an AH and then sends data to a Service

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 SPA packet is sent by the IH to the AH to request a connection following the format discussed previously.

b. Open connection and establish mutually authenticated communications

After sending the SPA packet, the IH attempts to open a TCP connection to the AH. If the AH determines that the SPA packet is valid, it will permit this TCP connection to be established. This will be followed by the required mutual authentication for the establishment of an mTLS connection.

In the case of UDP-based DTLS, this is a logical connection, since UDP is a connectionless protocol.

c. 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.

The Service ID is a unique value assigned by the Controller for each remote service. The IH is aware of Service IDs, because they have been previously sent by the Controller to the IH, as part of the *IH Services* message, depicted above in the IH-Controller protocol.

The Session ID is used by the IH and the AH to differentiate among different TCP connections for a specific remote Service.

`0x07`	`Service ID - 256-bit identifier` `Session ID - 256-bit identifier`

d. Open Appropriate Connection Type

This step, which is specific to the implementation and SDP deployment model, is used by the AH to establish a connection to the service on behalf of the IH. This may not be necessary for certain protocols, such as short-lived HTTPS connections, or connections that require IH-provided application-level authentication as part of the initial exchange (e.g., SSH). SDP-aware services may need this connection to be established to communicate the IH's or user's context from the AH, outside of the application protocol. Depending on the SDP deployment model, this connection may be a network connection, or a local host connection (e.g., inter-process communication).

e. Open Connection Response Message

The open-response message is sent by the AH to the IH to indicate whether the open request was successful. The Open Connection Request and Open Connection Response messages may be asynchronous for the IH, so the Service ID and Session ID codes are useful for the IH to associate this return value with its corresponding request.

`0x08`	`Status Code (16 bits)`	`Service ID - 256-bit identifier` `Session ID - 256-bit identifier`

f. 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. This message is implementation-dependent. Some SDP implementations may package application data in messages such as this, while others may use alternative approaches.

`0x09`	`Data Length (16 bits)`	`Service ID - 256-bit identifier` `Session ID - 256-bit identifier`

g. 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`	`Service ID - 256-bit identifier`
	`Session ID - 256-bit identifier`

z. User-Defined Message

This command (0xff) is reserved for any non-standard messages between the IH and the AH. This message is not shown in Figure 12.

`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 already contained in the additional fields of the log record.
`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 and activities that need to be logged.

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

Security/Connection Logs

Security logs are core to the SDP, and are also important in a broader context to detect larger-scale infrastructure attacks. Therefore, these logs are very useful if forwarded to a Security Information and Event Management (SIEM). They are 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

SPA authentication	*sec:auth*	**SPA** packet from IH to Controller **SPA** packet from AH to Controller **SPA** packet from IH to AH
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

Here is a brief scenario that depicts a complete outage, showing which log entries are written and 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]

> 1. IH tries to reconnect to Controller n times
>   ***ops:conn:reconnect*** log messages
> 2. 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
> 3. IH connects to the newly found Controller
>   ***ops:conn:up*** log message
> 4. 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 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:

> 5. IH connects to Controller
>   ***sec:auth*** log message - SPA authentication to Controller
>   ***ops:conn:up*** log message
> 6. IH mutually authenticates to Controller
>   ***sec:connection*** log message
> 7. IH connects to AH
>   ***sec:auth*** log message
>   ***ops:conn:up*** log message

Summary

SDP is an effective Zero Trust implementation.[19] This revised specification will encourage enterprises to adopt a Zero Trust paradigm for securing their applications, networks, users, and data. This has become critically important given both the shift toward cloud, as well as the ever-heightened threat landscape.

SDP is also proven to secure Enterprises, their use of Infrastructure as a Service cloud offerings, Network Function Virtualization, Software-Defined Networking, and IoT applications

This Version 2.0 of the Software-Defined Perimeter (SDP) Specification has been a long time in coming. Version 1.0 was published approximately eight years ago, in April 2014. During this time period, we've seen growing enthusiasm and adoption of Zero Trust principles, and a corresponding growth of interest and deployments of SDP-based solutions.

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

- SDP and its relationship to Zero Trust
- SDP architecture and components
- Onboarding and access workflows
- SPA message format, use of UDP, and alternatives
- Initial discussions on IoT devices and access policies

Additionally, we have provided enhanced sequence diagrams and explanation of connections and messages in the following three 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 include:

- Zero Trust policy model and SDP (that is, access via Policy decision point (PDP) and corresponding policy enforcement point (PEP))
- Zero Trust for IoT with SDP
- Secure storage and rotation of SDP credentials
- Device Validation

We look forward to future collaboration and publications on the topics above.

19 https://www.helpnetsecurity.com/2020/05/29/sdp-zero-trust/

References

Cloud Security Alliance (CSA), Integrating SDP and DNS: Enhanced Zero Trust policy enforcement - Pending publication Q1 2022 available at https://cloudsecurityalliance.org/artifacts/integrating-sdp-and-dns-enhanced-zero-trust-policy-enforcement/

Cloud Security Alliance (CSA), Software Defined Perimeter (SDP) and Zero Trust, May 27, 2020, available @ https://cloudsecurityalliance.org/artifacts/software-defined-perimeter-and-zero-trust/

Cloud Security Alliance (CSA), Zero Trust Presentation to OMG (Object Management Group), SDP: The Most Advanced Zero Trust Architecture, available @ https://cloudsecurityalliance.org/artifacts/sdp-the-most-advanced-zero-trust-architecture/

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/

National Institute of Standards and Technology (NIST), NIST SP 800-207, Zero Trust Architecture document, August 2020, available @ https://csrc.nist.gov/publications/detail/sp/800-207/final

Cybersecurity and Infrastructure Security Agency Cybersecurity (CISA) Division, Zero Trust Maturity Model (Draft), June 2021 Version 1.0 (Comment Period Ends October 1, 2021), available @ https://www.cisa.gov/publication/zero-trust-maturity-model

Office of Management and Budget, Federal Zero Trust Strategy (Draft), (Comment Period Ends September 21, 2021, available @ https://zerotrust.cyber.gov/federal-zero-trust-strategy/

Cybersecurity and Infrastructure Security Agency (CISA), in collaboration with the United States Digital Service and FedRAMP, (Comment Period Ends October 1, 2021), available @ https://zerotrust.cyber.gov/cloud-security-technical-reference-architecture/

Executive Order 14028 on Improving the Nation's Cybersecurity, May 12, 2021, available @ https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

National Security Agency (NSA), Embracing a Zero Trust Security Model, February 2021, available @ https://www.nsa.gov/Press-Room/Cybersecurity-Advisories-Guidance/smdpage11747/2/

Adoption of the Software-Defined Perimeter (SDP) Architecture for Infrastructure as a Service, by J. Singh, A. Refaey and J. Koilpillai, in Canadian Journal of Electrical and Computer Engineering, vol. 43, no. 4, pp. 357-363, Fall 2020, doi: 10.1109/CJECE.2020.3005316. https://ieeexplore.ieee.org/abstract/document/9240082

On IoT applications: a proposed SDP framework for MQTT, Electronics Letters, by Refaey, A.; Sallam, A.; Shami, A.: 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

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.

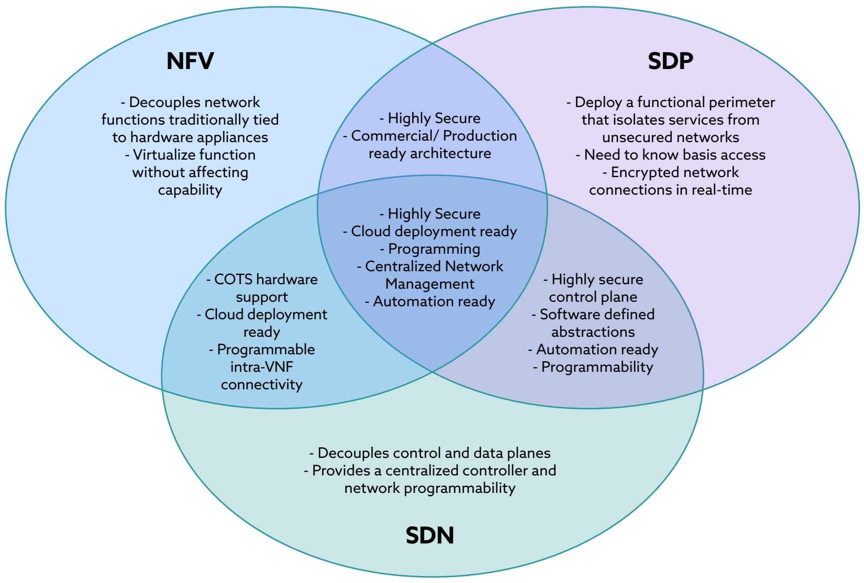
J. Singh, A. Refaey and A. Shami, "Multilevel Security Framework for NFV Based on Software Defined Perimeter," in IEEE Network, vol. 34, no. 5, pp. 114-119, September/October 2020, doi: 10.1109/MNET.011.1900563.

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).

Appendix A: SDP, SDN & NFV

In cloud computing environments, Software Defined Network (SDN) and Network Function Virtualization (NFV) technologies efficiently address challenges[20] 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 securing 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 with NFV is resource exhaustion. The software that uses a particular physical server's resources intensively may exhaust those resources and 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 simultaneously. This problem can be addressed by using SDP – by having the SDP controller define and implement a standard operating procedure that detects VMs that are throttled due to resource exhaustion—similar to DoS—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 through excessive administrative privileges 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.


NFV
- Decouples network functions traditionally tied to hardware appliances
- Virtualize function 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

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.

20 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.

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.

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

Appendix B: OSI / SDP Component Mapping

This section provides a mapping of OSI network layers to cloud layers, cloud stack functions, and SDP components. This mapping aims to show how SDP logically helps provide security at different layers of networking and cloud models.

OSI Layer	Cloud Layer	Cloud Stack	SDP Component
Application	Application	End-User Layer - Provides application and business value	SDP Client - provides user access to applications
Presentation	Service	Middleware - Functional components that applications use in tiers	SDP Controller - acts as middleware for brokering user access to resources
Session	Image	Operating System - Manages underlying virtualization properly	SDP policies - enforcing firewall rules and session management
Transport	Software-Defined Data Center	Cloud API - Enables creation of virtualized assets tied to resource pools/users	Mutual TLS
Network	Hypervisor	Virtualization - Provides virtualization of computing, storage & monitoring	SDP Accepting Hosts (Gateways)
Data Link	Infrastructure	Hardware - Physical devices in the data center	n/a
Physical	Infrastructure	Hardware - Physical devices in the data center	n/a