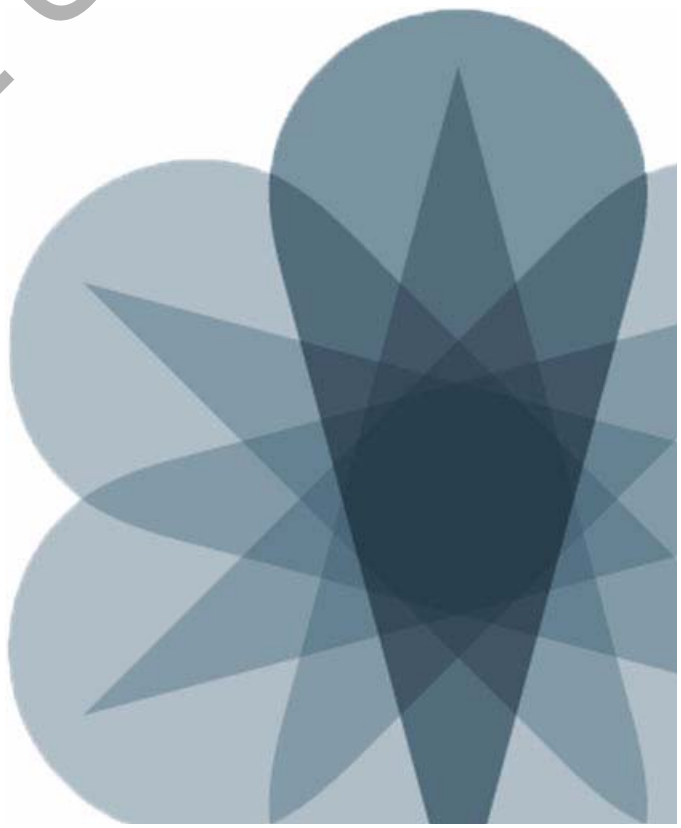
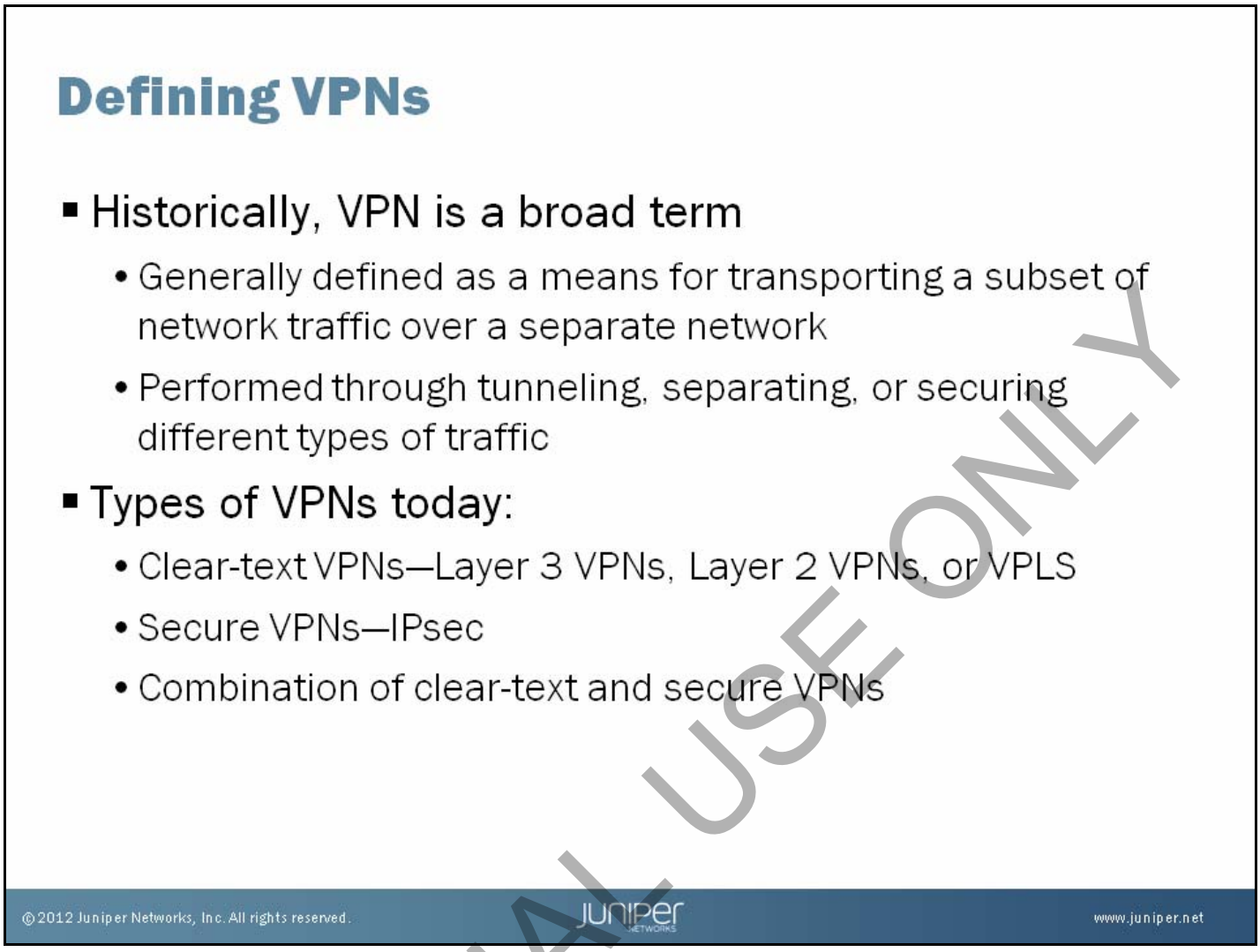
IPsec VPNs





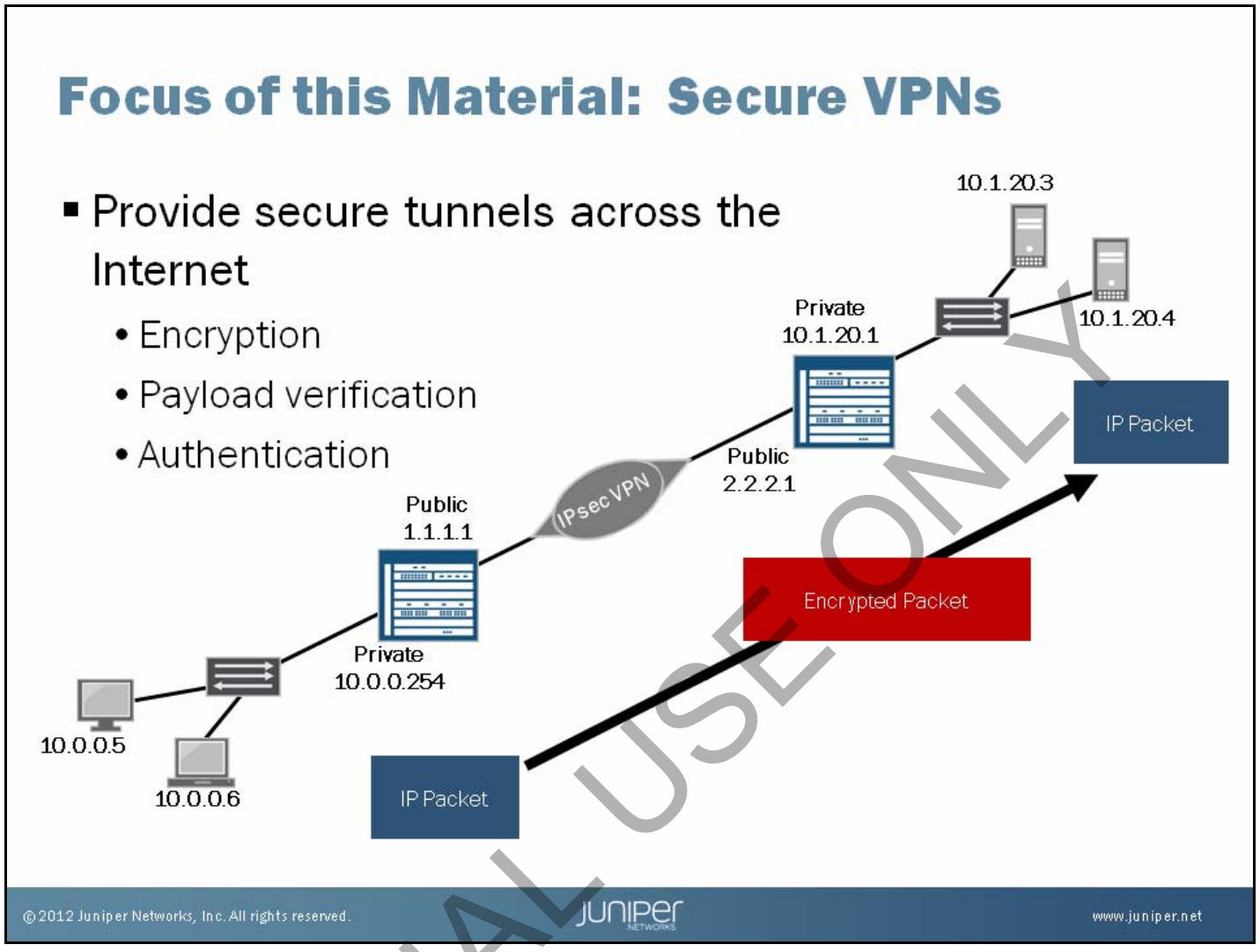
The Meaning Behind VPNs

VPNs are used to transport private network traffic over a public network infrastructure. The term VPN has been used broadly in the networking industry for decades. For instance, the networking industry has referred to X.25, Frame Relay, and ATM infrastructures as VPN networks. As the Internet spread and as carriers and service providers migrated all their service offerings to IP, new forms of VPNs emerged.

Types of VPNs Today

We can subdivide these new forms of VPNs into three categories:

- *Clear-text VPNs*: These VPNs include Layer 3 VPNs, Layer 2 VPNs (Kompella and Martini implementations), and virtual private LAN service (VPLS). These VPNs rely on MPLS services and the use of signaling protocols over IP.
- *Secure VPNs*: These VPNs are IPsec VPNs, carrying payload over IP securely. We will discuss these VPN types in this material.
- *Combination of clear-text and secure VPNs:* These VPNs are based on Layer 3 VPNs, built on MPLS technology, and compounded with IPsec security.

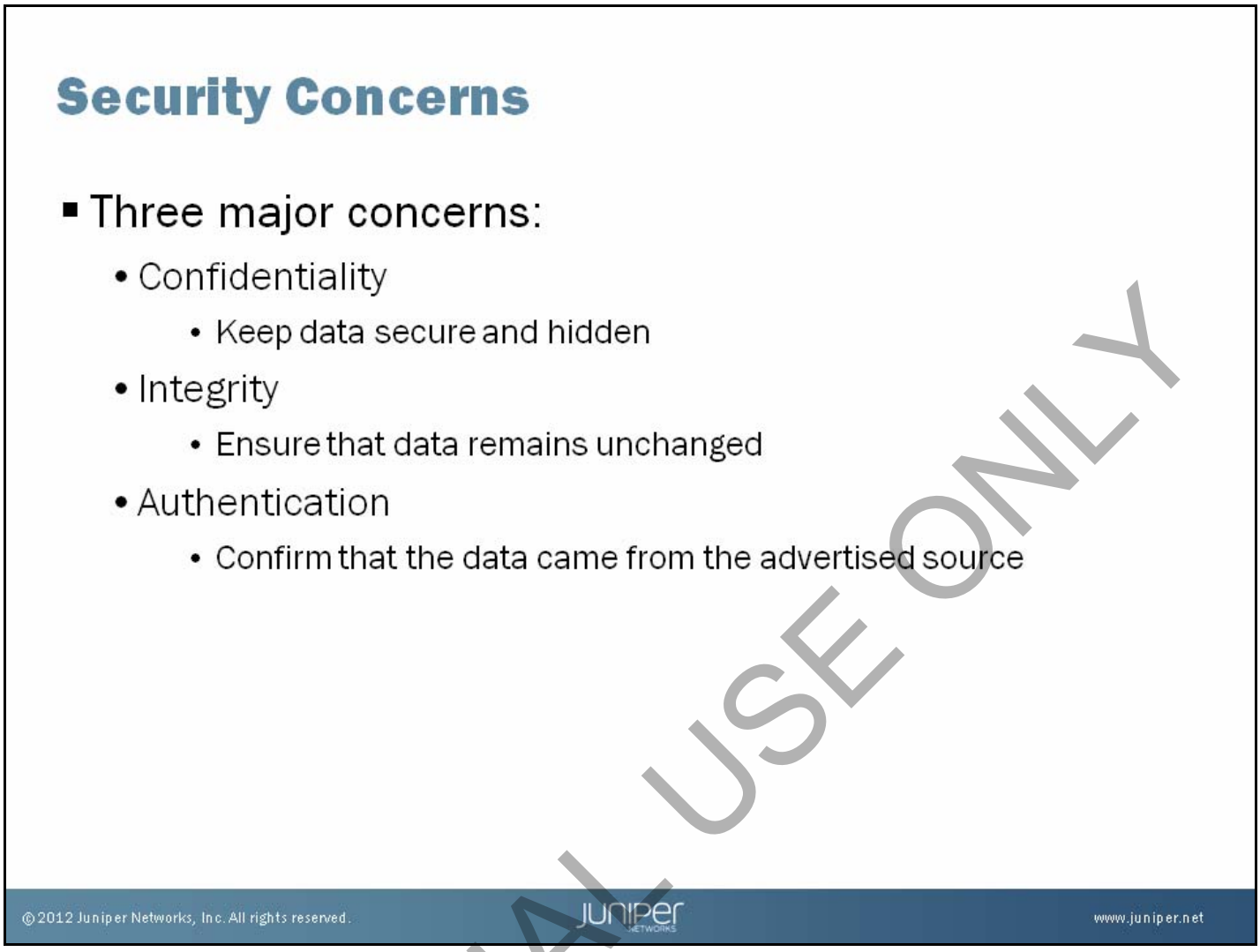


Secure VPNs

A network device that builds secure VPNs must be able to perform the following actions:

- Encrypt the original packet so that it cannot be easily decoded should it be intercepted on the public network;
- Verify the original payload ensuring data integrity; and
- Authenticate the originating device as a member of the VPN, rather than a random device operating on the public network.

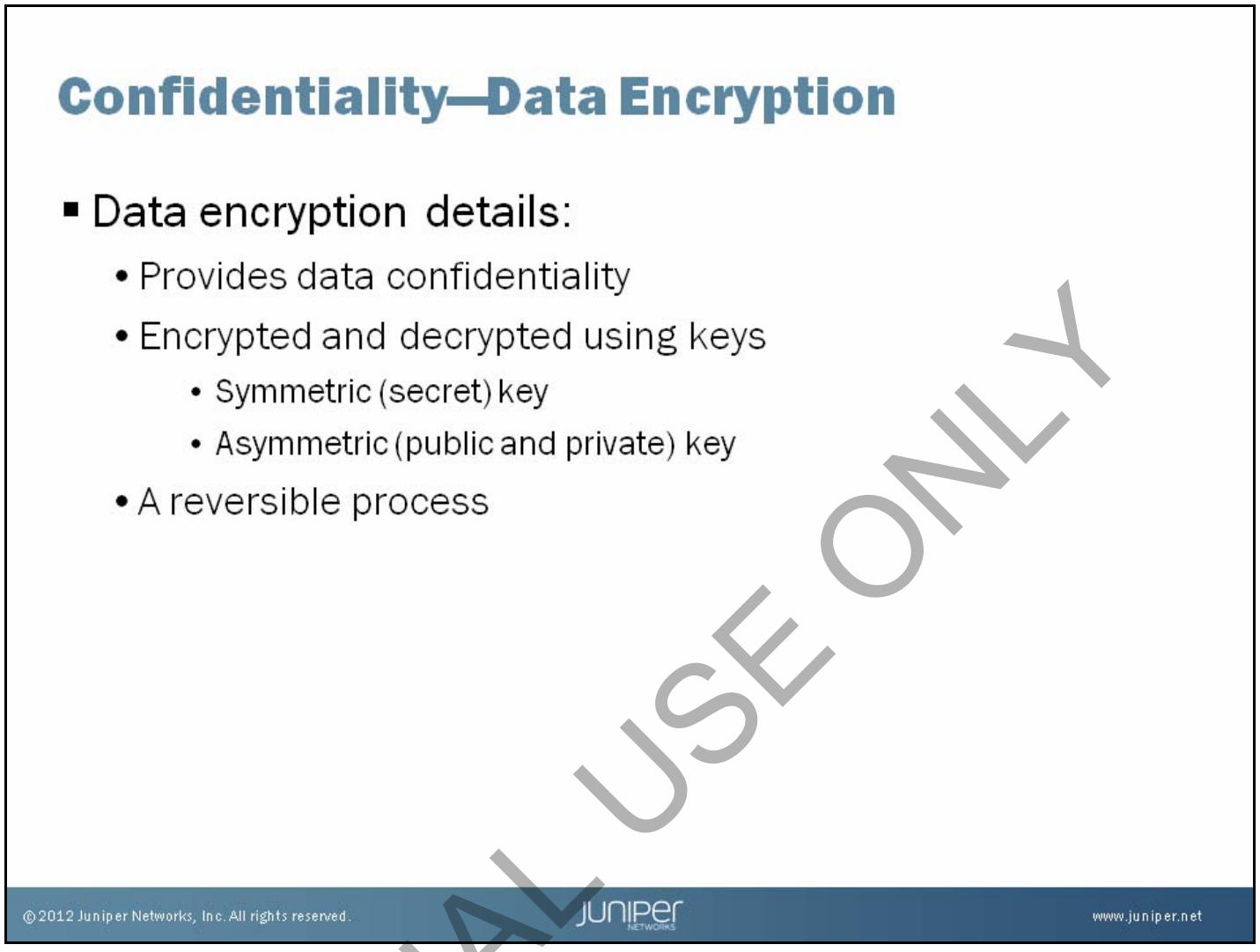
This material focuses on end-to-end static IPsec VPNs. However, note that security platforms running the Junos operating system also support end-to-site dynamic VPNs and VPN establishment using the NetScreen Remote VPN client. See the Juniper Networks technical documentation at http://www.juniper.net/techpubs for further information on these features.



Security Concerns

Three driving concerns exist for network security: confidentiality, integrity, and authentication.

- *Confidentiality*: Online banking, credit card information, or a company's competitive information—how do we keep this information secure from the *man in the middle*? We want information stored in such a way that if someone were to capture this datagram, the information would appear meaningless.
- *Integrity*: Even though the information might be secure and hidden, meaning that someone might not be able to determine or understand its contents, it could still be possible for someone to change it. Someone could tweak bits to change the data from what was originally sent through the network. So how do we ensure that if the data is compromised, the remote station recognizes this fact and refuses to process the information?
- *Authentication*: How does the remote station verify that the information came from the device from which it expected it to come? You do not want to be communicating and sending critical information to the wrong recipient!



Confidentiality—Data Encryption

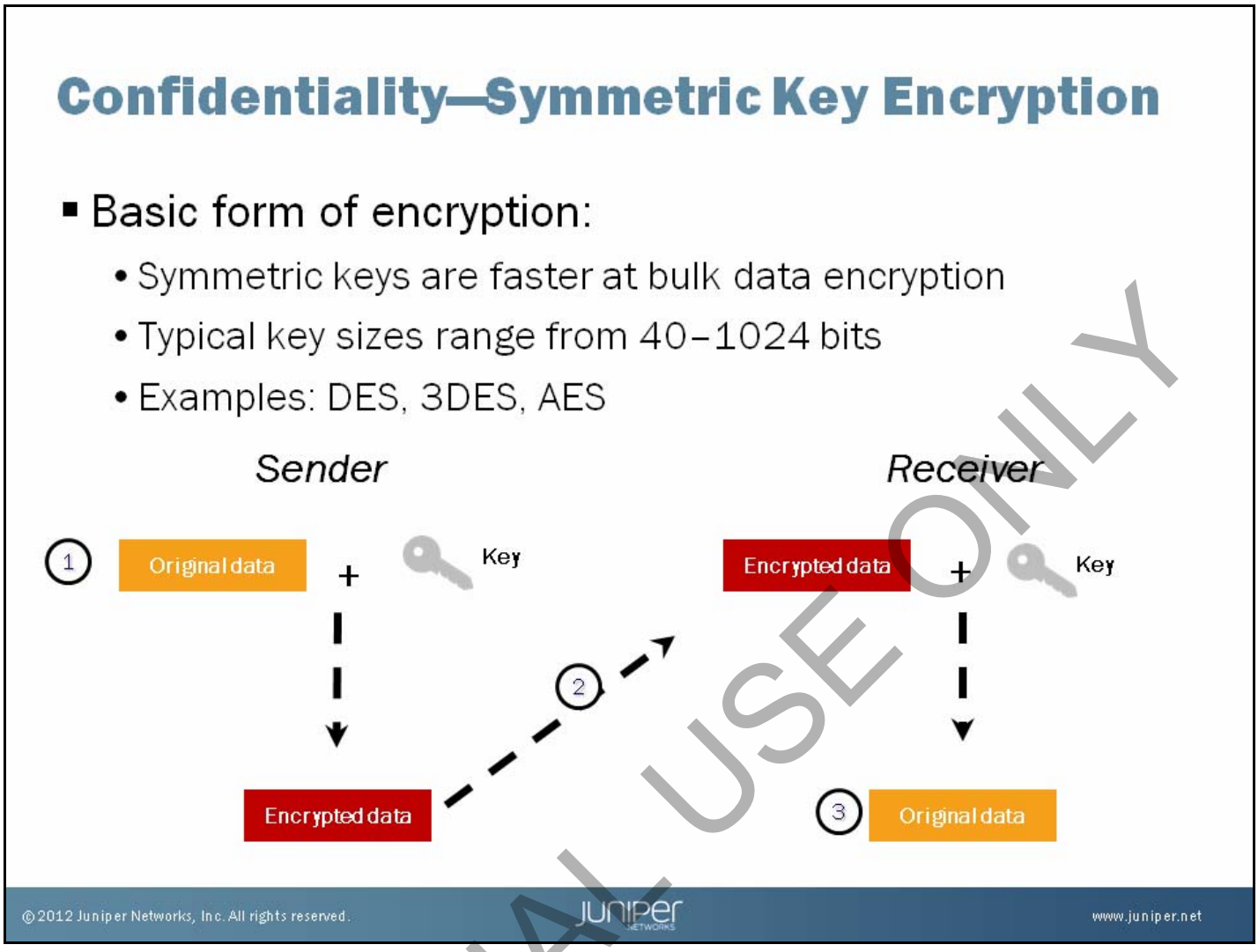
The first of the three VPN security concerns is *confidentiality*.

Encryption provides data confidentiality. Encryption is the method of taking user data—referred to as *plaintext*—and converting it into unreadable or secret data named *ciphertext*. An encryption algorithm and keys (strings of bits that *seed* the encryption process) are applied to the data, resulting in ciphertext.

To reverse the process and decrypt the ciphertext, you must know both the encryption algorithm and encryption key. You can decrypt encrypted data in one of two ways:

- *Symmetric key encryption*: This method uses the same key for both encryption and decryption; and
- *Asymmetric key encryption*: This method uses a private key for encryption and a mathematically related public key for decryption.

The cipher strength depends on the key size; the larger the key, the more secure the cipher output. The trade-off is in processing time—larger keys use more computational cycles to encrypt and decrypt.

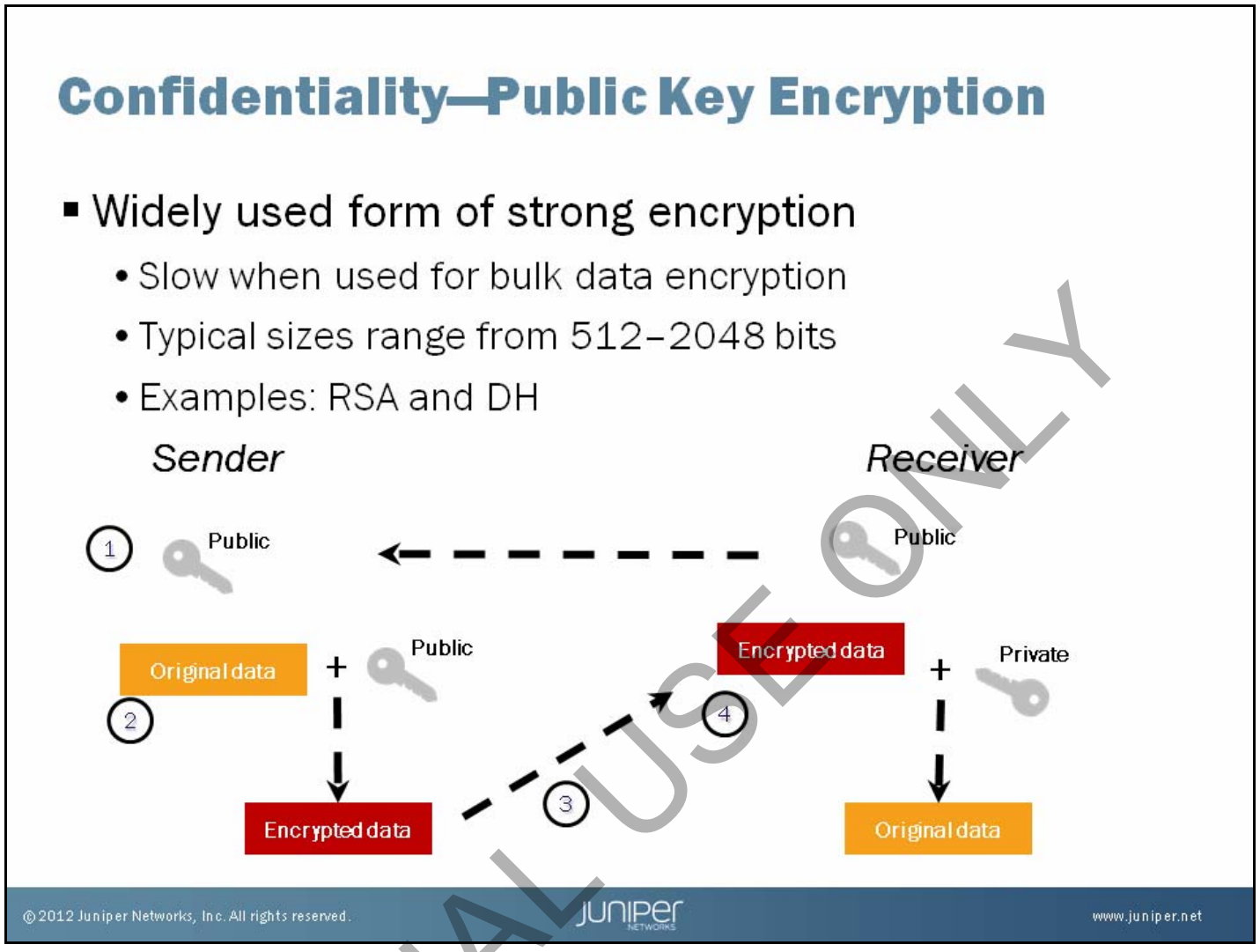


Confidentiality—Symmetric Key Encryption

Symmetric key encryption is the most straightforward form of encryption with the least amount of overhead. We refer to it as *symmetric* because the key used to encrypt the data is the same key used to decrypt the data. Thus, the same key must be known on both sides of a connection.

Symmetric key sizes range from 40 bits–1024 bits. These keys are considered to be very fast because they are not very long, and they are widely used for bulk data encryption. However, because the key must be known to both the sender and the receiver, key management is a problem when using symmetric keys.

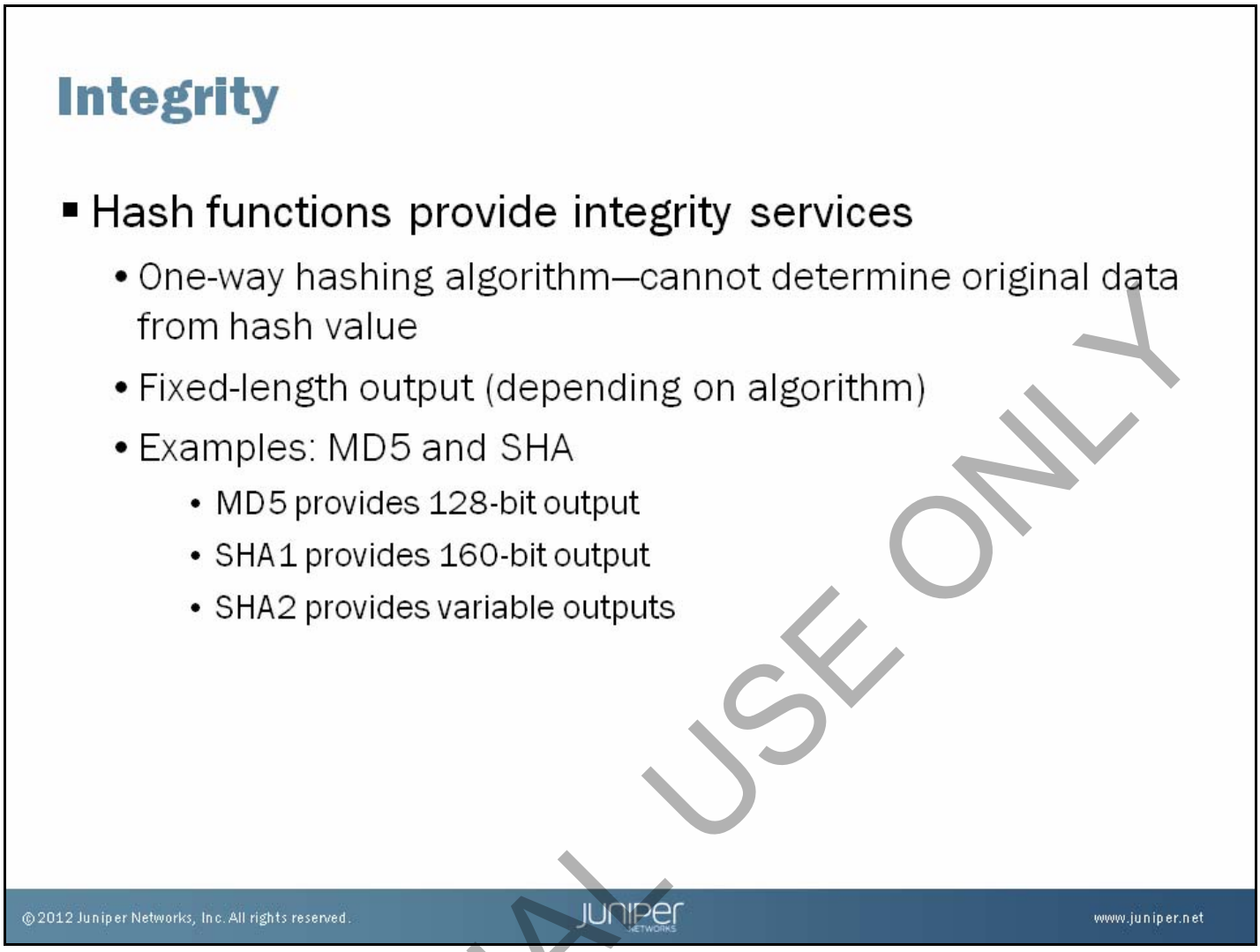
Examples of symmetric key encryption include Rivest Cipher 4 (RC4), Data Encryption Standard (DES), Advanced Encryption Standard (AES), and Blowfish.



Public Key Encryption

The public, asymmetric key encryption method requires a pair of mathematically related keys. One of the keys is kept secret and known only to the owner; this key is the private key. The owner distributes the other key widely and anyone can access it; this key is the public key. You can only decrypt data encrypted by the private key by using the corresponding public key, and vice versa. The keys are mathematically related such that it is almost impossible to derive one key out of another.

Public key sizes range from 512 to 2048 bits. Because of the large size, these keys are extremely slow and generally not feasible for bulk data encryption. However, public keys are widely used for user and device authentication (for example, digital certificates). An example of public, asymmetric key encryption is RSA.

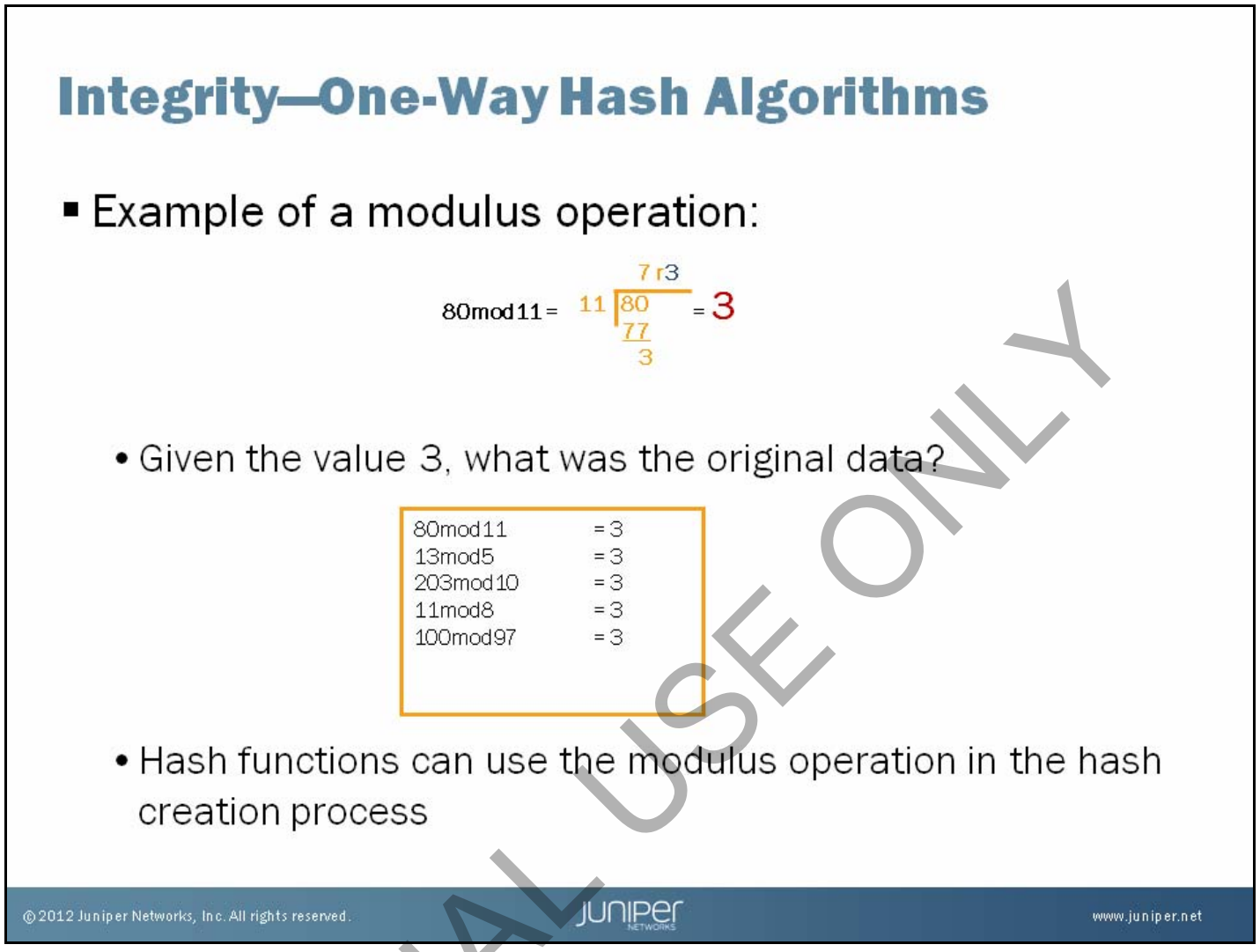


Integrity

Now that we have the data encrypted as it traverses the Internet, we must ensure that the data is not subject to modification along the way. Even though a novice hacker might not be able to crack the encryption algorithm and key, the hacker can still wreak havoc by modifying bits that the encrypted payload carries. If this modification happens, the decrypted output does not match the original data. Who knows what the consequences might be!

Hashing solves this problem by creating a *fingerprint* of the data, similar to a cyclic redundancy check (CRC) checksum. Before data travels, it traverses a hashing engine that produces a fixed-length hash output. It places the hash in a field in the packet along with the data before it travels over the network. The destination device takes the same data and runs it through the same hashing algorithm, calculating its own hash. The destination device then compares the hash that it calculated against the hash carried in the packet. The same hash in both locations proves data integrity in transit. If the hashes do not match, the packet drops.

The Junos OS supports MD5, SHA1, and 256-bit SHA2 hashing.



One-Way Hash Algorithms

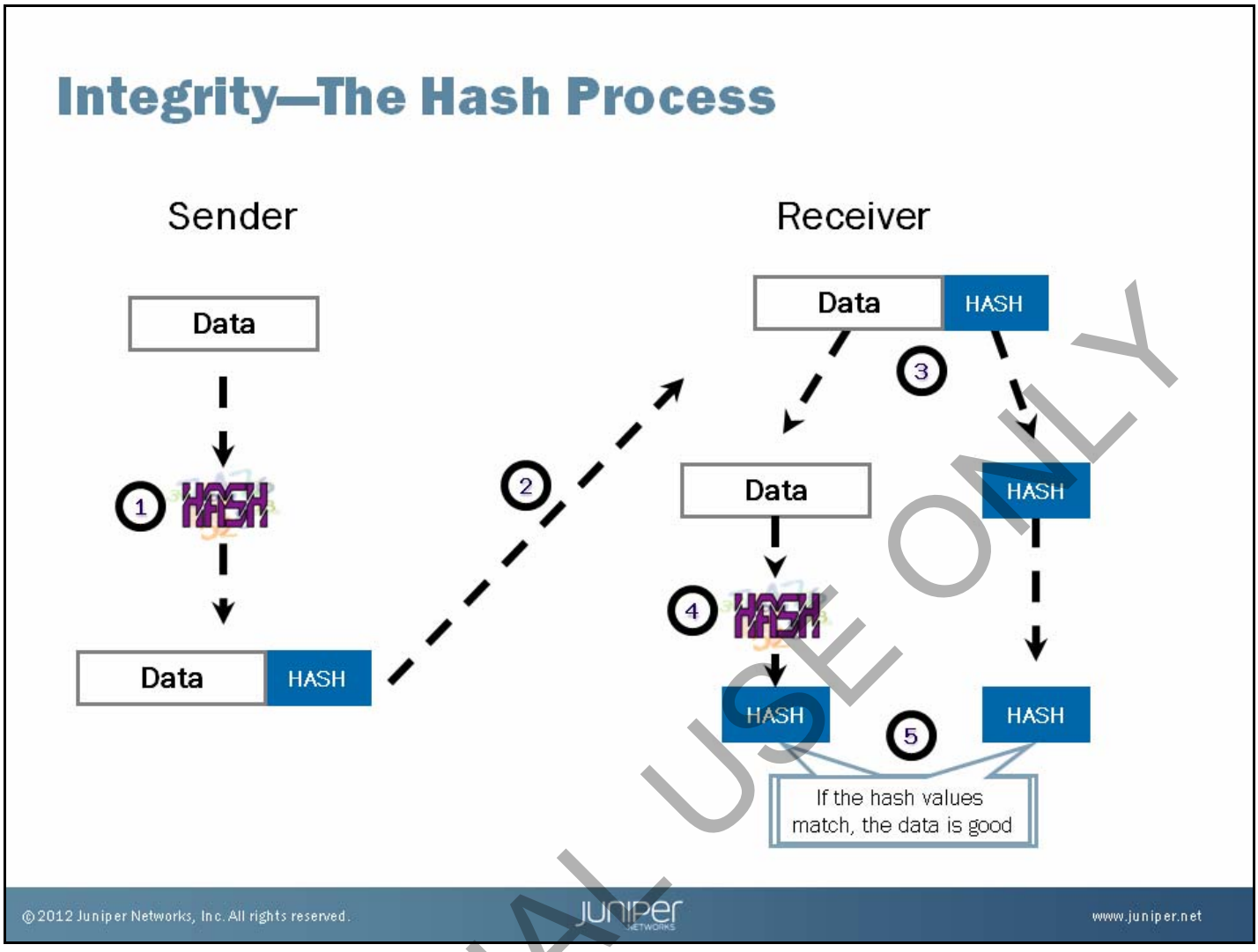
A hash function must have two basics properties:

- You should not be able to calculate the original data from the hashed output. This property ensures that you cannot derive the plaintext from the ciphertext.
- It must be mostly collision resistant. A collision occurs when two different inputs give the same output. It must not be possible to predict a different input value that gives the same output. This property is necessary because the purpose of hashing is to verify that the data has not changed.

To see how it is possible to create a one-way function, think of the example on the slide, which shows a modulus operation. Given the value of 3, it is not possible to determine the original value because an infinite range of possible answers exists.

However, this example is not suitable as a real-world hash function because it does not satisfy the collision-resistant requirement—a malicious person can change the plaintext by any multiple of the modulus number and know that the hashed value remains the same.

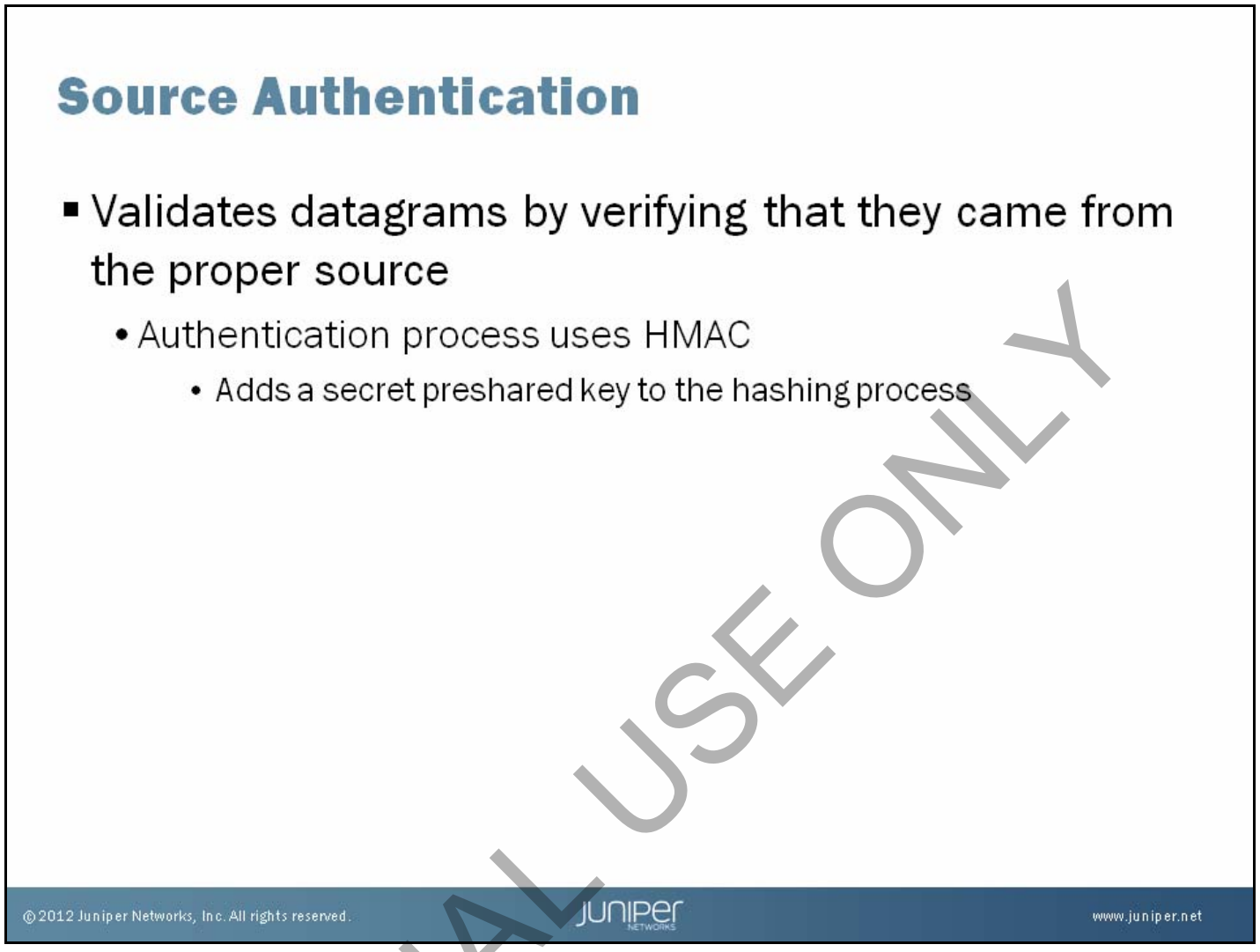
The most secure and widely used hash function is the Secure Hash Algorithm 1 (SHA-1). SHA-1 is preferred over Message Digest 5 (MD5), although MD5 is still widely supported. These functions produce a fixed-length output that is useful when working with IP packets because the overhead of transmitting the hash value is predictable.



Integrity—The Hash Process

The following list outlines the hash process:

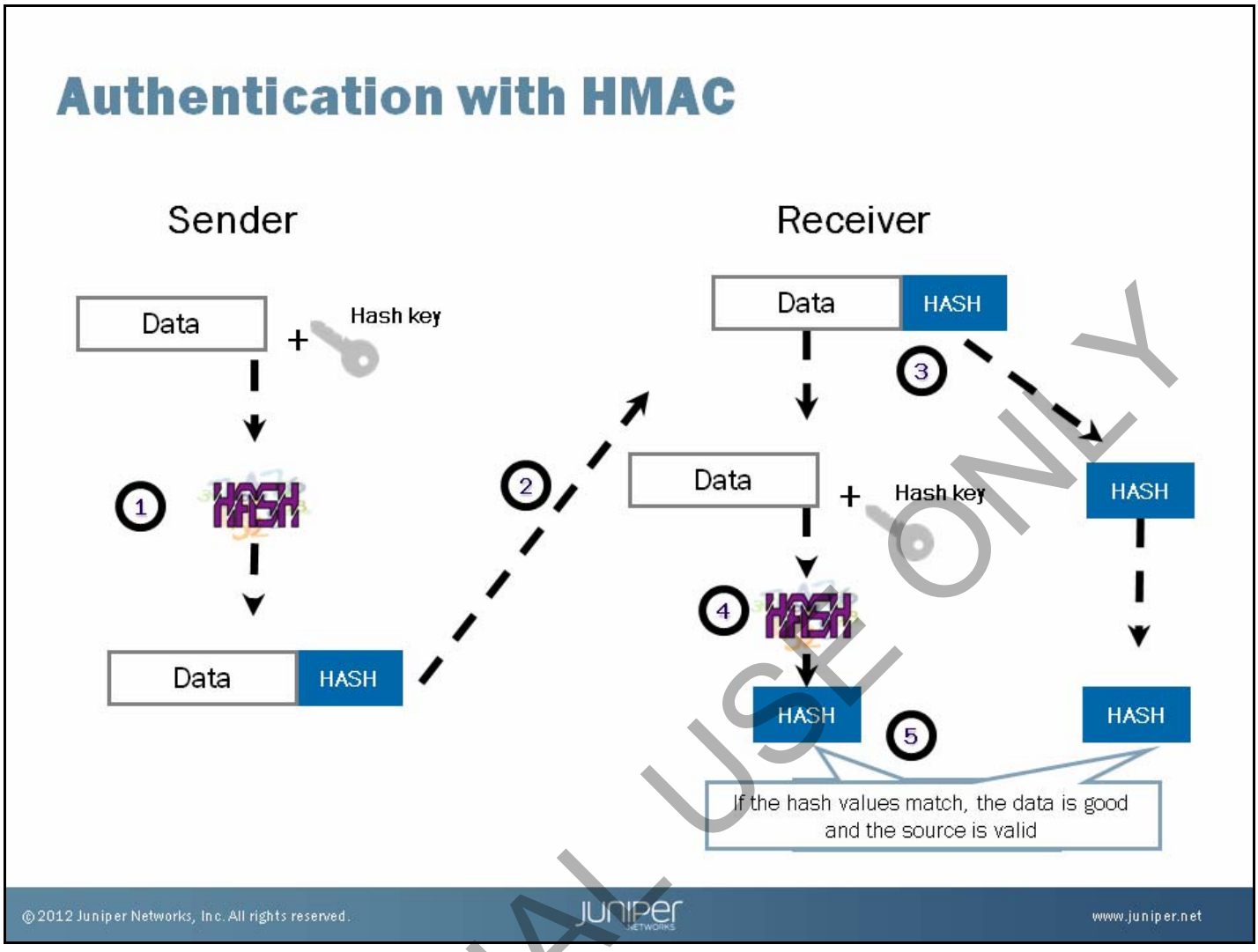
1. The sender runs the data through the hash process.
2. The sender appends the hash value to the data and sends both the data and the hash value to the receiver.
3. The receiver separates the data and the hash value.
4. The receiver hashes the data.
5. The receiver then compares the calculated hash value to the received hash value. If the hash values match, the data is unaltered.



Source Authentication

Encryption protects the packet contents from being viewed on the public network. Hashing verifies that the data is unaltered. But how do you validate the source of the data?

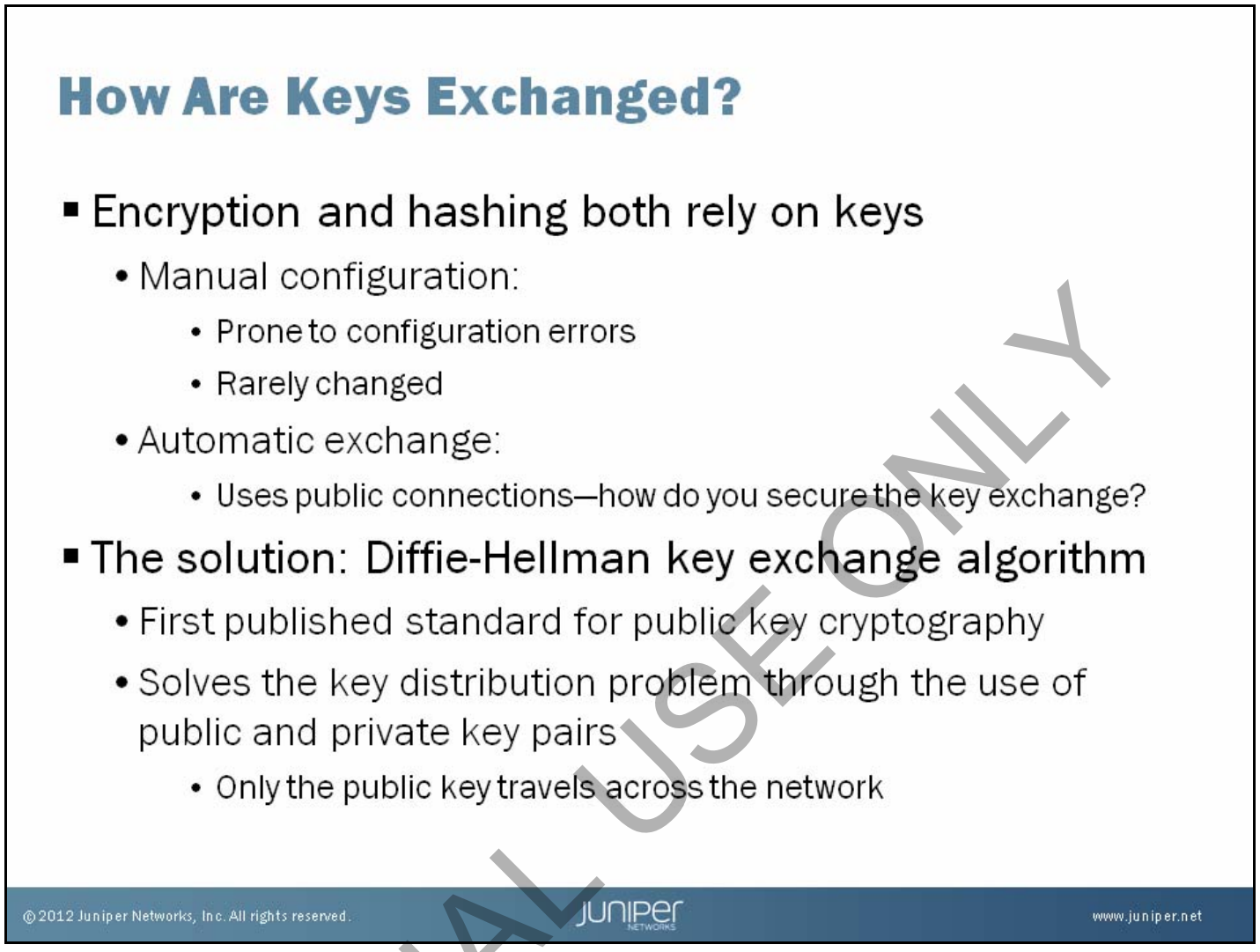
The software performs source authentication using the Hashed Message Authentication Code (HMAC). The sender appends a secret preshared key to the data, then performs the hash function. For hashes to successfully match, the receiver must append the same key value to the data before performing the hash function. The key itself never transmits along with the data.



HMAC Authentication

The following list outlines hashing with HMAC:

1. The sender appends the preshared key to the data, then performs the hash function.
2. The data and hash value travel to the receiver.
3. The receiver separates the data and the hash value.
4. The receiver appends the preshared key to the data, then performs the hash function.
5. The receiver then compares the calculated hash value to the received hash value. If the hash values match, the data is unaltered. If the hash values do not match, either the data itself is corrupt, or the keys did not match, meaning the source is invalid. Either way, the receiver discards the packet.



Key Exchange

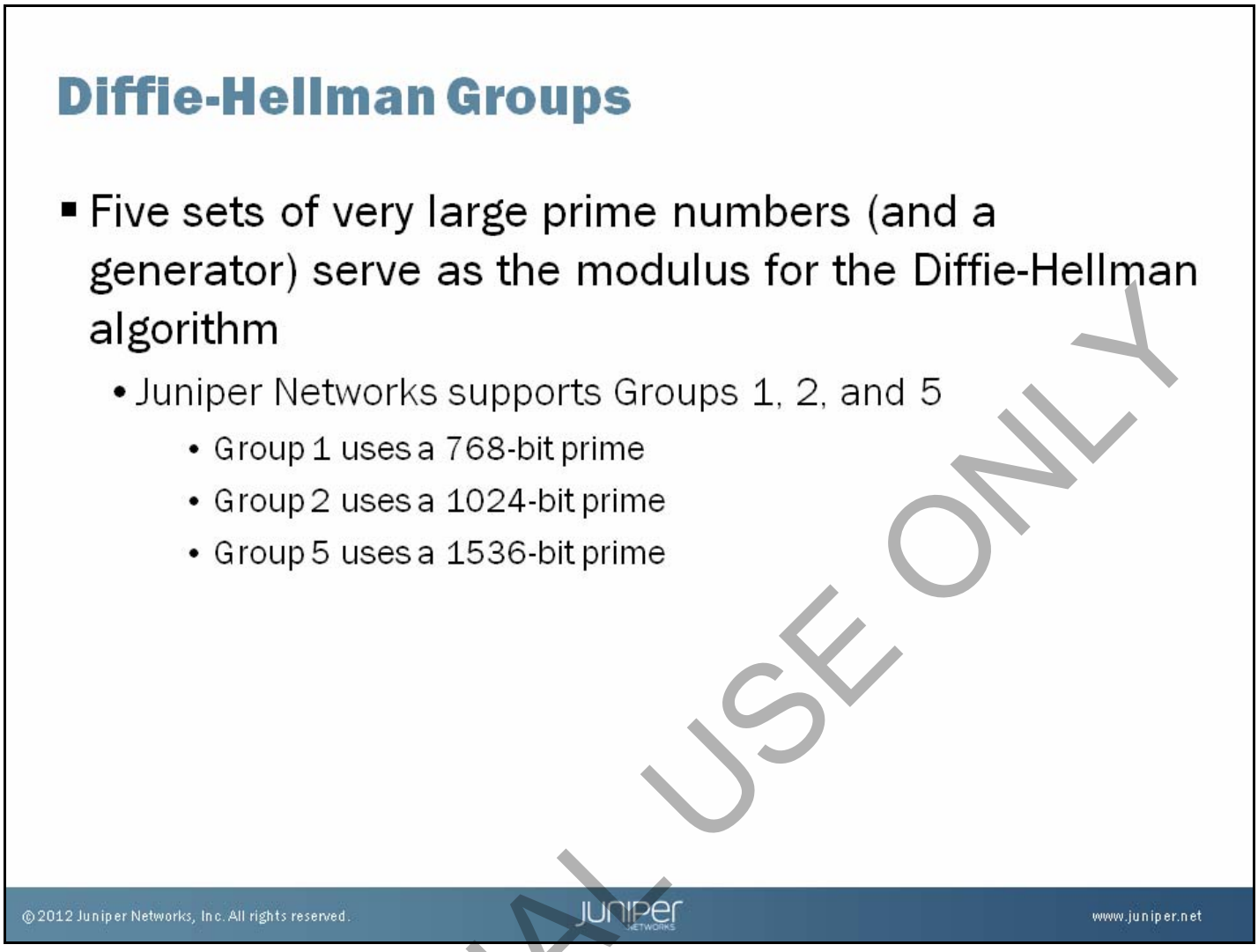
As we already discussed, both encryption and authentication are dependent on security keys, which leads to the problem of key exchange. If keys must be the same on both sides of a connection, how can you securely exchange key information?

One option is to manually configure the keys on both sides of the connection. Manual key configuration is straightforward, but misconfigurations, especially when each device has a different administrator, are common. Furthermore, manual configuration usually means that keys rarely change, which is itself a potential security issue; given a large enough sample, any code can be broken.

Automating the key exchange process is a good idea, but we must overcome the problem of sending keys across a public network. Anyone intercepting the key has the ability to decode the data.

The Solution

Whitfield Diffie and Martin Hellman developed a solution to this problem in 1970. The Diffie-Hellman algorithm is a method whereby two parties can agree upon a secret key known only to them. The strength of the technique is that it allows the participants to create the secret value over an unsecured medium without exchanging the secret value itself. This method also makes it impossible to perform reverse generation of the secret if it is somehow intercepted.

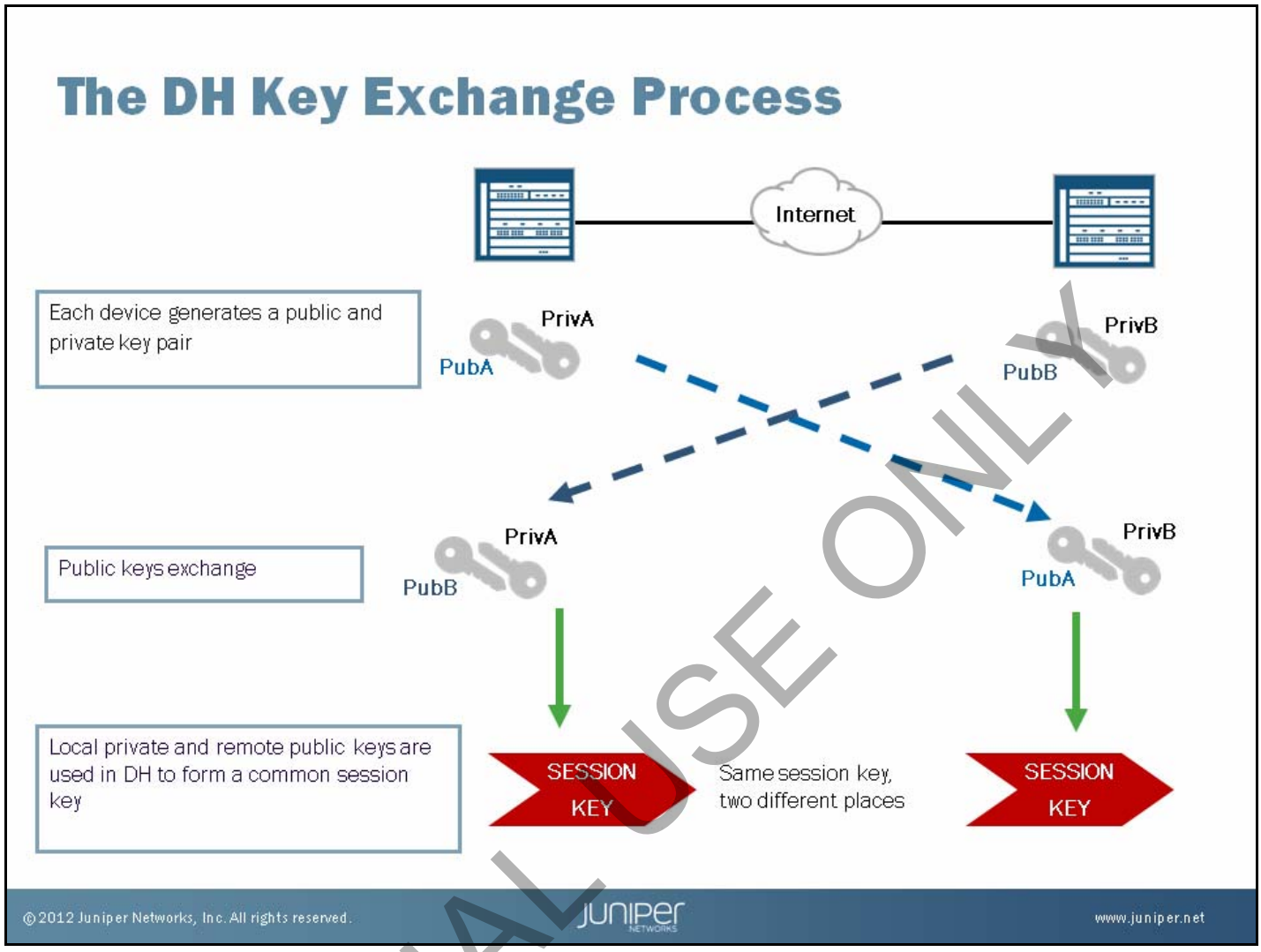


Diffie-Hellman Groups

Diffie and Hellman proposed five groups of prime numbers and generator values to use in their key exchange algorithm. Each group generates unique keys using a combination of exponential and modulus calculations.

The Junos OS supports Diffie-Hellman (DH) Groups 1, 2, and 5. The larger the prime number—the stronger the key—and the more computationally intensive the calculation. Diffie-Hellman Group 1 uses a 768-bit prime number. Diffie-Hellman Group 2 uses a 1024-bit prime number. Diffie-Hellman Group 5 uses a 1536-bit prime number.

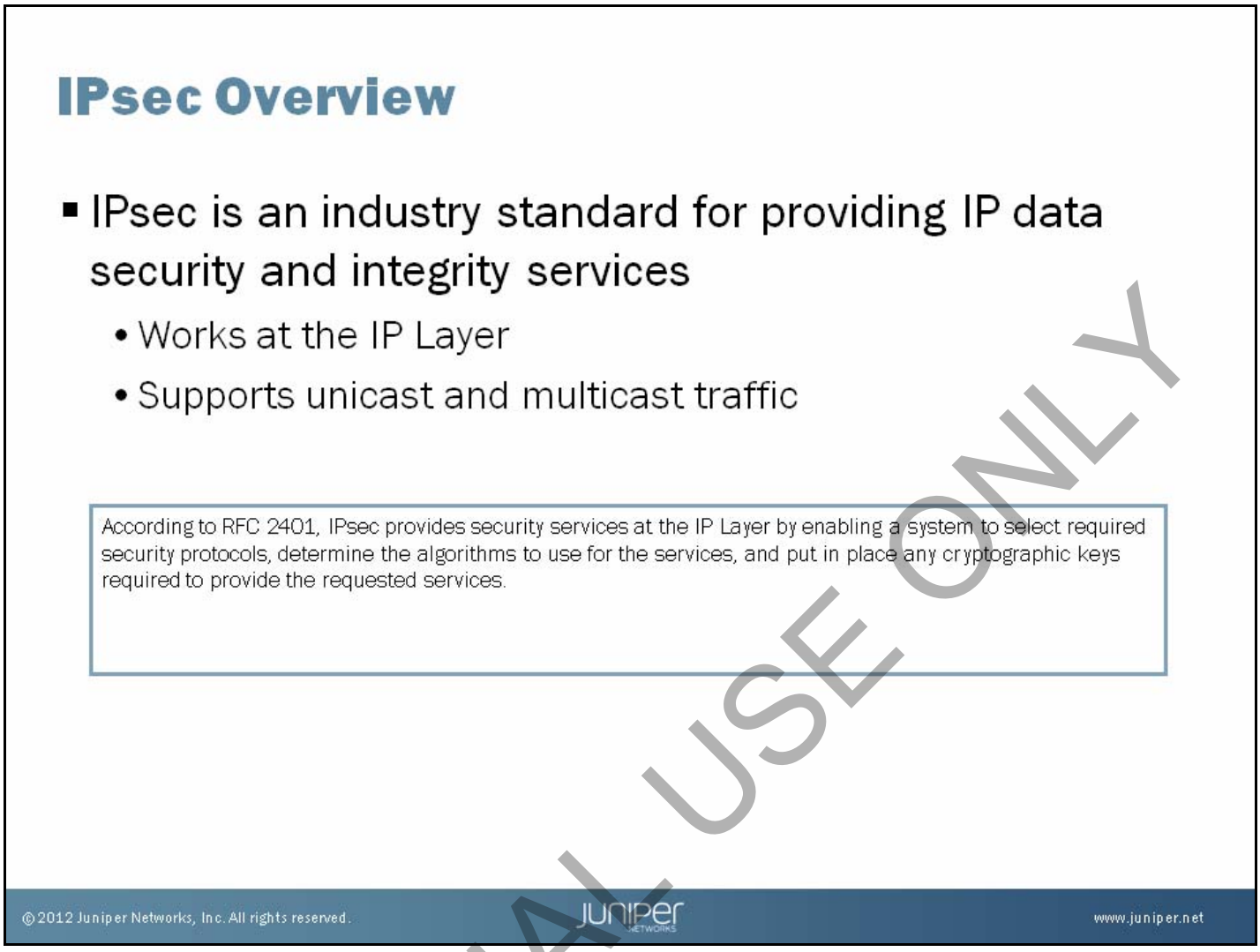
You must configure both tunnel peers to use the same DH group; otherwise, the key generation process fails.



The DH Key Exchange Process

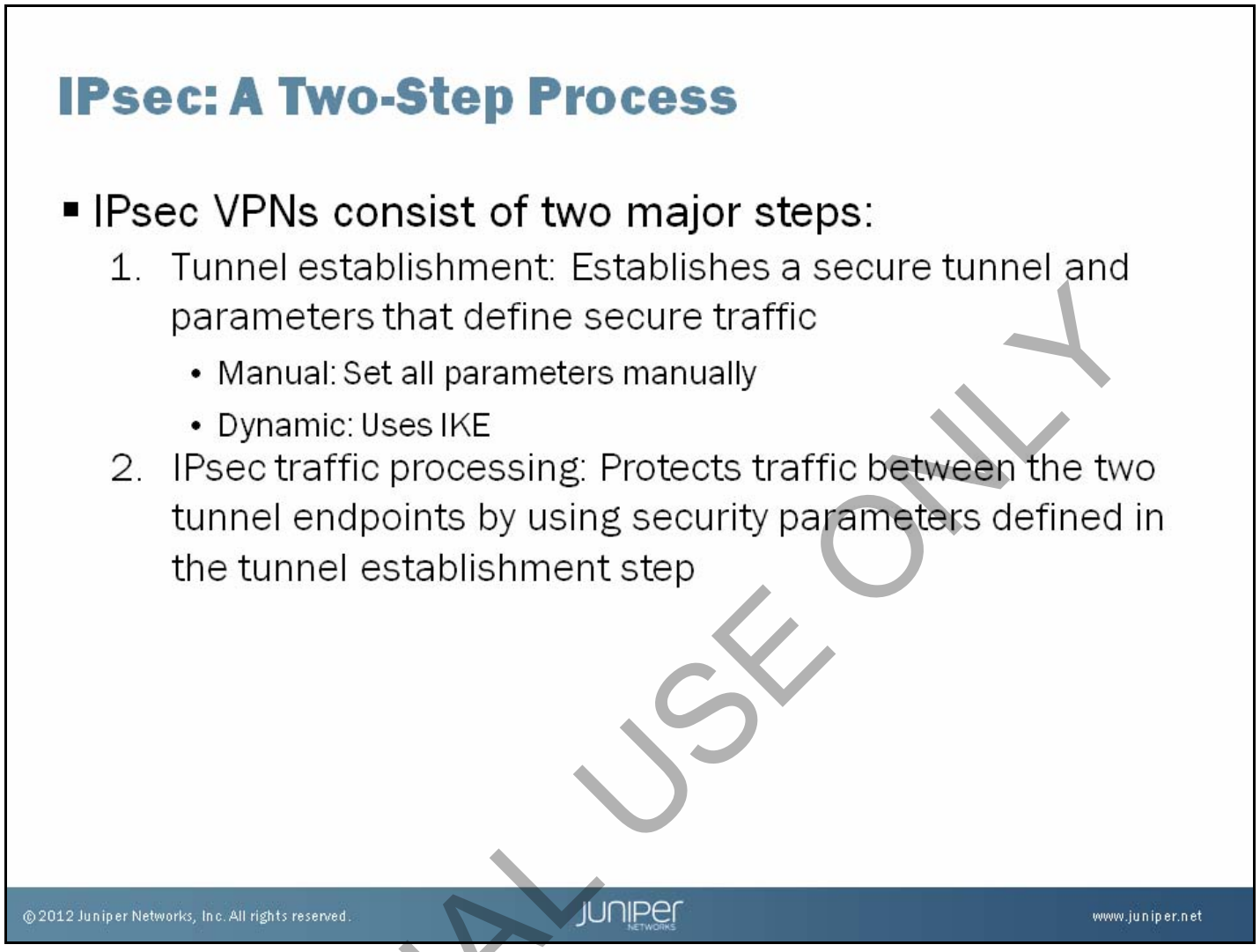
Using the same DH group, each Junos security platform creates unique public and private keys. These keys are mathematically related by means of the DH algorithm.

The public key values exchange across the network. Each peer then runs its local private key and the received public key value through the DH algorithm to compute a common session key. The session key itself never passes across the network.



IPsec Overview

IPsec is a set of standards that defines how the encryption, validation, and authentication methods we just discussed are actually implemented in networks. IPsec works at Layer 3; it supports both unicast and multicast traffic.

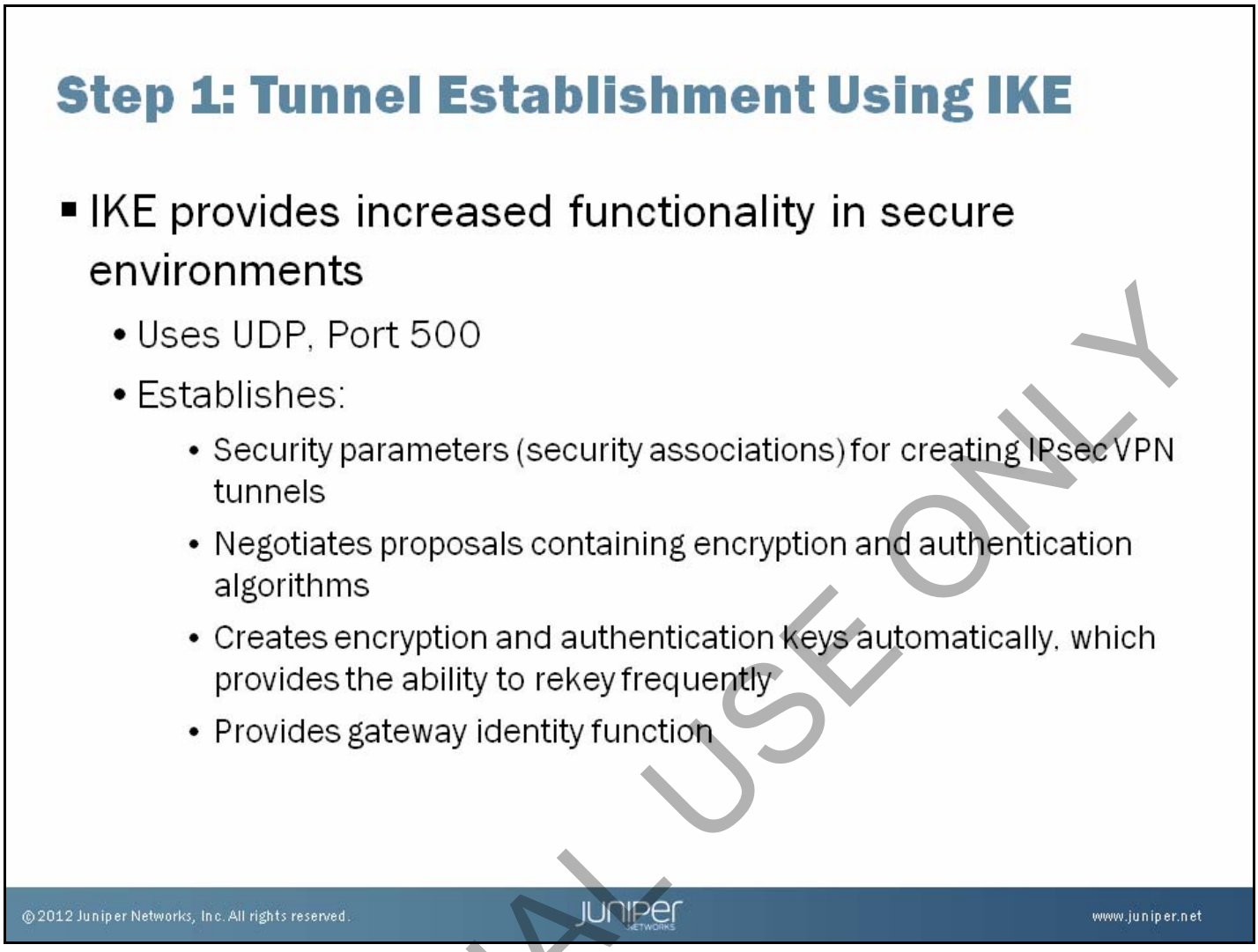


IPsec: A Two-Step Process

IPsec VPNs consists of two major steps:

1. *IPsec tunnel establishment*: You can manually establish an IPsec tunnel or the Internet Key Exchange (IKE) protocol can do it dynamically.
2. *IP traffic processing*: During this step payload protection takes place using security parameters defined in the tunnel establishment phase.

We cover the first step—IPsec tunnel establishment using IKE—on the next several pages.



Step 1: Tunnel Establishment Using Internet Key Exchange

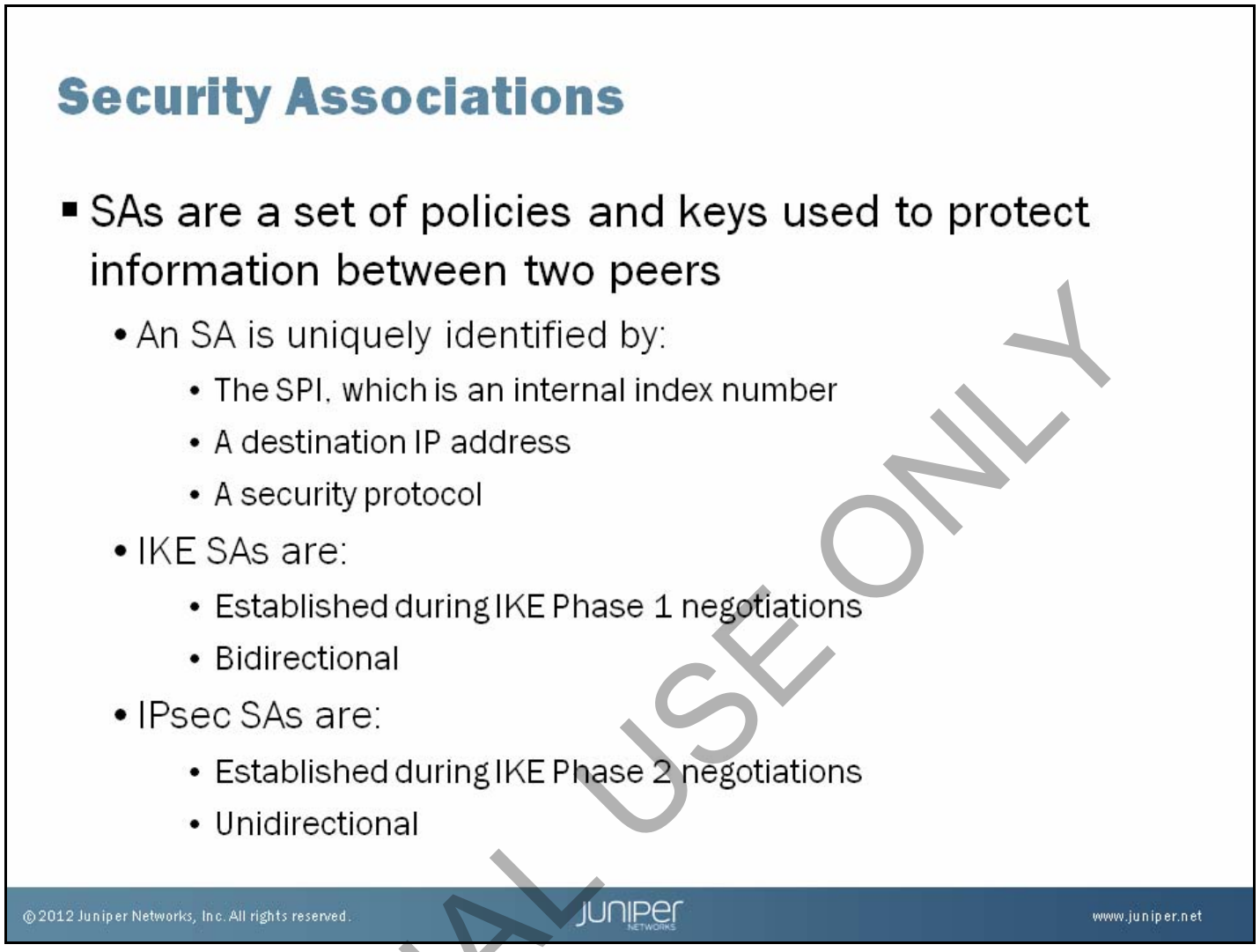
IKE is a secure key management protocol used by IPsec to have information exchanged in a secure and dynamic manner with little or no intervention. The IKE proposal exchange is Phase 1 of the IPsec tunnel establishment process. The following attributes exchange between IPsec peers as a part of the IKE process:

- Encryption algorithm;
- Hash algorithm;
- Authentication method; and
- Diffie-Hellman group.

Once IPsec peers negotiate these attributes, they secure future attribute exchanges used to protect data. IKE exchanges authenticate using one of the following methods:

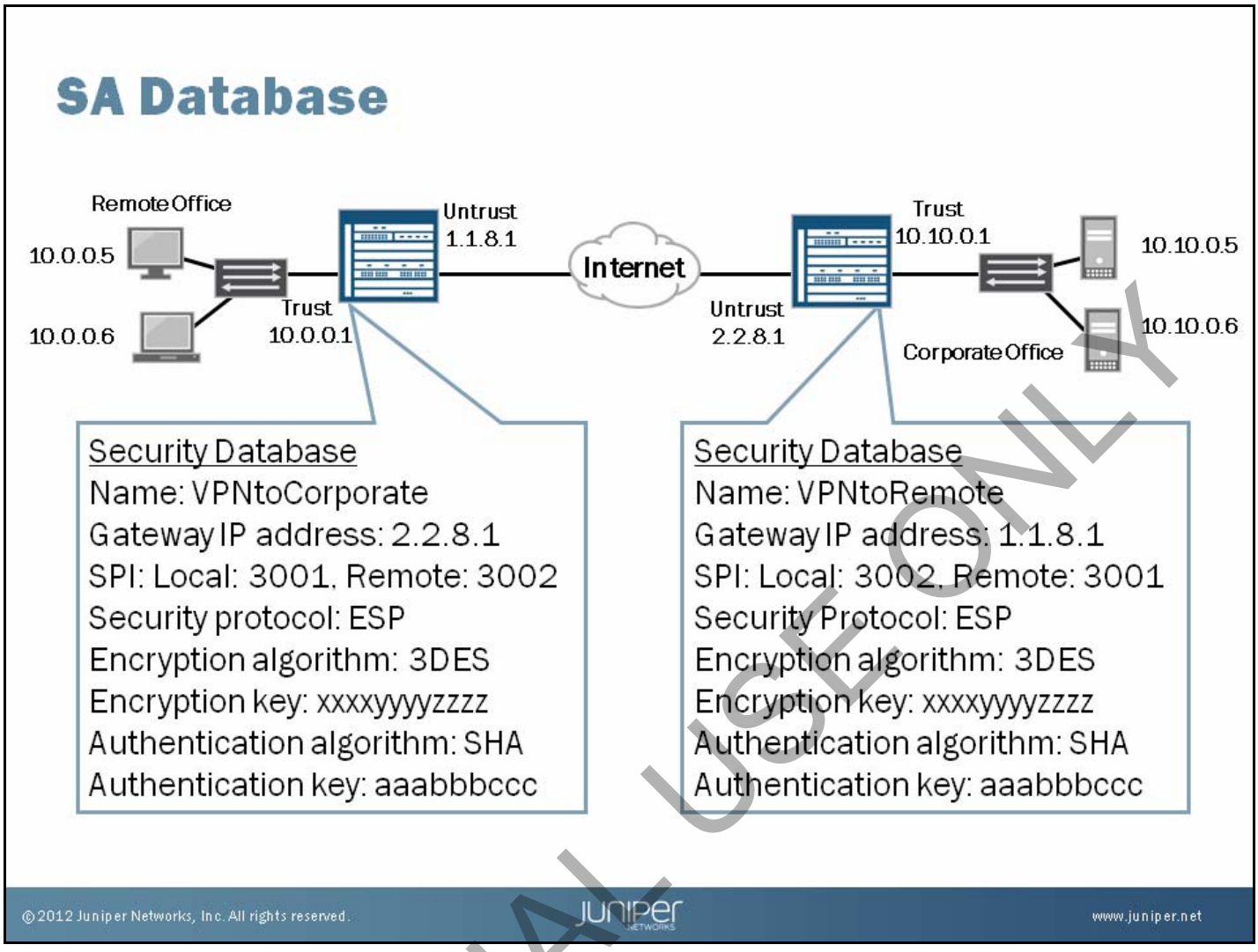
- Preshared keys;
- Digital signatures; or
- Public key encryption.

IKE is preferable to manual keys in IPsec implementations because of the ease of management and scalability.



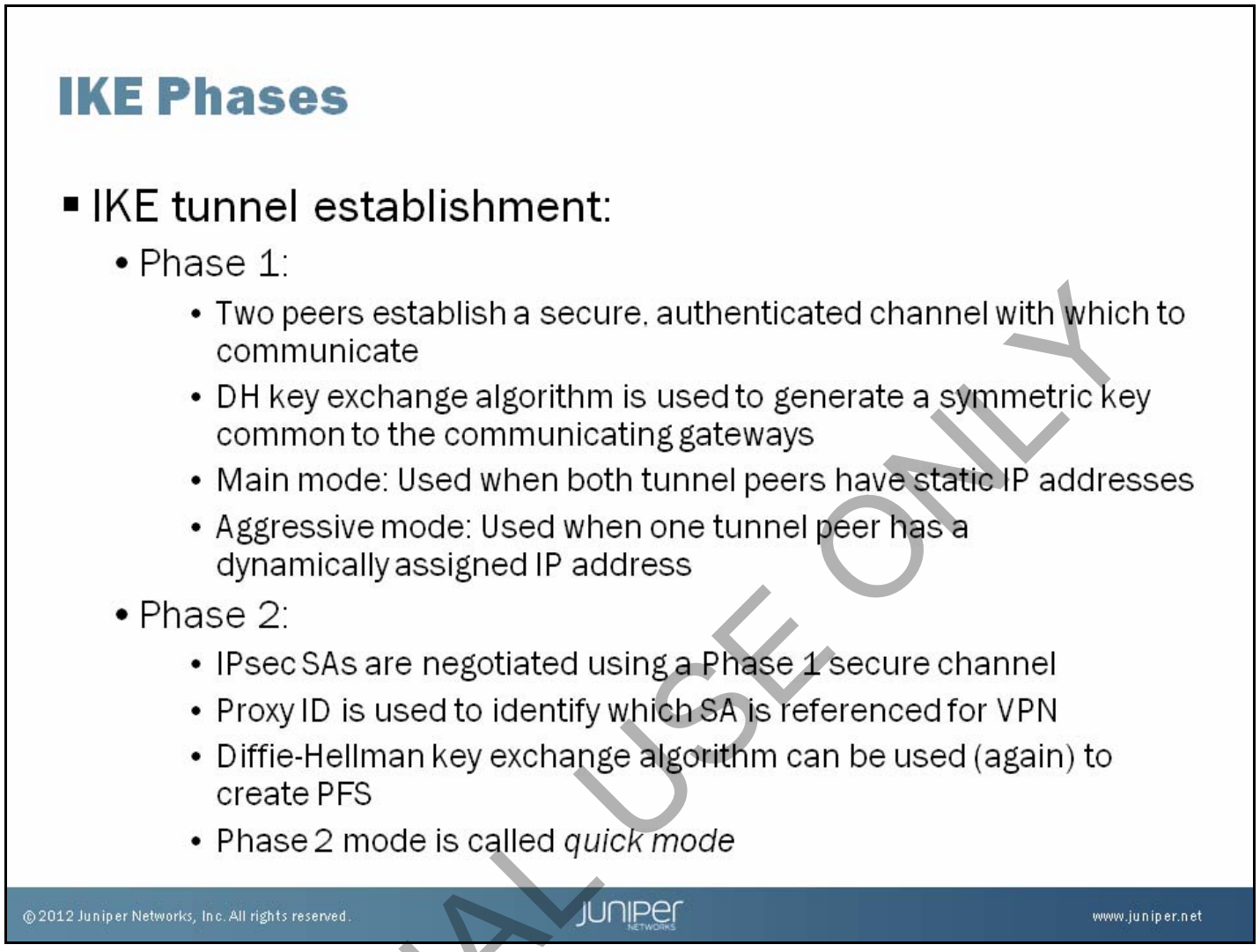
Security Associations

A security association (SA) is a set of policies and keys used to protect information. SAs establish upon the successful completion of IKE negotiations. An SA is uniquely identified by the security parameter index (SPI) value, the tunnel destination address, and the security protocol—Encapsulating Security Payload (ESP) or authentication header (AH)—in use. The lifetime of an SA can be based on either a time value or a value determined by the volume of traffic protected by the proposal.



SA Database

SAs are stored in a security association database. Each entry includes the name of the particular VPN, the remote gateway IP address, the SPIs for each direction, the agreed-upon security protocol, encryption, and authentication algorithms and keys.

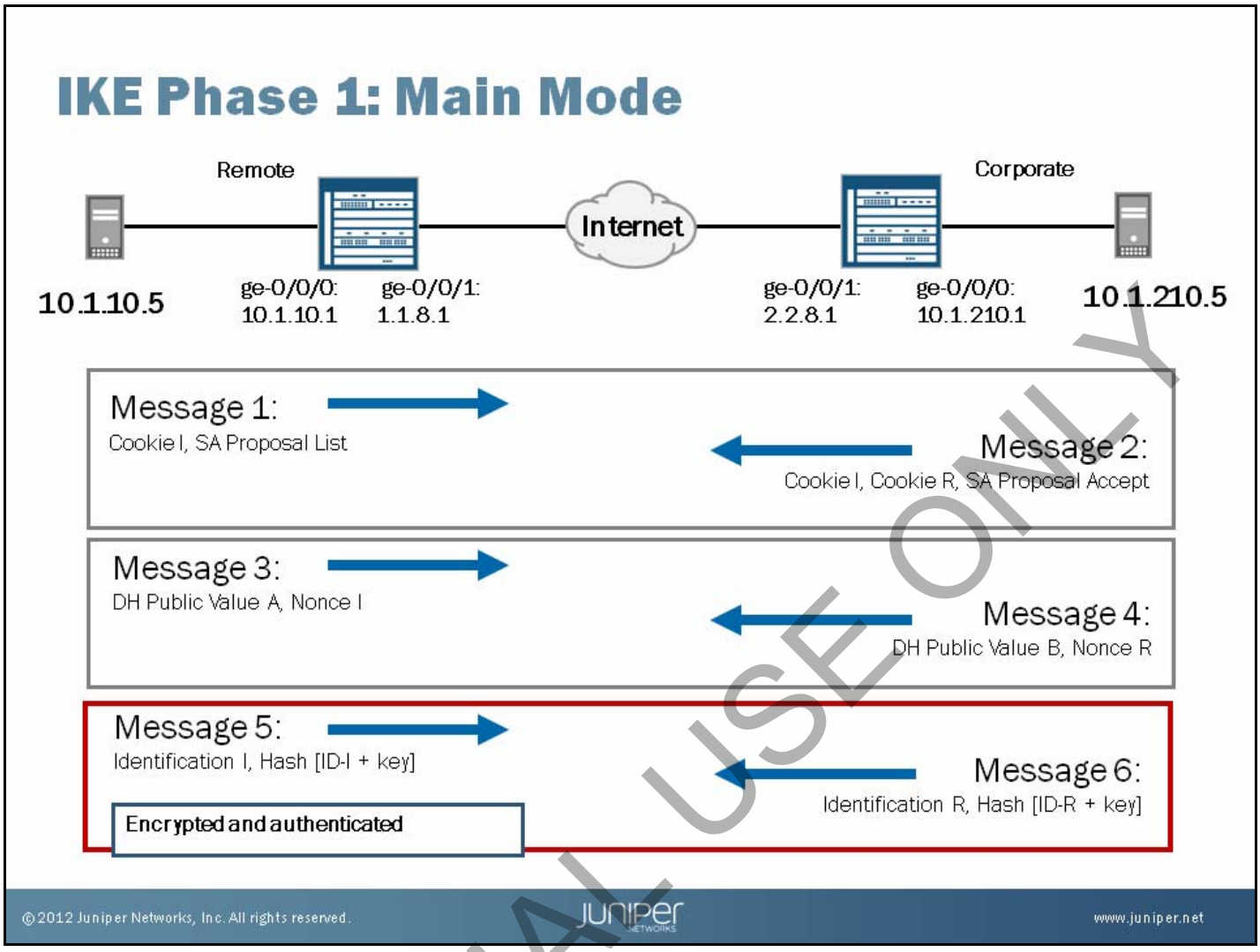


IKE Phases

IKE tunnel establishment happens in two phases:

- Phase 1 establishes a secured channel between gateways for Phase 2 negotiations to occur. The Diffie-Hellman key exchange algorithm establishes a shared key for encryption.
- Phase 2 establishes the specific VPN connections. SAs are negotiated on behalf of IPsec to determine the encryption and authentication algorithms to use when sending user data. The SA is identified by a unique SPI that is also negotiated during Phase 2.

A single Phase 1 channel can establish multiple Phase 2 SAs or VPNs. If wanted, a second Diffie-Hellman exchange can be performed during Phase 2 to negotiate a new tunnel key. Because of the encryption on this exchange, it is named *Perfect Forward Secrecy* (PFS).



IKE Phase 1: Main Mode

IKE main mode is used when both tunnel peers have static IP addresses. The Phase 1 exchange determines the following attributes:

- Encryption algorithm;
- Hash algorithm;
- DH group; and
- Authentication method:
  - Preshared keys;
  - Digital signatures; and
  - Public key encryption.

The first two messages validate the peer configuration (by checking the cookie against the locally configured peer IP address) and negotiate the parameters listed previously. Both tunnel peers must have at least one configured matching proposal for Phase 1 exchange success.

The next two messages exchange Diffie-Hellman public key values and nonces necessary to compute the shared key.

The last two messages send simple identification information using the negotiated key; these messages validate that the key calculation was correct.

Continued on next page.

IKE Phase 1: Main Mode (contd.)

For Message 1 and Message 2, peers exchange cookies and SA proposals. Cookies are 8-byte pseudo-random numbers generated by the sending machine (I=initiator) and receiving machine (R=receptor). Every cookie is unique to the machine and to each particular exchange. These cookies guarantee uniqueness and replay protection by hashing the sender's IP address, port, protocol, and timestamp, which results in a unique identifier known only to the originator. Hence, they are included in every IPsec packet and used to identify communication. In turn, the receptor inserts its known cookie in Message 2 if it accepts the SA proposal.

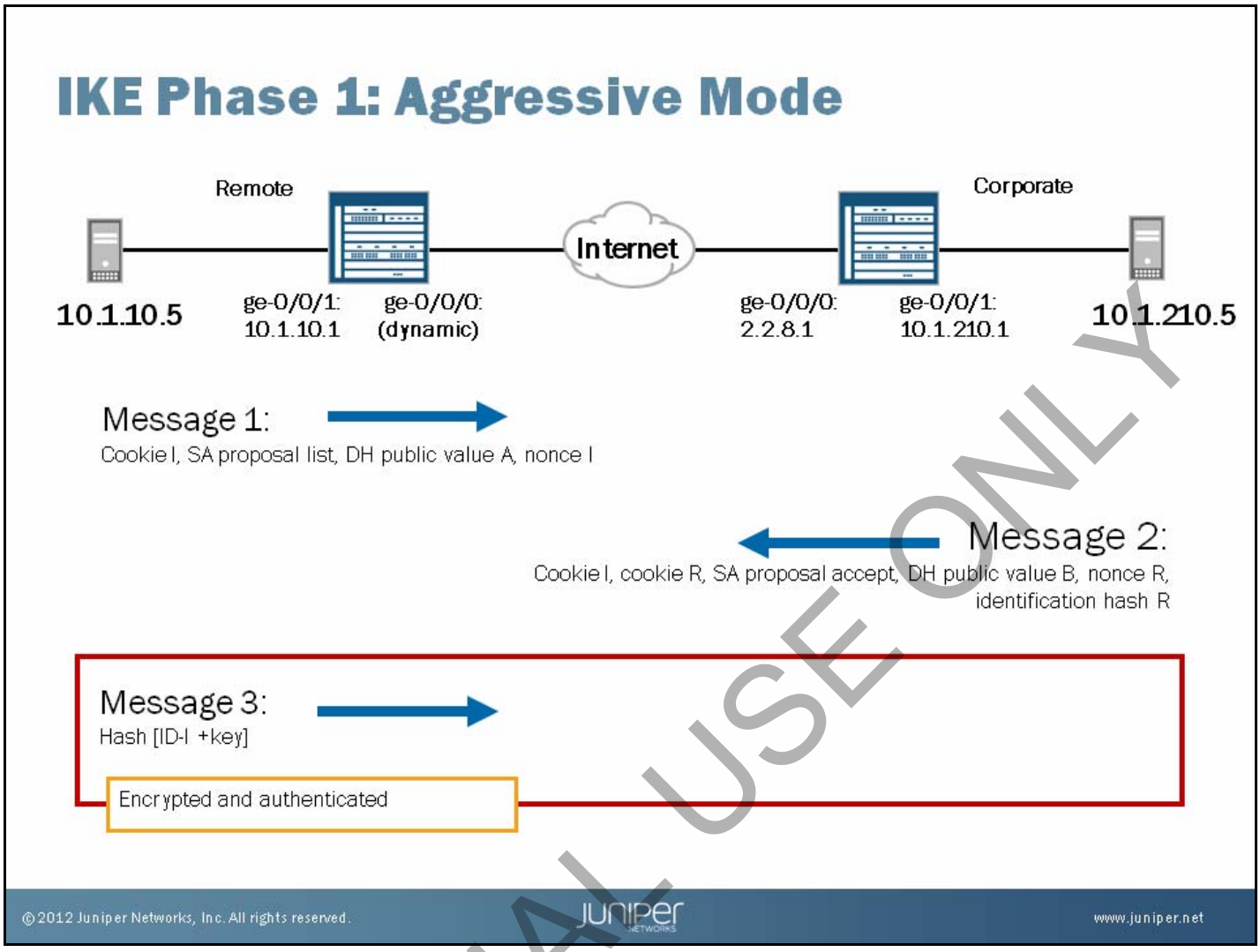
An IPsec SA proposal contains the following:

- Phase 1 authentication method (*main mode* or *aggressive mode*);
- DH group number;
- Encryption algorithm;
- Authentication algorithm; and
- Key lifetime.

For Message 3 and Message 4, the DH public values exchange to create a common session key. Nonces, which are essentially random numbers, also exchange at this time for use as seeds for keys generated later.

After both sides exchange their DH public values, a key is created on each side to encrypt the rest of the IKE Phase 1 messages. The session key is a result of the exchanged public keys traveling to each partner.

Messages 5 and 6 use the pre-shared key in the HMAC algorithm.

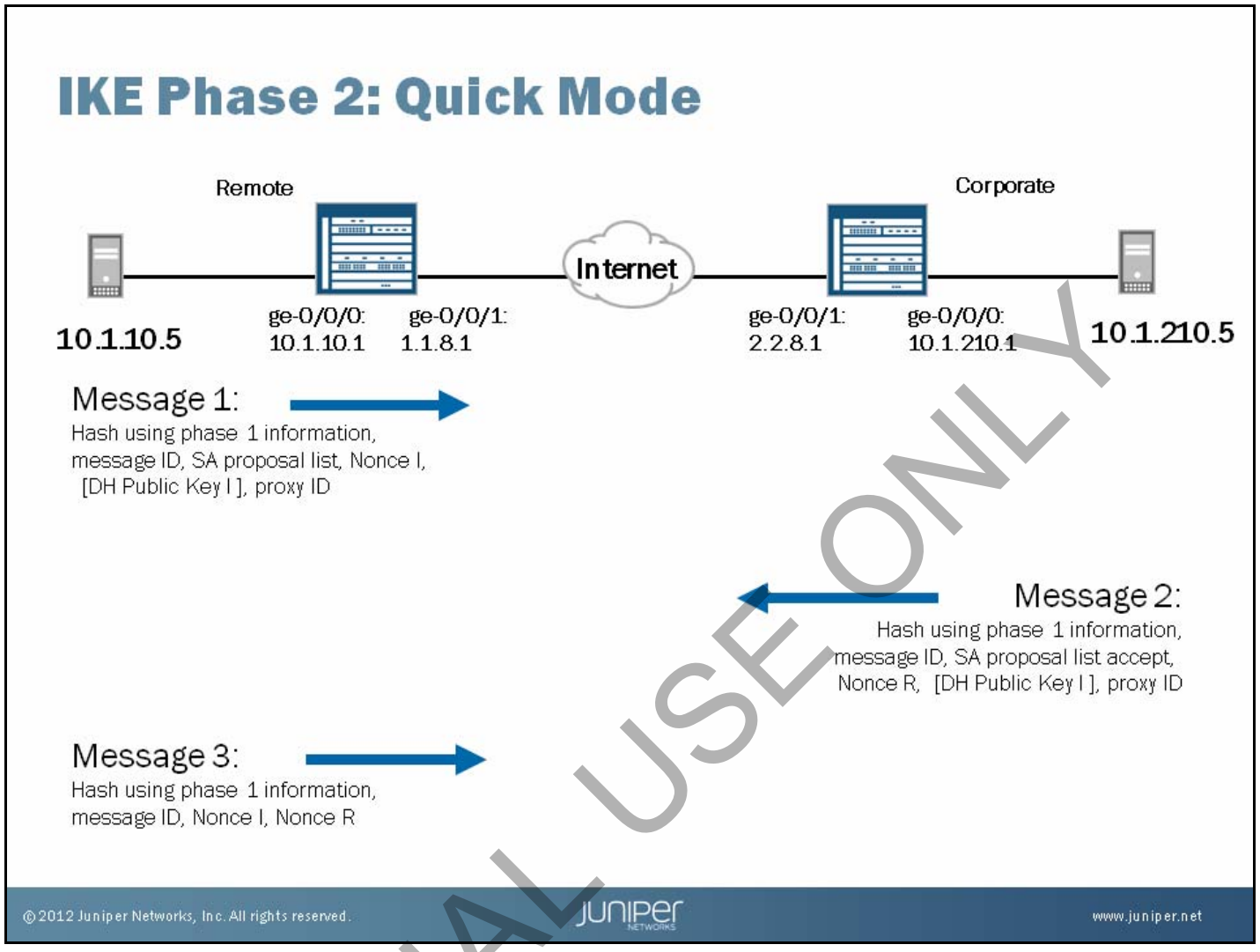


IKE Phase 1: Aggressive Mode

IKE aggressive mode is used when one of the tunnel peers has a dynamic IP address that could be a remote end user dialing in to the Internet, or a remote site using DHCP to acquire an IP address. (Main mode cannot be used because the first two messages validate peer IP addresses. In the case of a dynamic host address, the peer cannot preconfigure the address.)

Phase 1 aggressive mode must initiate by the device with the dynamic IP address. The first two messages negotiate policy and exchange DH public values and nonces. In addition, the second message authenticates the responder; the ID hash is compared with the locally configured peer ID.

The third message authenticates the initiator and provides a proof of participation in the exchange.



IKE Phase 2: Quick Mode

Once Phase 1 is complete, proposals exchange to establish a specific VPN. The following attributes are negotiated in Phase 2:

- Security protocol (ESP or AH);
- Tunnel mode or transport mode;
- Proxy IDs; and
- Optional DH group.

Upon successful completion of quick mode, user data encrypts between the configured IPsec peers. Both tunnel peers must have at least one matching proposal configured for Phase 2 exchange success.

The result of Phase 2 is to create an IPsec VPN for user data to securely transmit through the network.

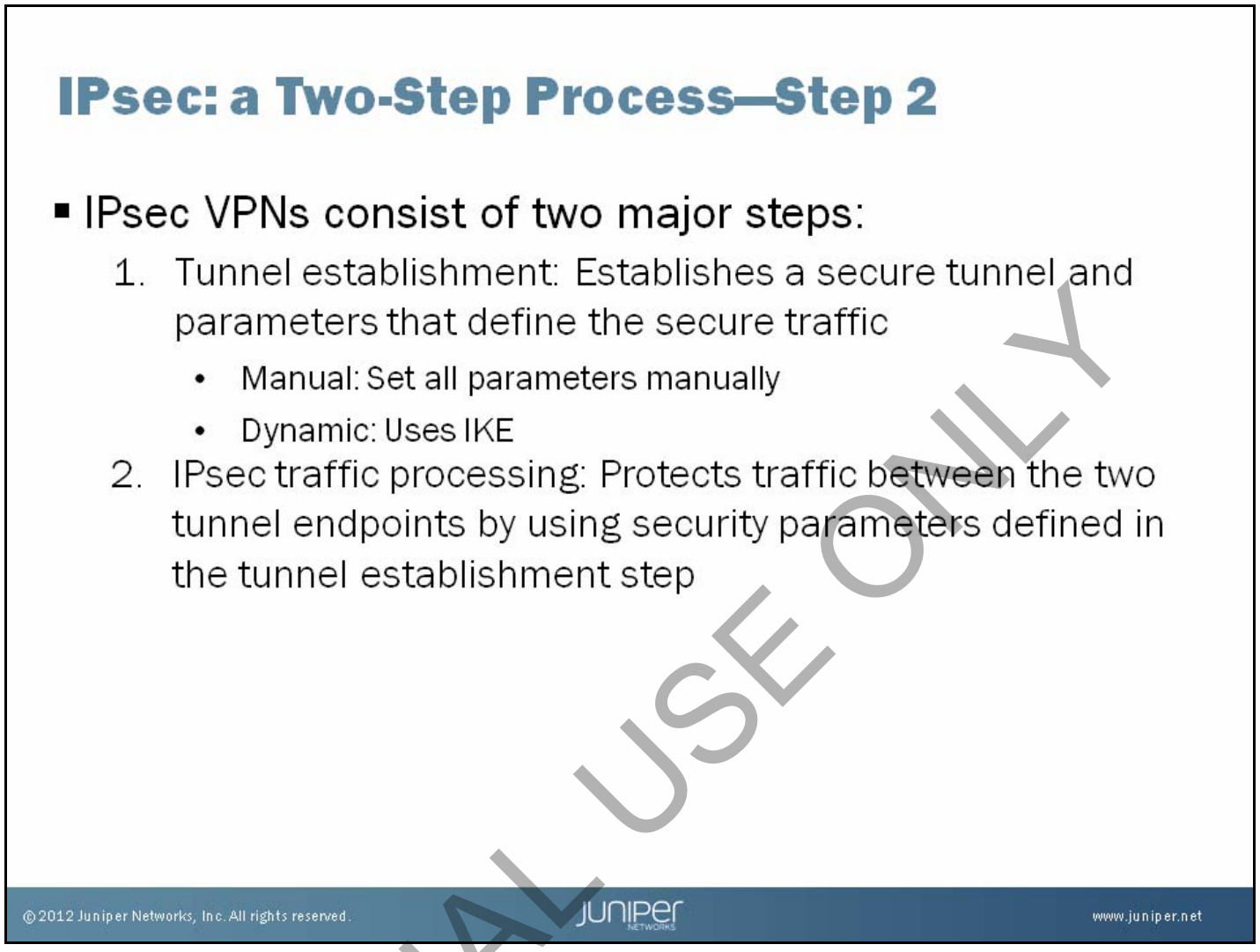
Continued on next page.

IKE Phase 2: Quick Mode (contd.)

For Message 1 and Message 2, a Phase 2 proposal list exchanges. The list contains encrypted and authenticated information that determines the algorithms and keys for encrypting and authenticating user data. The Phase 2 proposal list contains the following items:

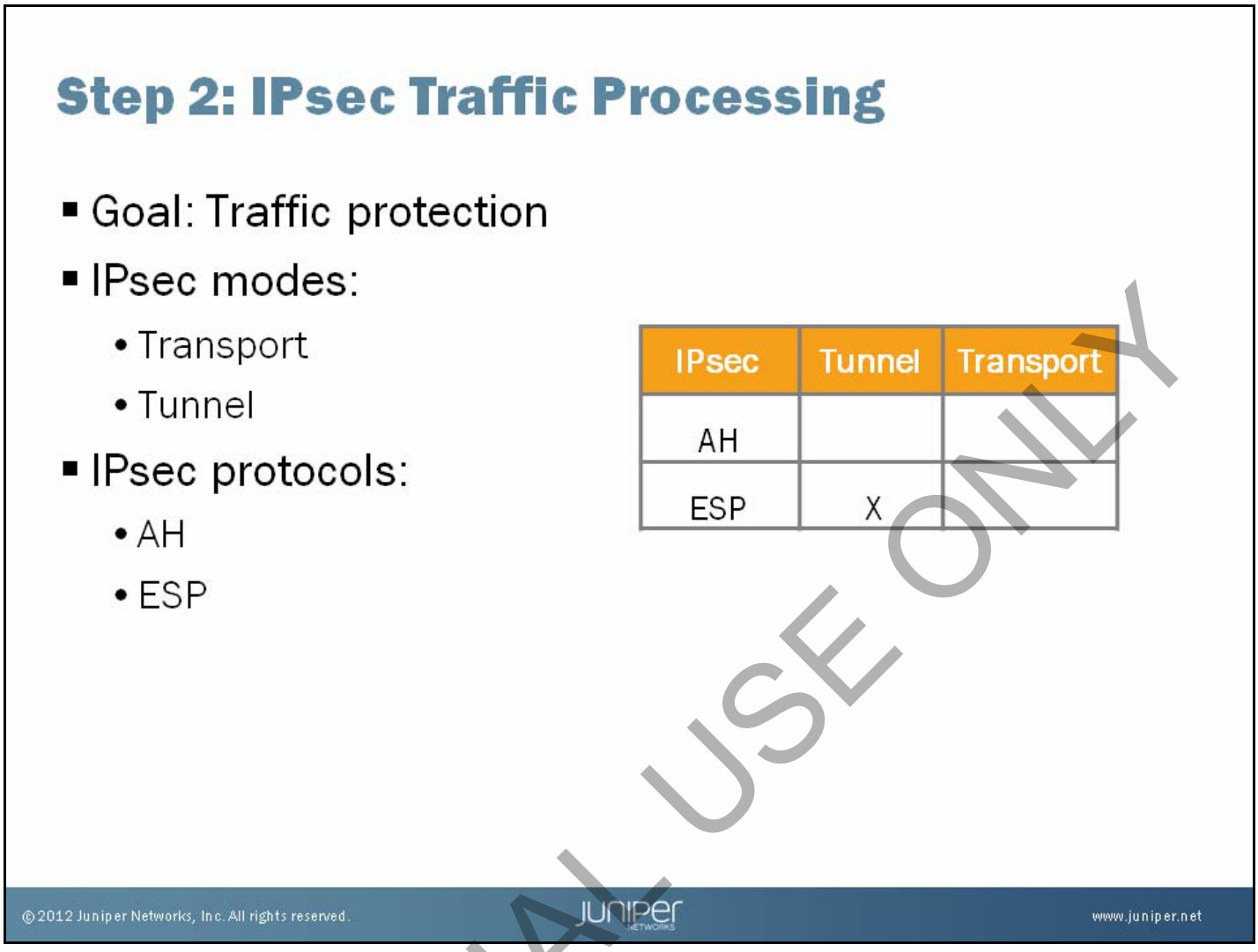
- ESP or AH;
- DH group number (0 for no PFS);
- Encryption algorithm;
- Authentication algorithm;
- Key lifetime;
- Proxy ID (policy rule); and
- DH public keys (optional if using PFS).

Message 3 acknowledges information sent from quick mode Message 2 so that the Phase 2 tunnel can establish.



IPsec: A Two-Step Process—Step 2

Now that we have covered the tunnel establishment step of the IPsec process, we cover the next step—IPsec traffic processing. Once the IPsec tunnel establishes, the devices are ready to send the payload using the IPsec attributes and protocols, which ensure payload protection.



Goal of IPsec Traffic Processing

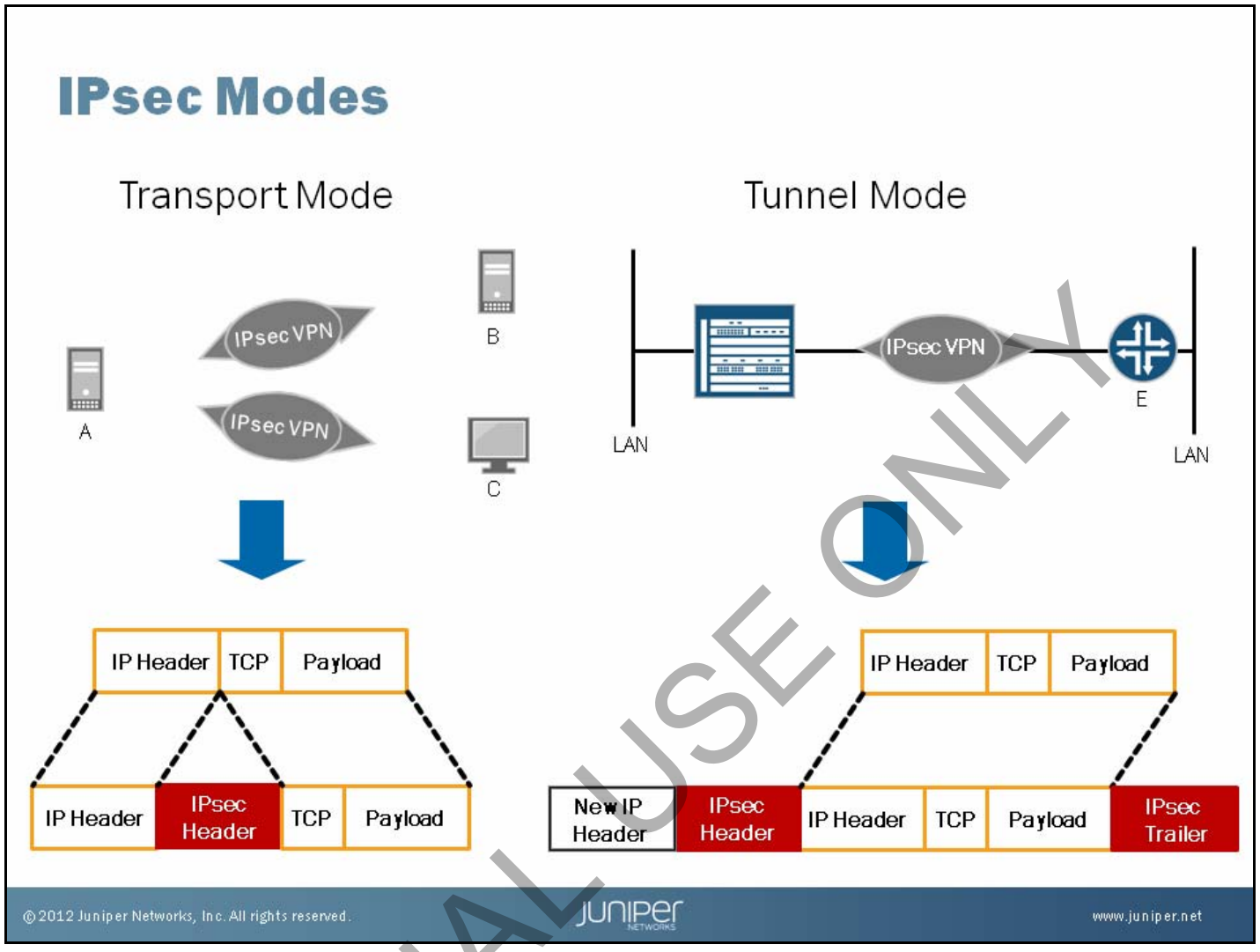
During the IPsec traffic processing step, the devices have one goal—to ensure traffic protection. The most commonly used IPsec mode is ESP tunnel mode.

IPsec Modes

IPsec handles the payload using one of two modes—transport or tunnel. We discuss each mode in the next few pages.

IPsec Protocols

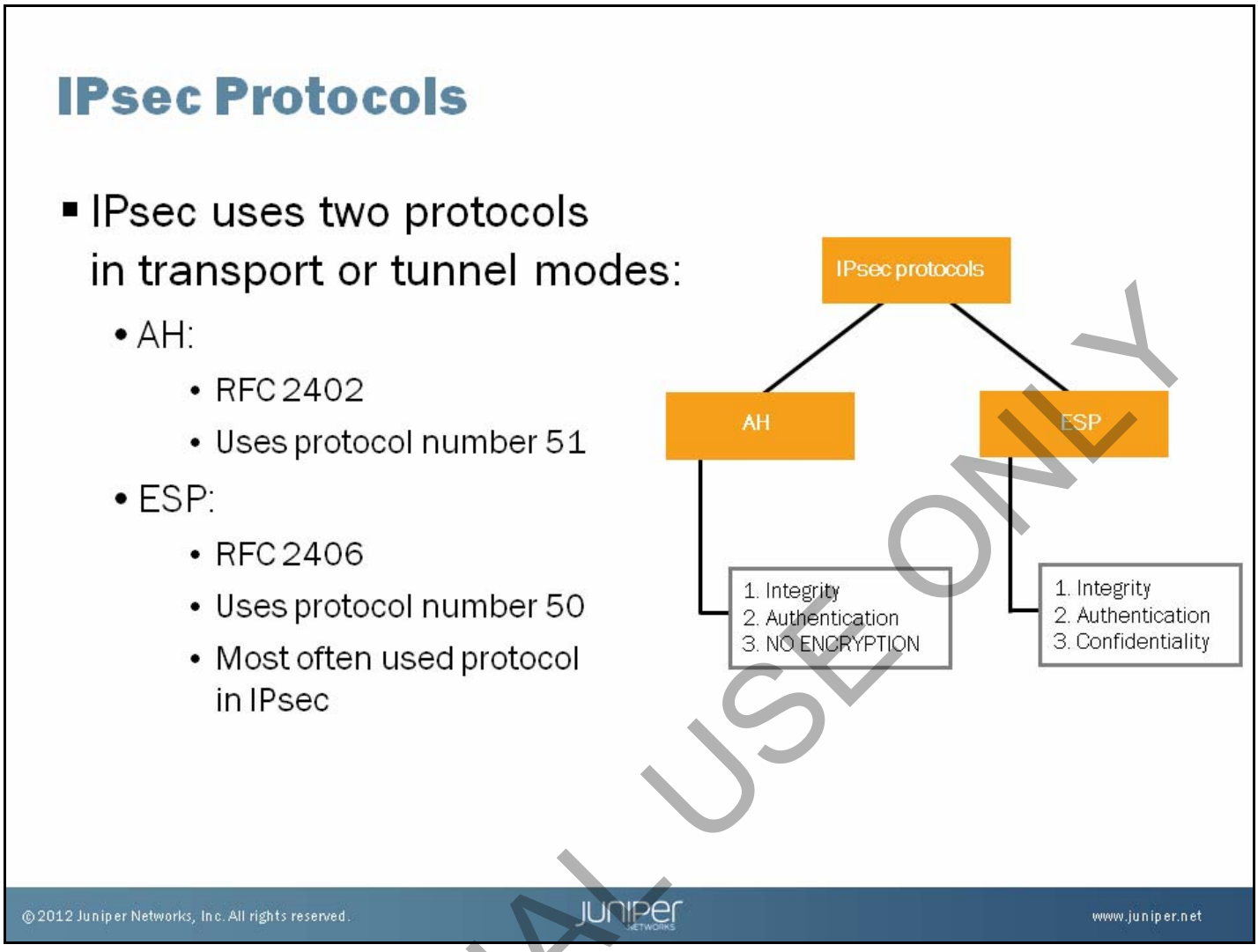
IPsec uses two protocols to ensure payload security—the AH protocol and the ESP protocol. Again, we discuss each of these protocols in the next few pages.



IPsec Modes

You can implement IPsec in the following two modes:

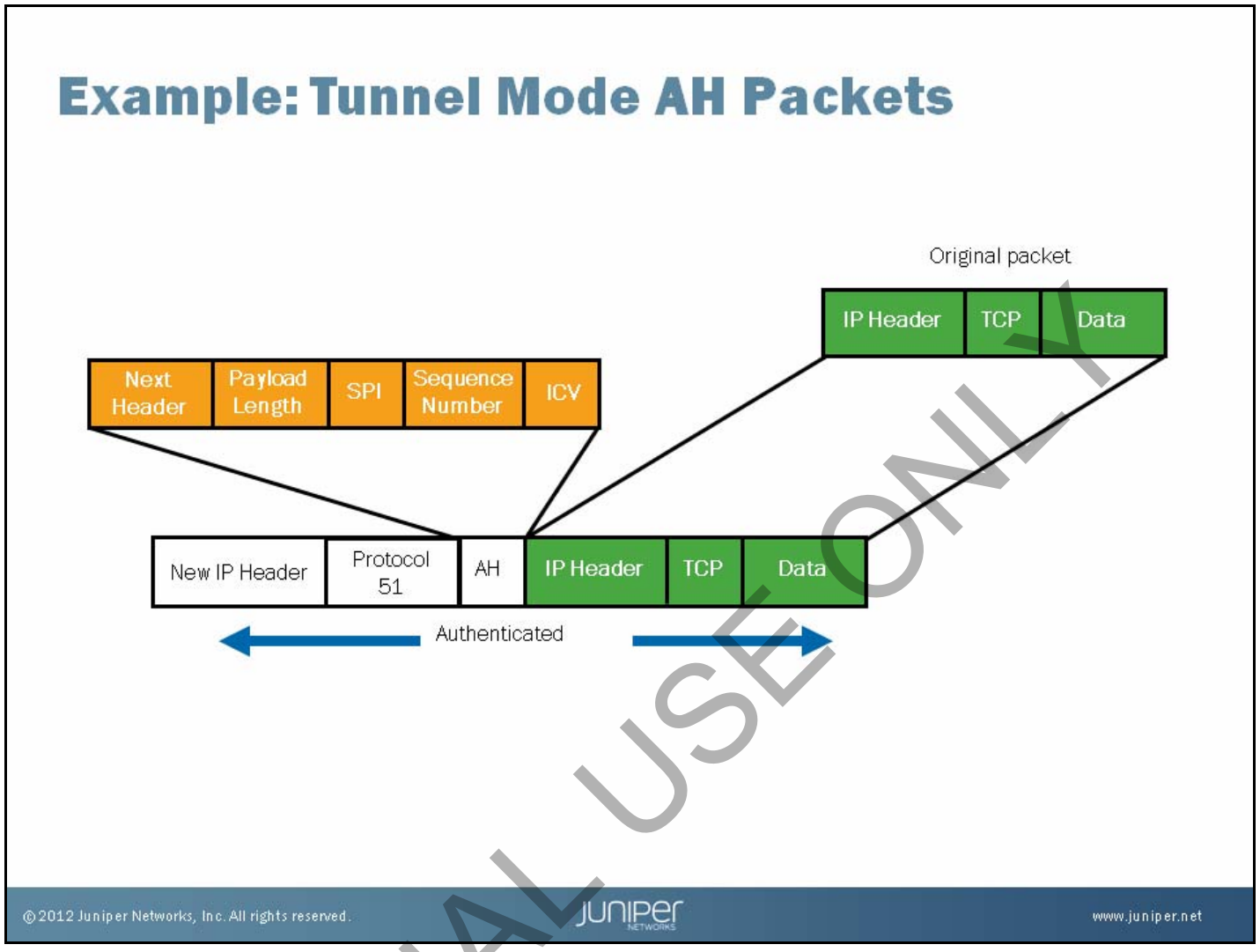
- *Tunnel mode*: This mode is the most commonly implemented method. Tunnel mode is implemented between IPsec gateways or an IPsec gateway and a remote client providing secure access to the networks behind the gateway. In this method, end systems need not be aware of the IPsec protocol suite. All encryption and decryption takes place on the IPsec gateways on behalf of the hosts behind the gateway.
- *Transport mode*: This mode is implemented between IPsec end systems. End systems should be aware of the IPsec protocol suite. They do all the encryption and decryption of data.



IPsec Protocols

Two protocols exist that IPsec can use to ensure payload security—AH protocol and ESP:

- AH provides only data integrity, authentication, and antireplay services. AH is identified by IP protocol number 51. It uses MD5 or SHA-1 to provide data integrity services. AH provides no encryption of data in the packets.
- ESP provides data confidentiality, data integrity, authentication, and antireplay services. It does not use a transport protocol like TCP or UDP; it rides directly on top of IP using protocol number 50. ESP uses symmetric key algorithms like DES, triple Data Encryption Standard (3DES), or AES, and hash methods like MD5 and SHA-1 to provide security services. Antireplay services ensure that third parties cannot capture and retransmit datagrams. By checking sequence numbers, a receiver can determine receipt of a packet and can discard any repetitions.

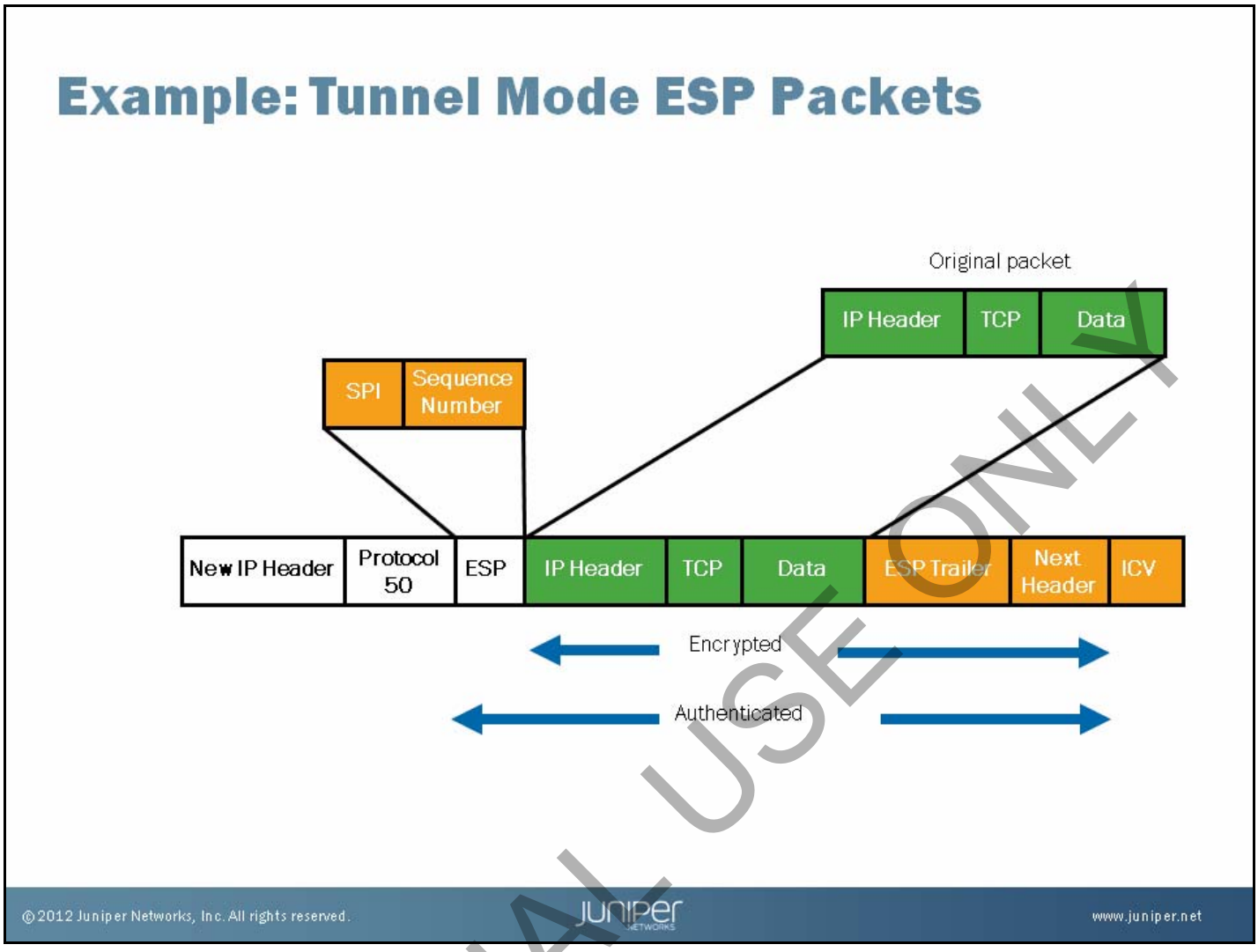


Example: Tunnel Mode AH Packets

AH authenticates only the immutable fields in the IP header. Fields like time to live (TTL) and type of service (ToS) change during packet transit, so these fields do not receive authentication. The new IP header contains protocol number 51, signifying AH.

The AH header contains the following items:

- *Next header*: Information on the next expected segment;
- *Payload length*: Indicates the size of the payload;
- *SPI*: An arbitrary 32-bit value that, in combination with the destination IP address and security protocol (AH), uniquely identifies the security association for this datagram; and
- *Sequence number*: An unsigned 32-bit field containing a monotonically increasing counter value (sequence number). It is used to detect antireplay.



Example: Tunnel Mode ESP Packets

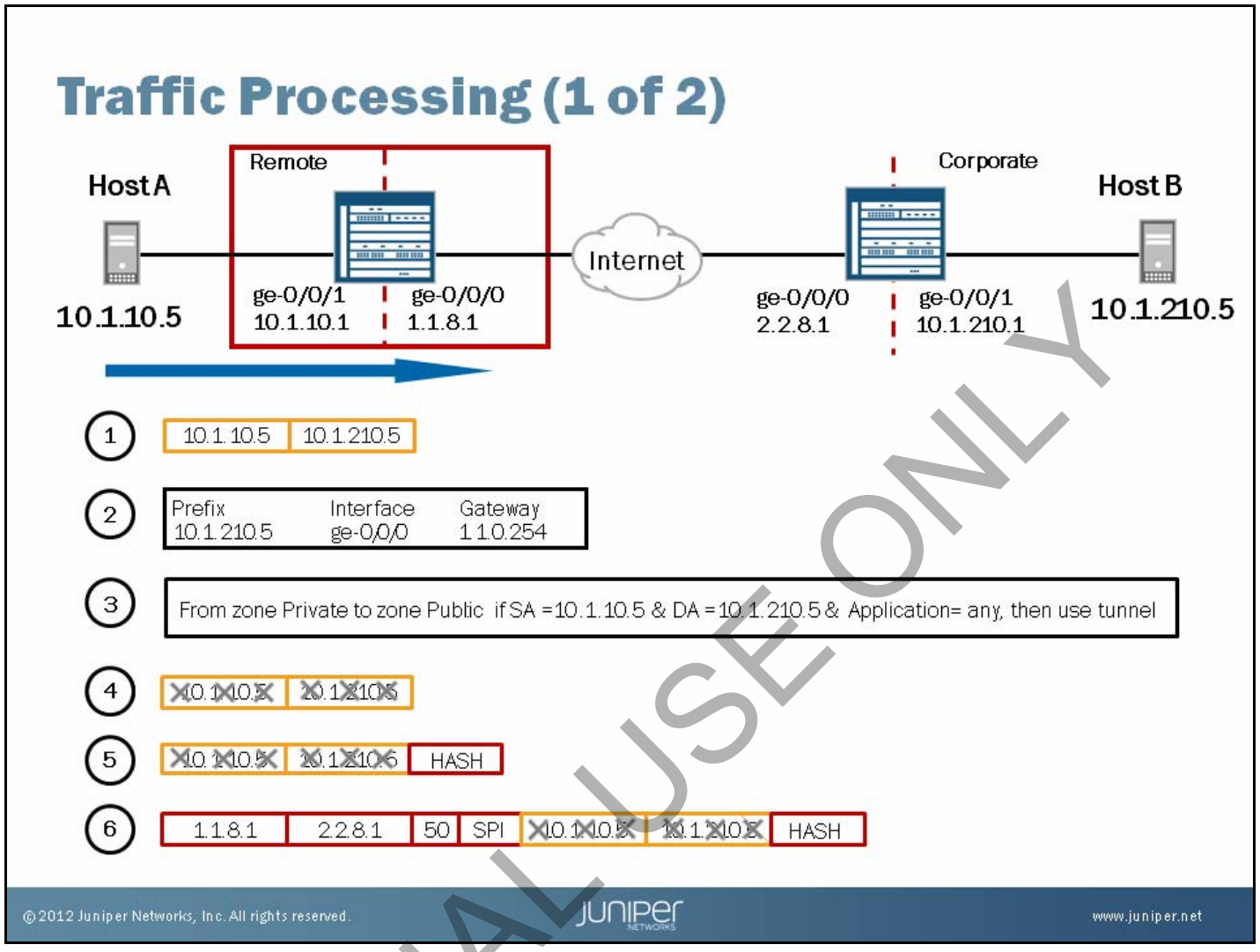
In tunnel mode, the ESP header inserts between the new IP header and the original IP header. The new IP header contains protocol 50, representing ESP. The ESP header contains the following information:

- *SPI*: An arbitrary 32-bit value that, in combination with the destination IP address and security protocol (ESP), uniquely identifies the security association for this datagram; and
- *Sequence number*: An unsigned 32-bit field containing a monotonically increasing counter value (sequence number); it is used to detect antireplay.

The ESP trailer contains the following information:

- *Padding/pad length*: Depending on original data size, padding might be required to fill the packet; and
- *Next header*: Information on the next expected segment.

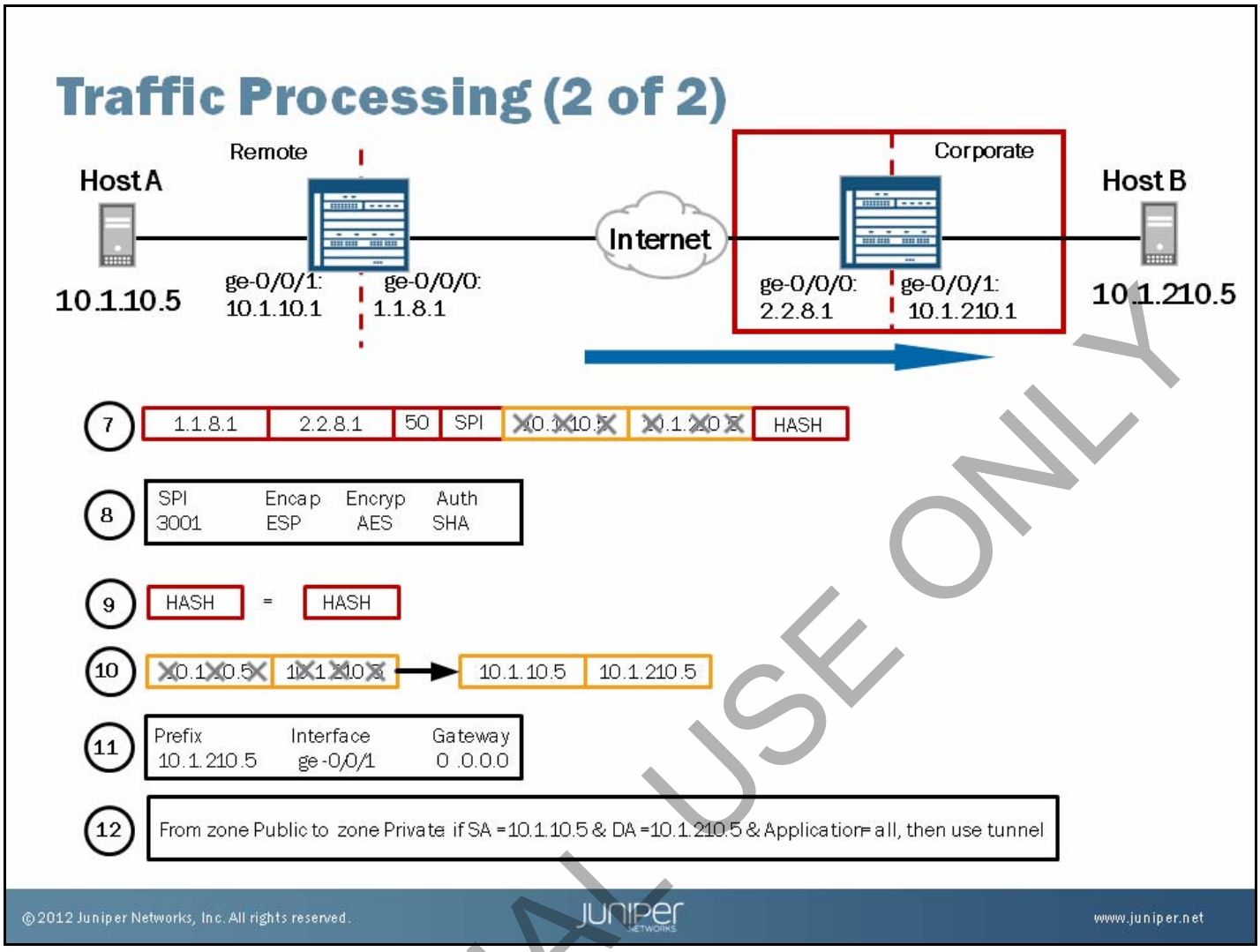
ESP Auth is the integrity check value (that is, the hash value) for this packet.



Traffic Processing: Part 1

The following list describes the order of traffic processing:

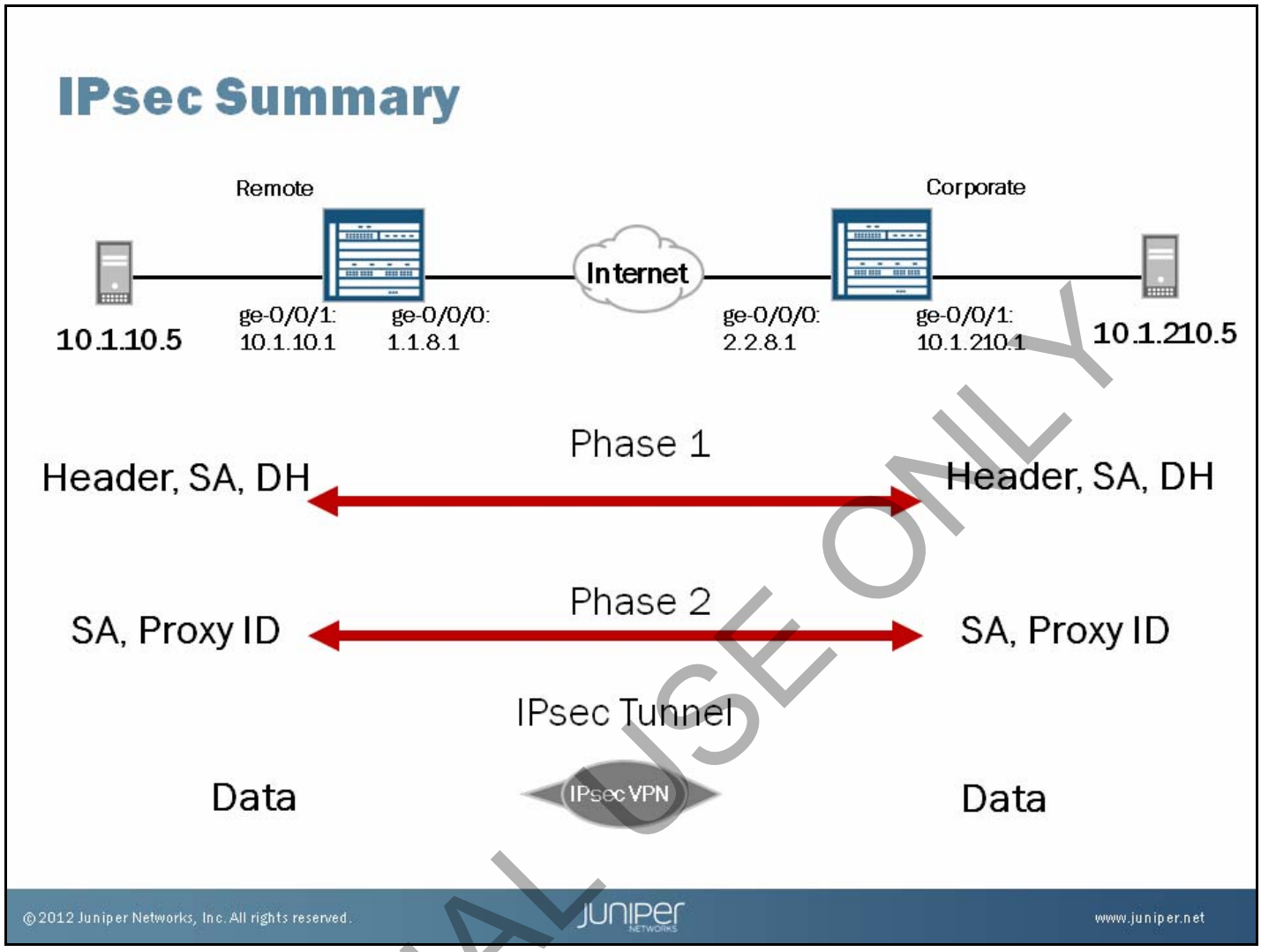
1. The packet arrives at the remote Junos security platform.
2. The Junos OS looks up the destination route and determines the Egress Zone.
3. The Junos OS looks up the security policy. The traffic matches a tunnel policy.
4. The original packet receives encryption.
5. The Junos OS hashes the packet with an authentication key.
6. The Junos OS builds the tunnel packet with a new IP header, IPsec header, and hash value. The new packet travels to the tunnel peer.



Traffic Processing: Part 2

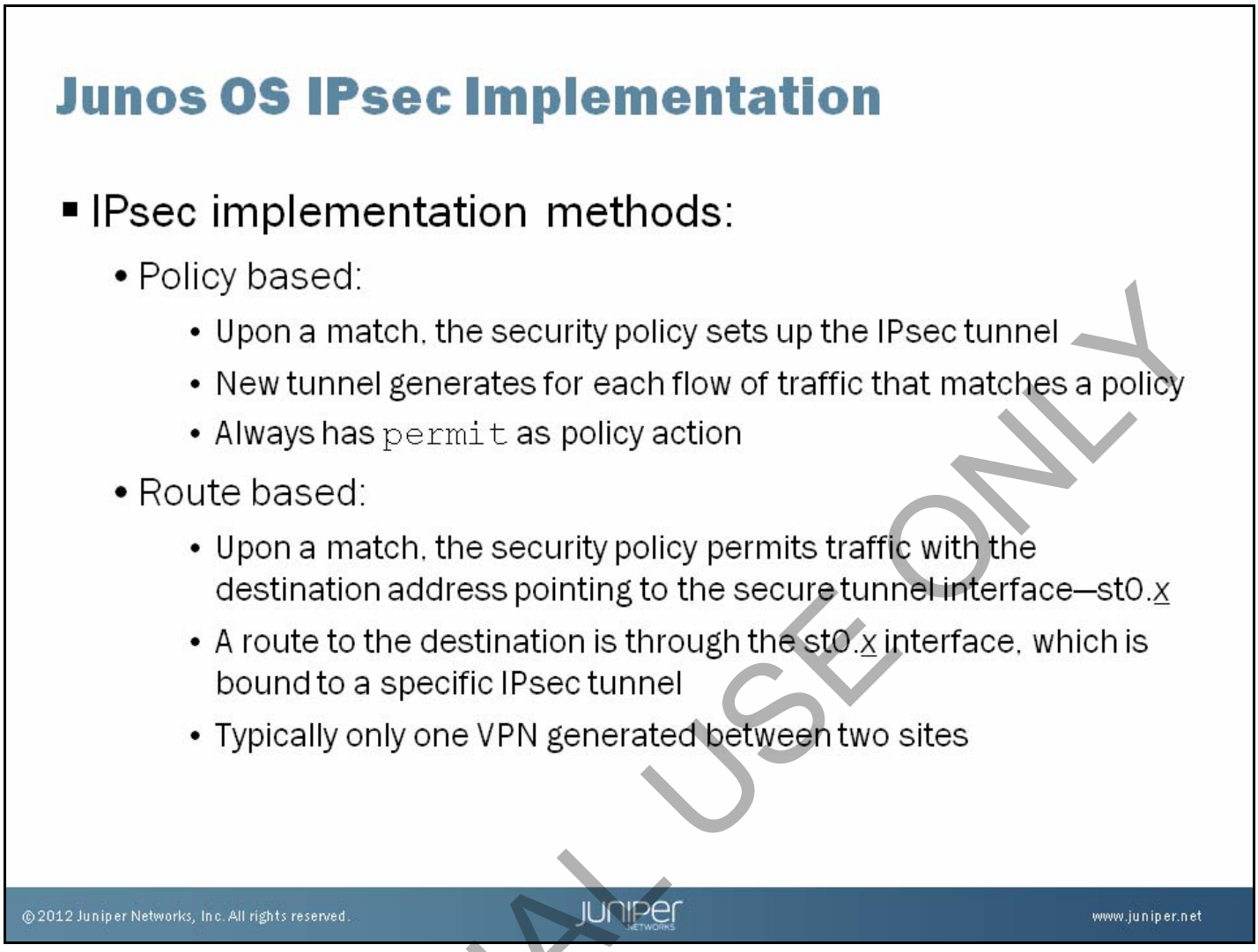
This list is a continuation of the list from the previous page, describing the order of traffic processing:

7. The corporate Junos security platform receives the encrypted packet.
8. The Junos OS looks up the incoming SPI in the local SA database. The matching record contains encryption and authentication algorithms, and keys.
9. The Junos OS compares the locally calculated hash with the received hash.
10. The Junos OS decrypts the packet.
11. The Junos OS performs the routing lookup for the decrypted packet and determines the Egress Zone.
12. The Junos OS checks the associated security policy. It forwards the packet if the tunnel policy exists for the packet.



IPsec Processing Summary

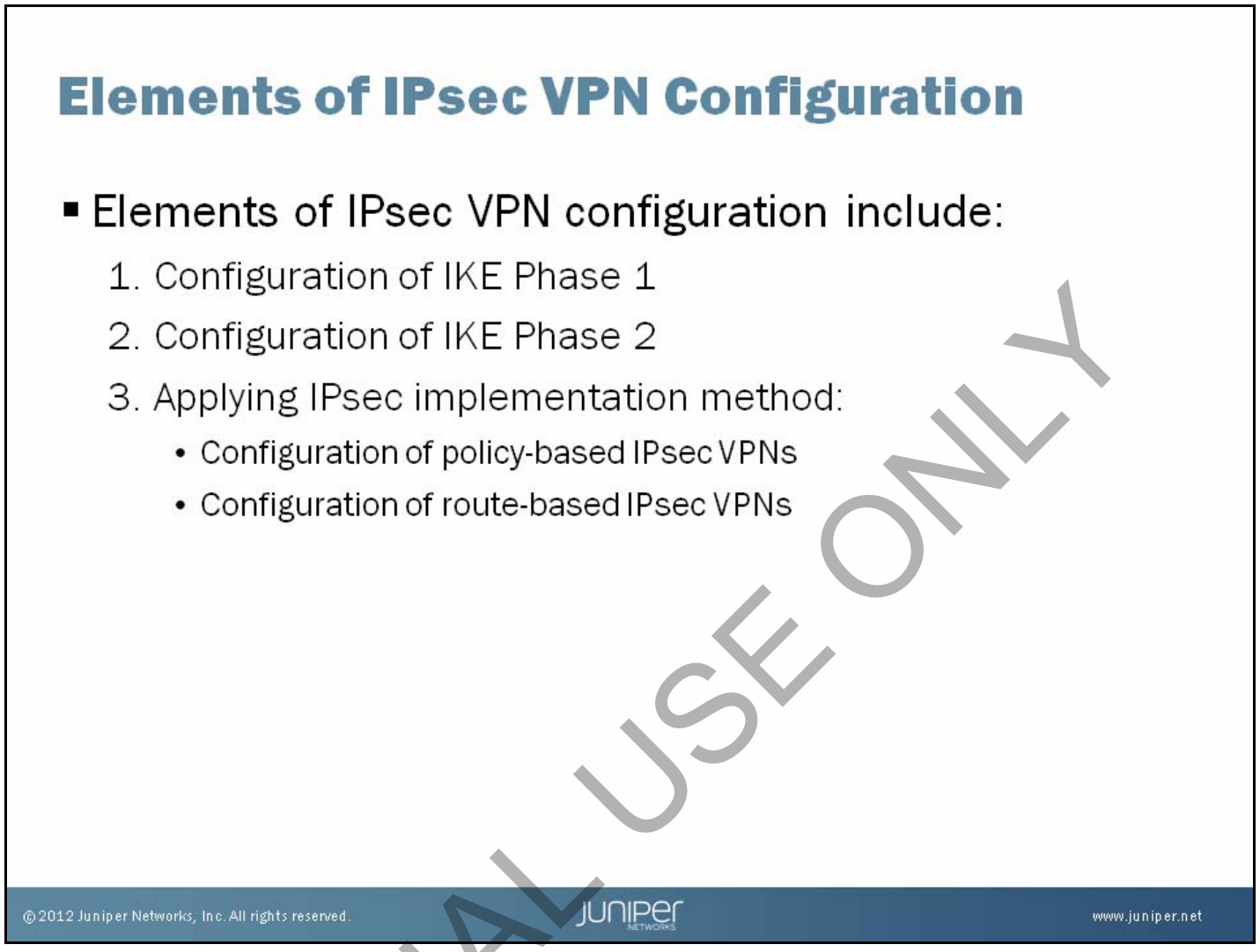
The slide provides a summary of all the steps of IPsec traffic processing.



IPsec Implementation Methods

The Junos OS offers two methods for IPsec VPN implementation:

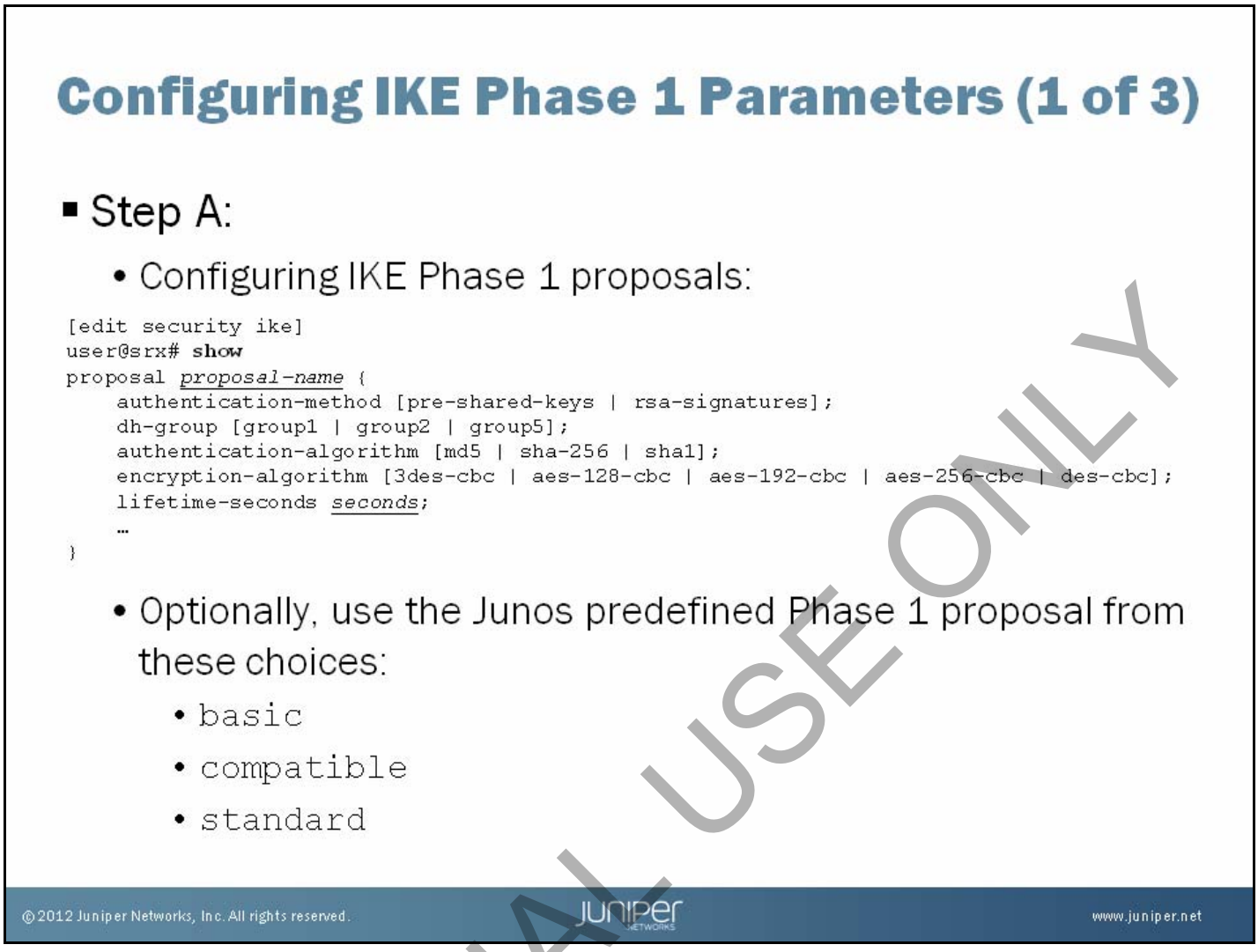
- *Policy-based VPNs*: To implement this method, a security policy specifies as its action the IPsec VPN tunnel for transit traffic that meets the policy's match criteria. Policy-based VPNs are required when one endpoint of the tunnel uses dynamic addressing. For policy-based IPsec VPNs, a new tunnel generates for each flow of traffic that matches the policy. Because each tunnel requires its own negotiation process and a separate pair of SAs, the use of policy-based IPsec VPNs can require more resources than route-based IPsec VPNs.
- *Route-based VPNs:* Unlike the process for policy-based IPsec VPNs, for route-based IPsec VPNs, a policy refers to a destination address—not an IPsec VPN tunnel. Because a destination address is used, route-based VPNs are generally the best VPNs to use when a routing protocol adjacency must be formed across the tunnel. When the Junos OS searches a route that must send traffic to the destination address, it finds a route associated with a secure tunnel interface (st0.x). The tunnel interface is bound to a specific IPsec VPN tunnel, and traffic routes to the tunnel if the policy action is `permit`. With a route-based IPsec VPN, in most cases, only one VPN exists between two sites.



Elements of IPsec VPN Configuration:

IPsec VPN configuration consists of three steps:

1. Configuring IKE Phase 1;
2. Configuring IKE Phase 2; and
3. Applying the IPsec implementation method, which includes implementing either policy-based VPNs or route-based VPNs.



Configuring IKE Phase 1 Parameters: Step A

IKE Phase 1 configuration requires that you perform the following steps:

A. Configure IKE Phase 1 proposals;

B. Configure IKE policies and reference the proposals; and

C. Configure the IKE gateway and reference the policy.

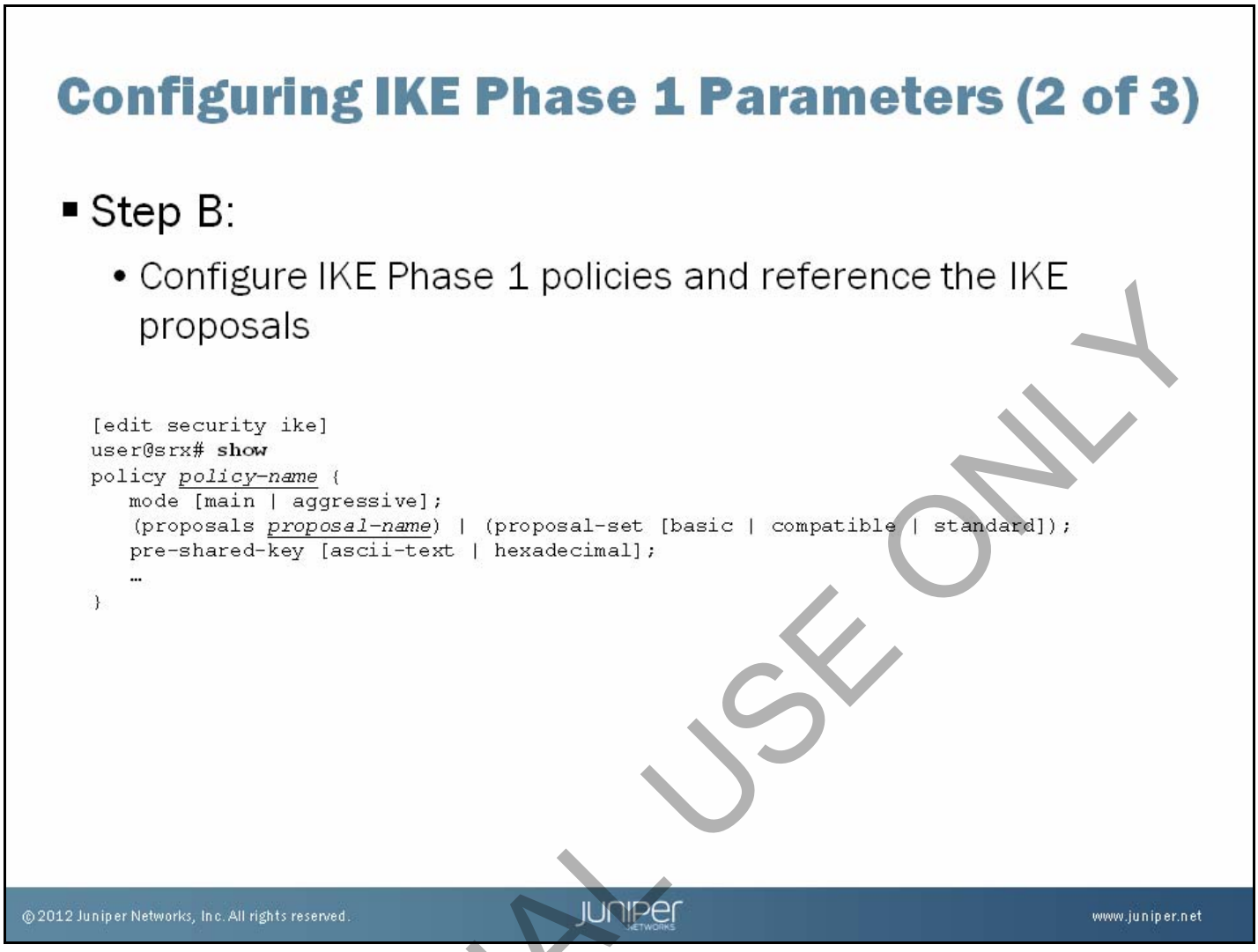
The slide addresses Step A, which is optional, because you can use the Junos predefined proposals. The following are the predefined proposals:

- `basic`:
  - *Proposal 1*: preshared key, DH g1, DES, and SHA1
  - *Proposal 2*: preshared key, DH g1, DES, and MD5
- `compatible`:
  - *Proposal 1*: preshared key, DH g2, 3DES, and SHA1
  - *Proposal 2*: preshared key, DH g2, 3DES, and MD5
  - *Proposal 3*: preshared key, DH g2, DES, and SHA1
  - *Proposal 4*: preshared key, DH g2, DES, and MD5

Continued on next page.

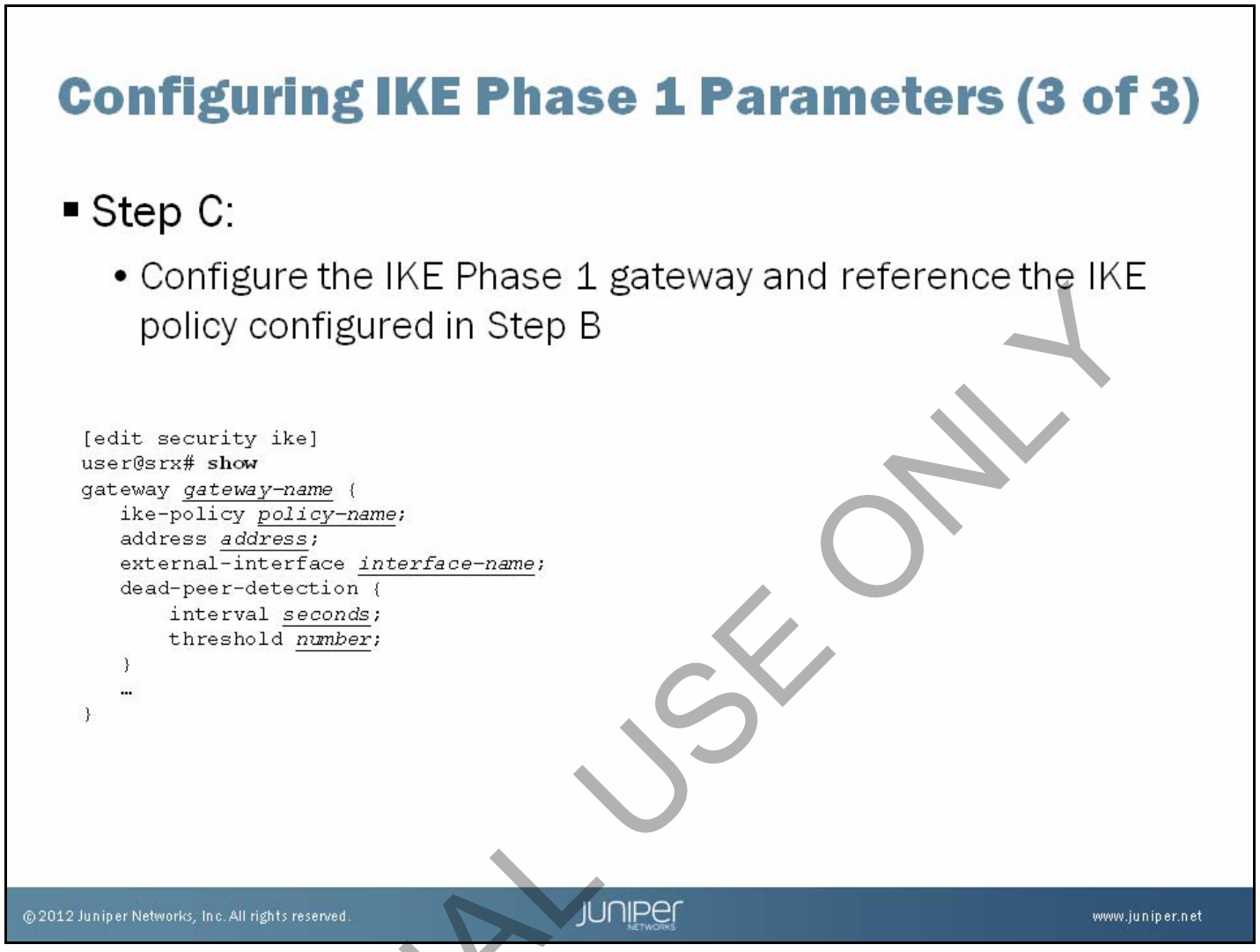
Configuring IKE Phase 1 Parameters: Step A (contd.)

- `standard`:
  - *Proposal 1*: preshared key, DH g2, 3DES, and SHA1
  - *Proposal 2*: preshared key, DH g2, AES128, and SHA1



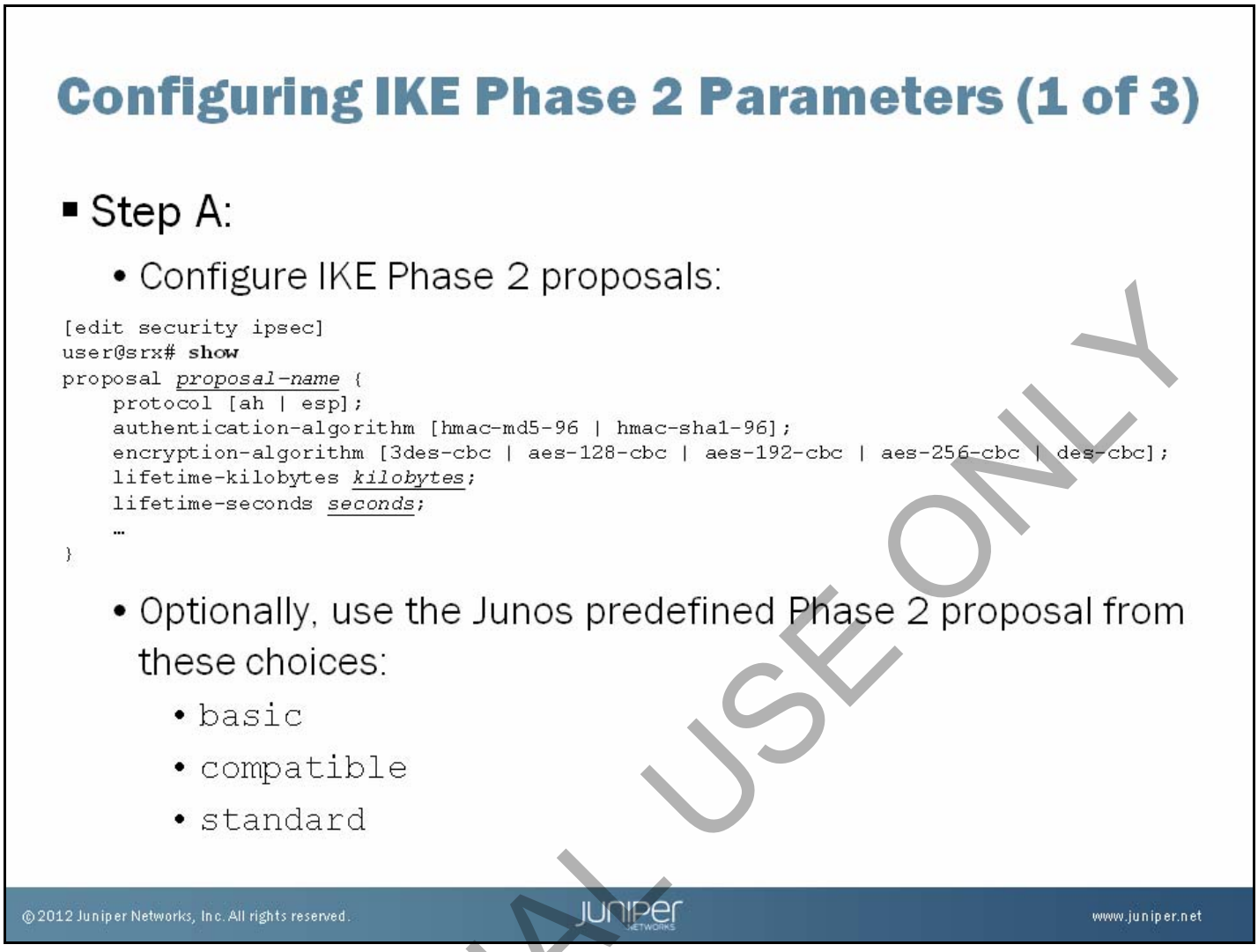
Configuring IKE Phase 1 Parameters: Step B

The slide illustrates the syntax for Step B of IKE Phase 1 configuration, which is policy configuration. For this step you must either refer to the preconfigured proposal from Step A or use a system-defined proposal. Also, within the policy you must specify the preshared key and mode of IKE—main or aggressive.



Configuration of IKE Phase 1 Parameters: Step C

The slide illustrates the last step of IKE Phase 1 configuration, which is gateway configuration. In this step you must refer to the policy configured in the previous step, define the peer address, and specify the outgoing interface. Optionally, you can configure dead peer detection (DPD) to send a DPD request packet if the device does not receive traffic from a peer for the number of seconds specified with the **`interval`** option. You can also configure DPD to consider the peer unavailable after a threshold number of interval periods is reached. For example, assume that the interval value is 10 seconds and the threshold value is 5. If the device does not receive traffic from a peer for 10 seconds, it sends a DPD request packet to it. The Junos security platform then considers the peer unavailable after five sequences of waiting for 10 seconds.



Configuring IKE Phase 2 Parameters: Step A

IKE Phase 2 configuration requires that you configure the following steps:

A. IKE Phase 2 proposals;

B. IKE Phase 2 policies; and

C. IKE Phase 2 VPN tunnel.

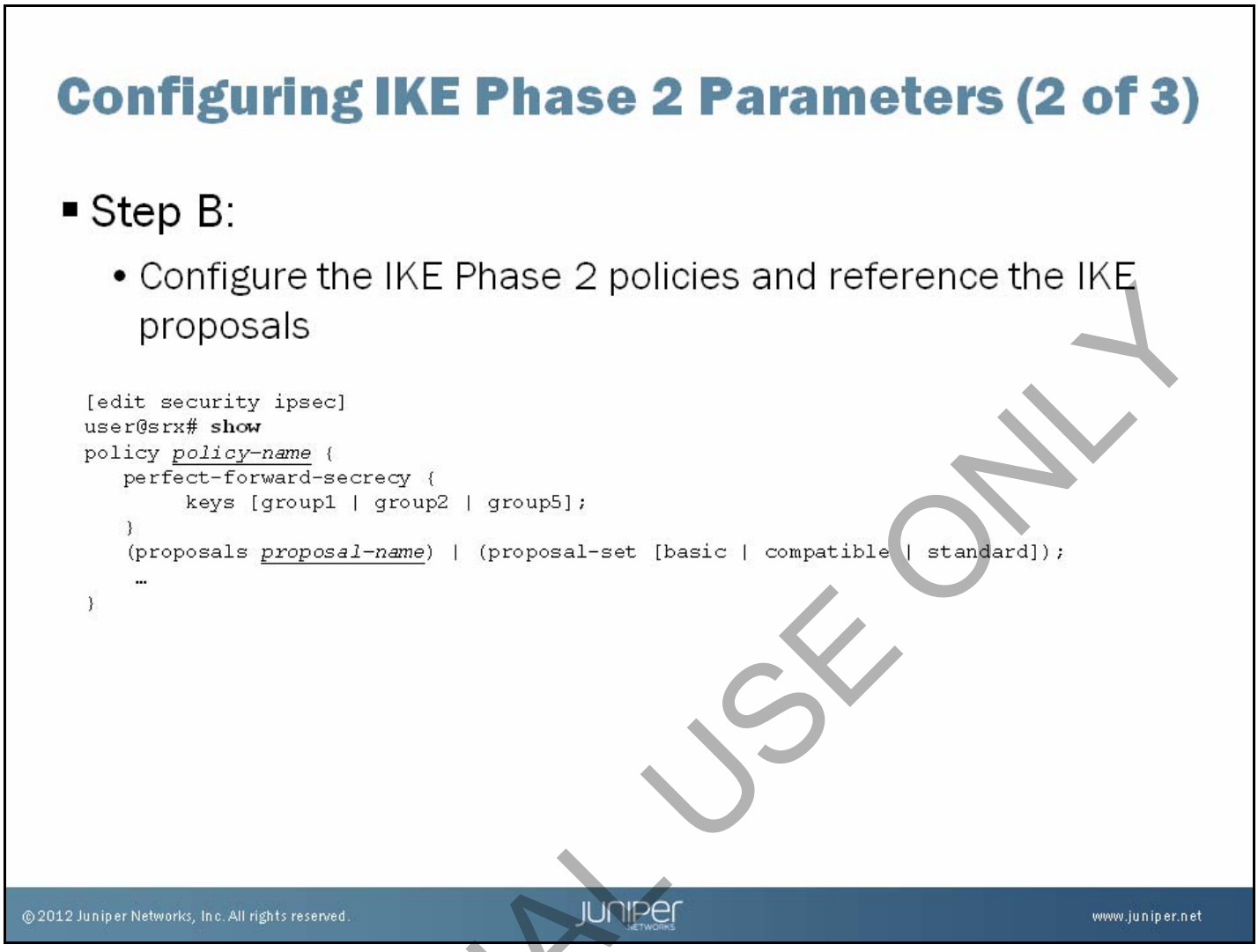
The slide addresses Step A, which is optional, because you can use predefined proposals. The following are the predefined proposals:

- `basic`:
  - *Proposal 1*: no PFS, ESP, DES, and SHA1
  - *Proposal 2*: no PFS, DH g1, DES, and MD5
- `compatible`:
  - *Proposal 1*: no PFS, ESP, 3DES, and SHA1
  - *Proposal 2*: no PFS, ESP, 3DES, and MD5
  - *Proposal 3*: no PFS, ESP, DES, and SHA1
  - *Proposal 4*: no PFS, ESP, DES, and MD5

Continued on next page.

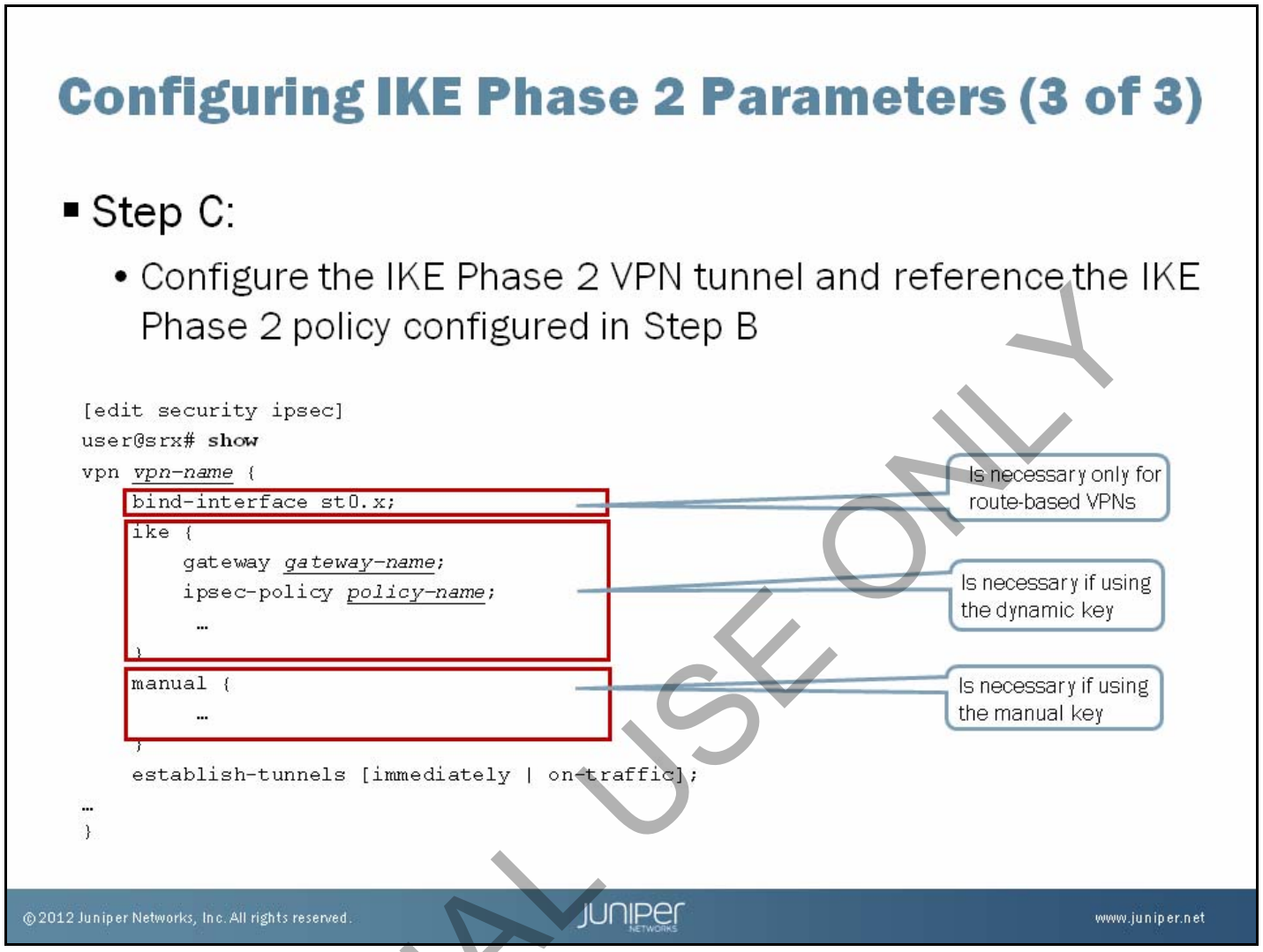
Configuring IKE Phase 2 Parameters: Step A (contd.)

- `standard`:
  - *Proposal 1*: ESP, DH g2, 3DES, and SHA1
  - *Proposal 2*: ESP, DH g2, AES128, and SHA1



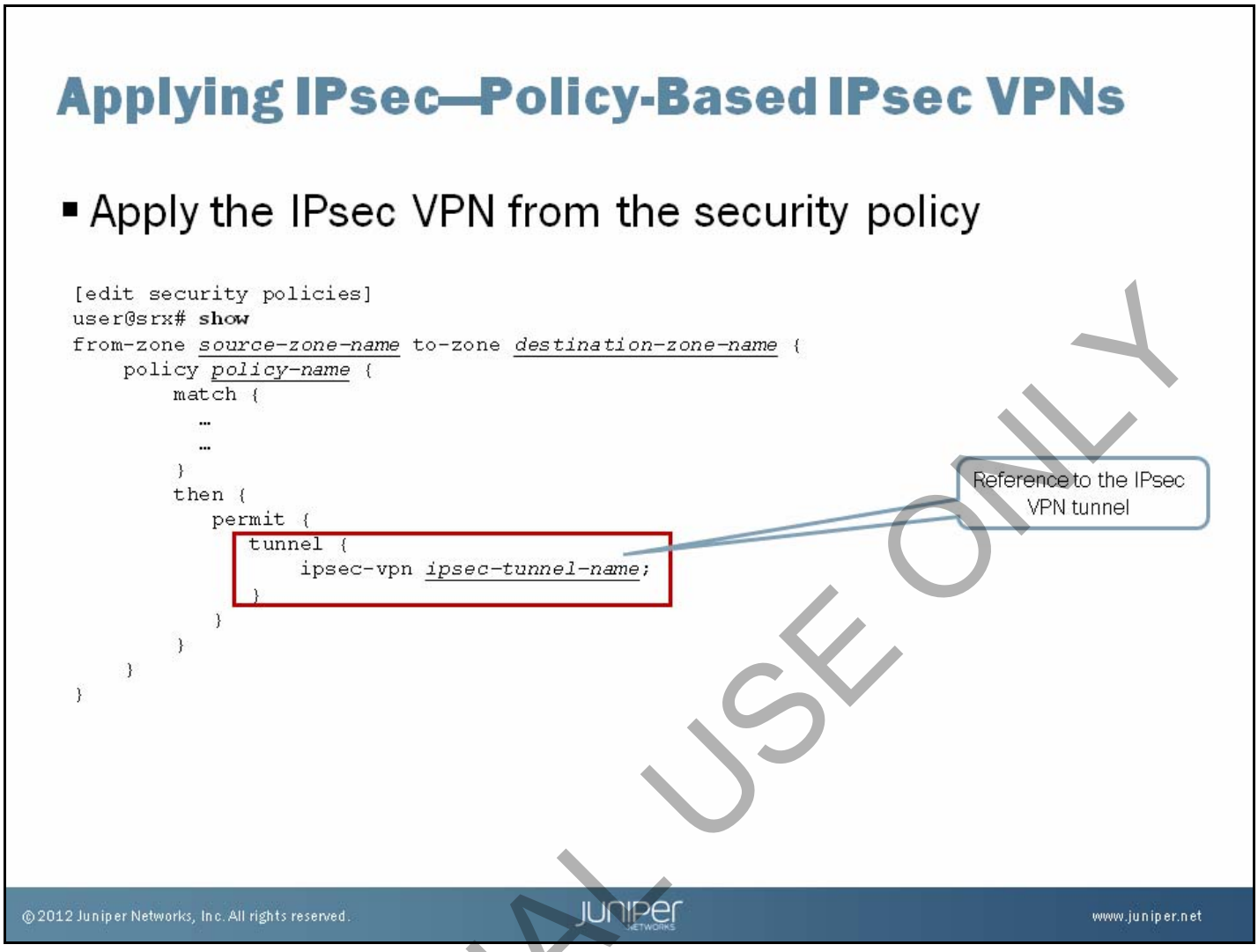
Configuring IKE Phase 2 Parameters: Step B

The slide illustrates the syntax for Step B of IKE Phase 2 configuration, which is policy configuration. For this step, you must either refer to the preconfigured proposal from Step A or use a system-defined proposal. Also, within the policy, you have the option to configure PFS to use with the three supported groups of DH as the method for the Junos OS to generate the encryption key.



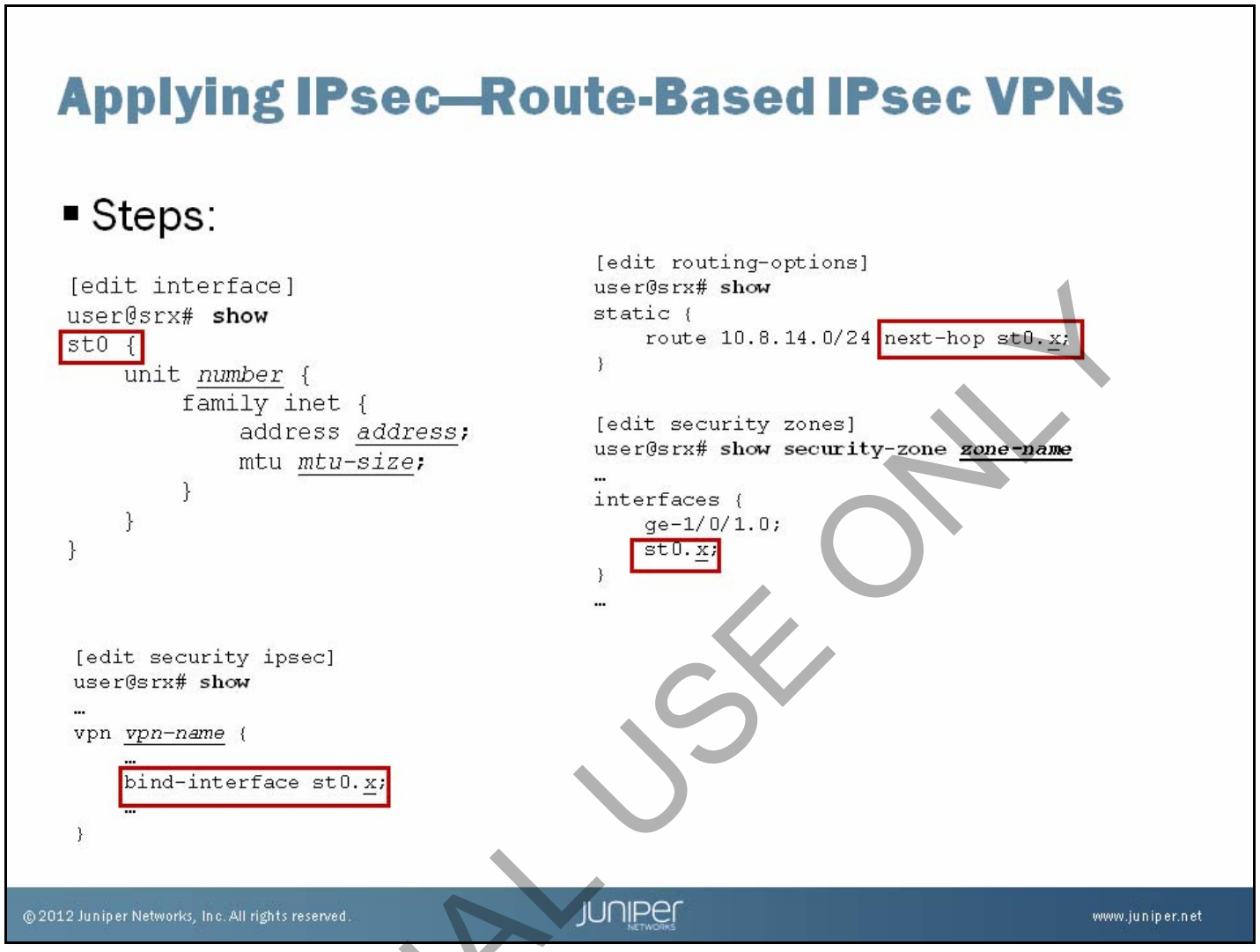
Configuring IKE Phase 2 Parameters: Step C

The slide illustrates the last step of IKE Phase 2 configuration, which is VPN configuration. In this step, you must refer to the policy defined in the previous step, as well as the gateway preconfigured in Step C of IKE Phase 1. If you are configuring a route-based VPN, you must bind the st0.x interface to the VPN, as illustrated on the slide. If you manually set up a tunnel, you must specify all the necessary attributes manually. Should you choose to do so, you can set up all the necessary parameters under the `security ipsec vpn` configuration stanza. The optional **`establish-tunnels`** command specifies when to activate IKE-- **`immediately`**, or **`on-traffic`**. The **`immediately`** option signals the device to activate IKE immediately after VPN configuration or configuration changes are committed. The **`on-traffic`** option signals the device to activate IKE only when payload traffic flows.



Applying IPsec—Policy-Based IPsec VPNs

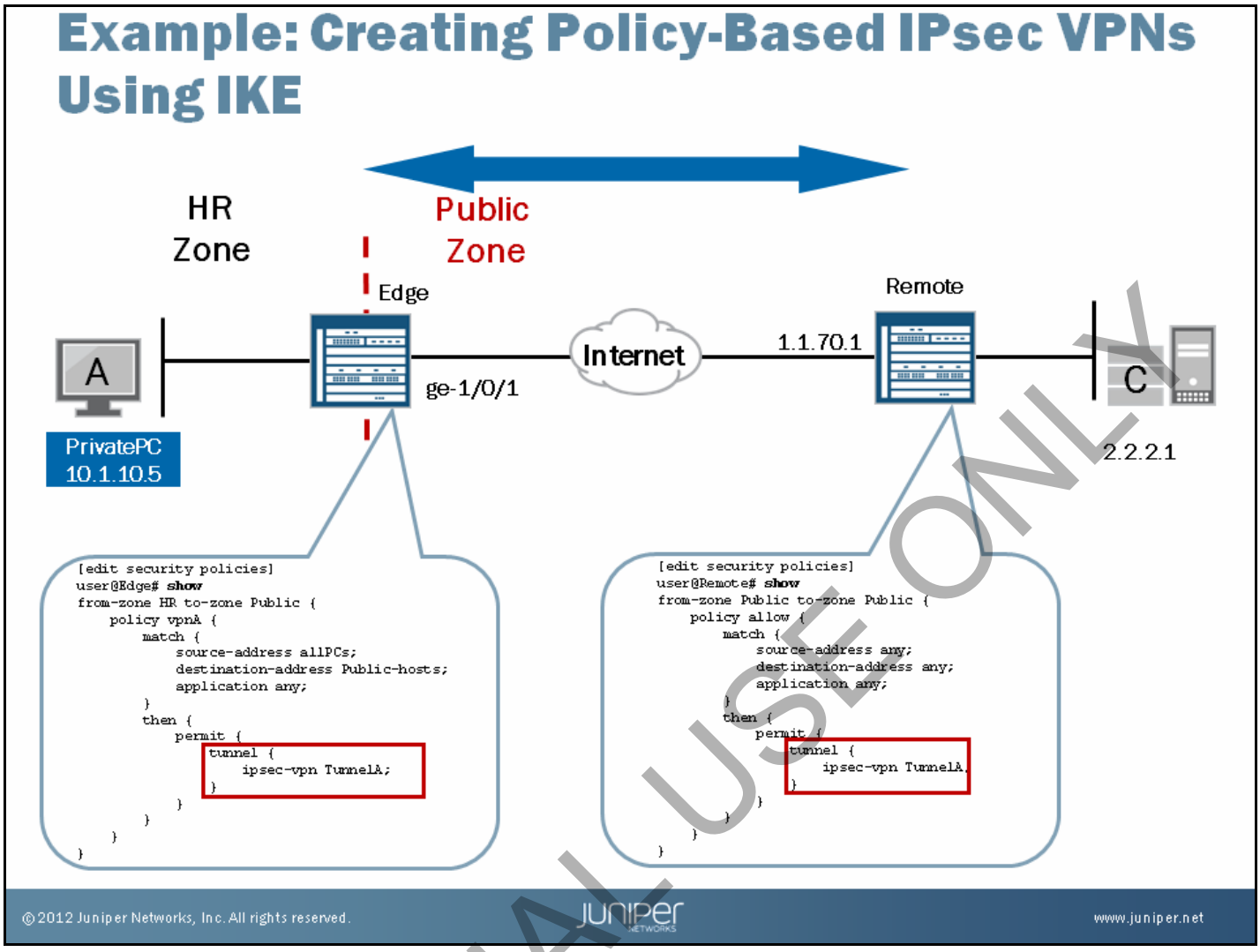
If you are implementing a policy-based IPsec VPN, you must apply the configured VPN from within the security policy, as illustrated on the slide.



Applying IPsec—Route-Based IPsec VPNs

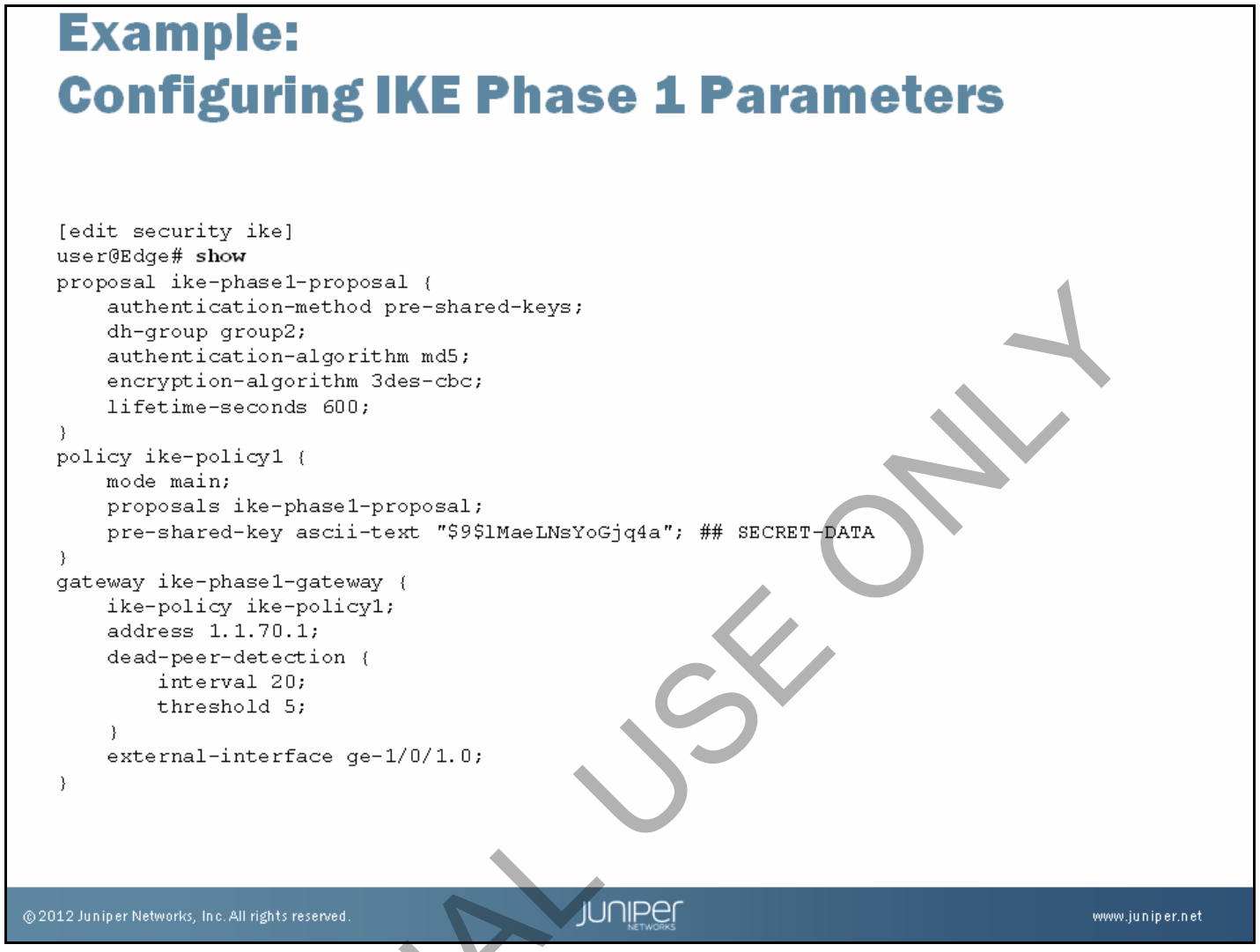
If you are implementing a route-based IPsec VPN, you must perform the following steps:

1. Configure the secure tunnel interface (st0.x);
2. Configure a static route or enable dynamic routing that points to the st0.x interface;
3. Add the st0.x interface to the appropriate security zone; and
4. Bind the st0.x interface to the IPsec VPN.



Example: Creating Policy-Based IPsec VPNs Using IKE

Consider the following example: you must implement a policy-based IPsec VPN between two SRX Series Services Gateways named *Edge* and *Remote*, as illustrated on the slide. A policy-based IPsec VPN implies that you must refer to the VPN tunnel from within the policies at each end, as demonstrated on the slide.

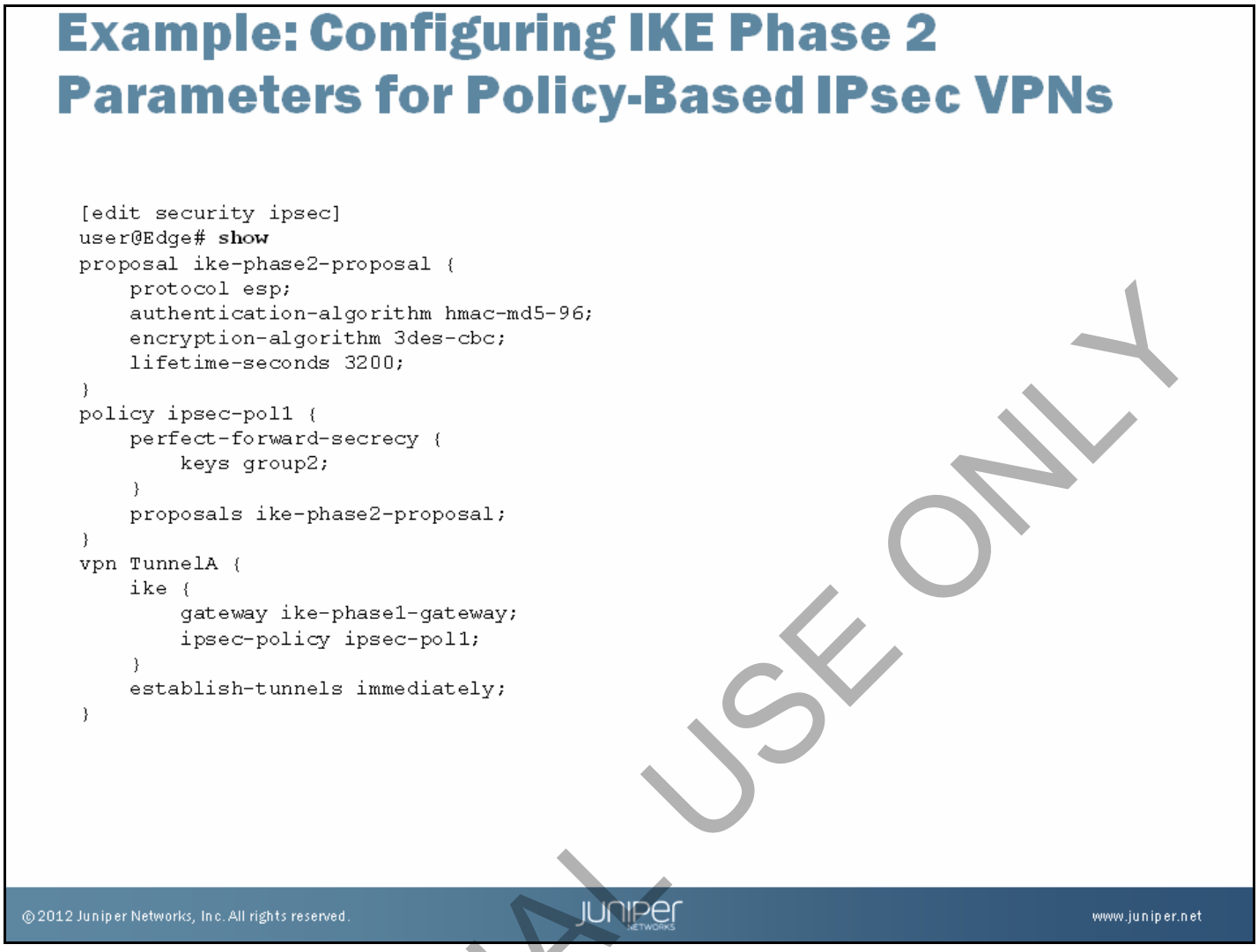


Example: Configuring IKE Phase 1 Parameters

The slide illustrates the configuration of the following parameters for IKE Phase 1:

1. *Proposal for IKE Phase 1*: Recall that this step is optional, because you can use the Junos predefined proposals (the choices are `basic`, `compatible`, or `standard`). On the slide, we named the configured proposal *`ike-phase1-proposal`*. We decided to use authentication algorithm `md5`, encryption algorithm `3des-cbc`, a DH key exchange of `group 2`, and preshared keys as the authentication method.
2. *A policy, called `ike-policy1`*: Within the policy we specified the mode that IKE Phase 1 will use—main mode, in this case. We referred to the proposal *`ike-phase1-proposal`*, and we specified the preshared key.
3. *Gateway, called `ike-phase1-gateway`*: Within the `gateway` stanza we referred to the policy *`ike-policy1`*, specified the address of peer *`Remote`* (1.1.70.1), and specified the external interface that IKE will use to establish the tunnel (`ge-1/0/1.0`). Also, we decided to use DPD so that a peer sends a DPD request packet to another peer if it does not hear from it for 20 seconds. Suppose it is *`Edge`* that sends the DPD request packet to *`Remote`*. After sending the DPD request packet, *`Edge`* considers *`Remote`* to be unavailable after five sequences of waiting for 20 seconds.

You must repeat the illustrated configuration on the *`Remote`* device, defining the appropriate external interface and gateway address.

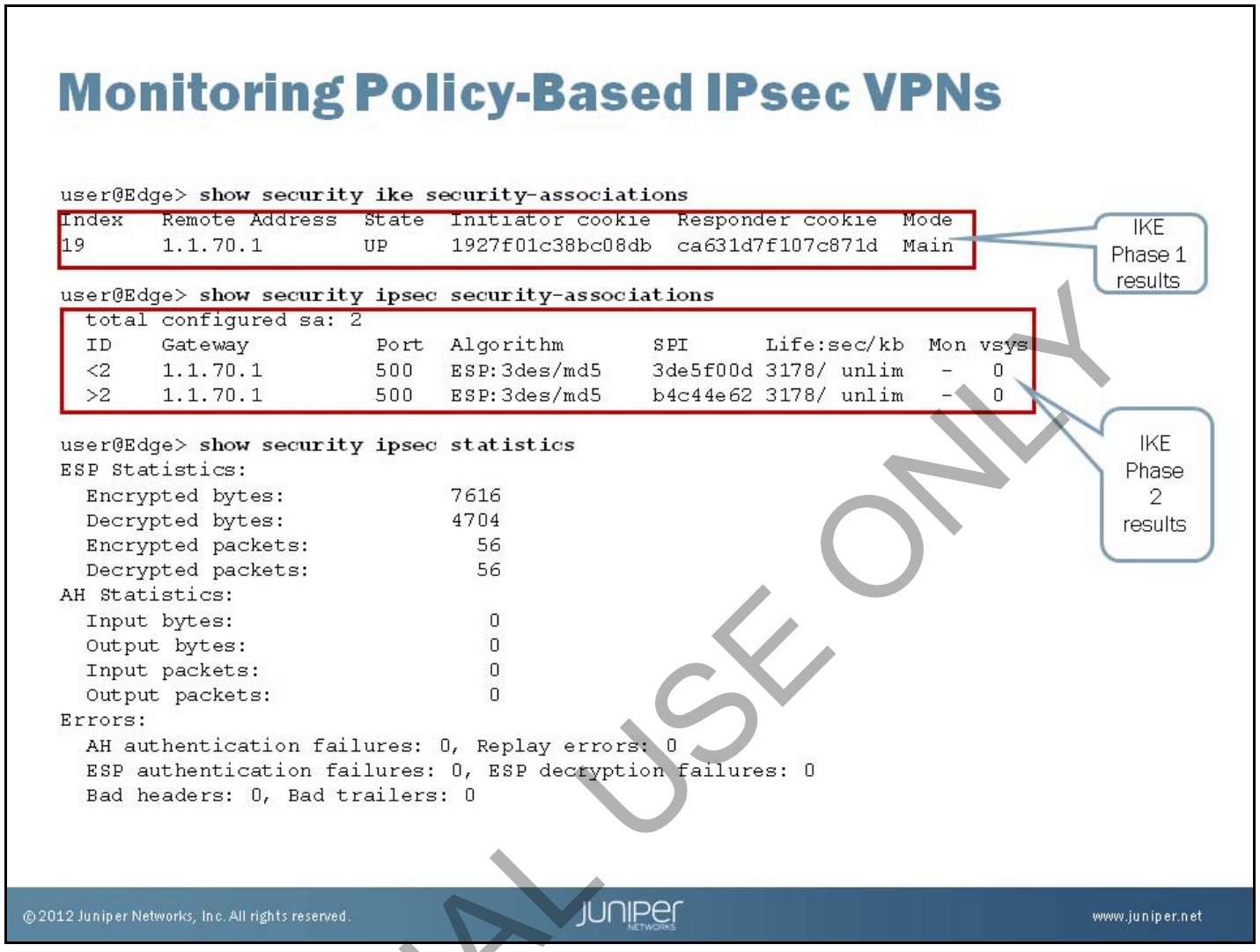


Example: Configuring IKE Phase 2 Parameters for Policy-Based IPsec VPNs

The slide illustrates configuration of IKE Phase 2 parameters for our example. Our configuration consists of the following:

1. *A proposal for IKE Phase 2*: Recall that this step is optional because you can use the Junos predefined proposals (the choices are `basic`, `compatible`, or `standard`). We named the configured proposal *`ike-phase2-proposal`*. We decided to use authentication algorithm `hmac-md5-96`, encryption algorithm `3des-cbc`, and the ESP protocol.
2. *A policy called `ipsec-pol1`*: Within the policy we referred to the proposal *`ike-phase2-proposal`*, and we specified that IPsec will use DH Group 2 as its PFS.
3. *A VPN tunnel, called `TunnelA`*: Within the tunnel we referred to the gateway *`ike-phase1-gateway`* and the IKE Phase 2 policy *`ipsec-pol1`*. We also specified that tunnels should establish immediately.

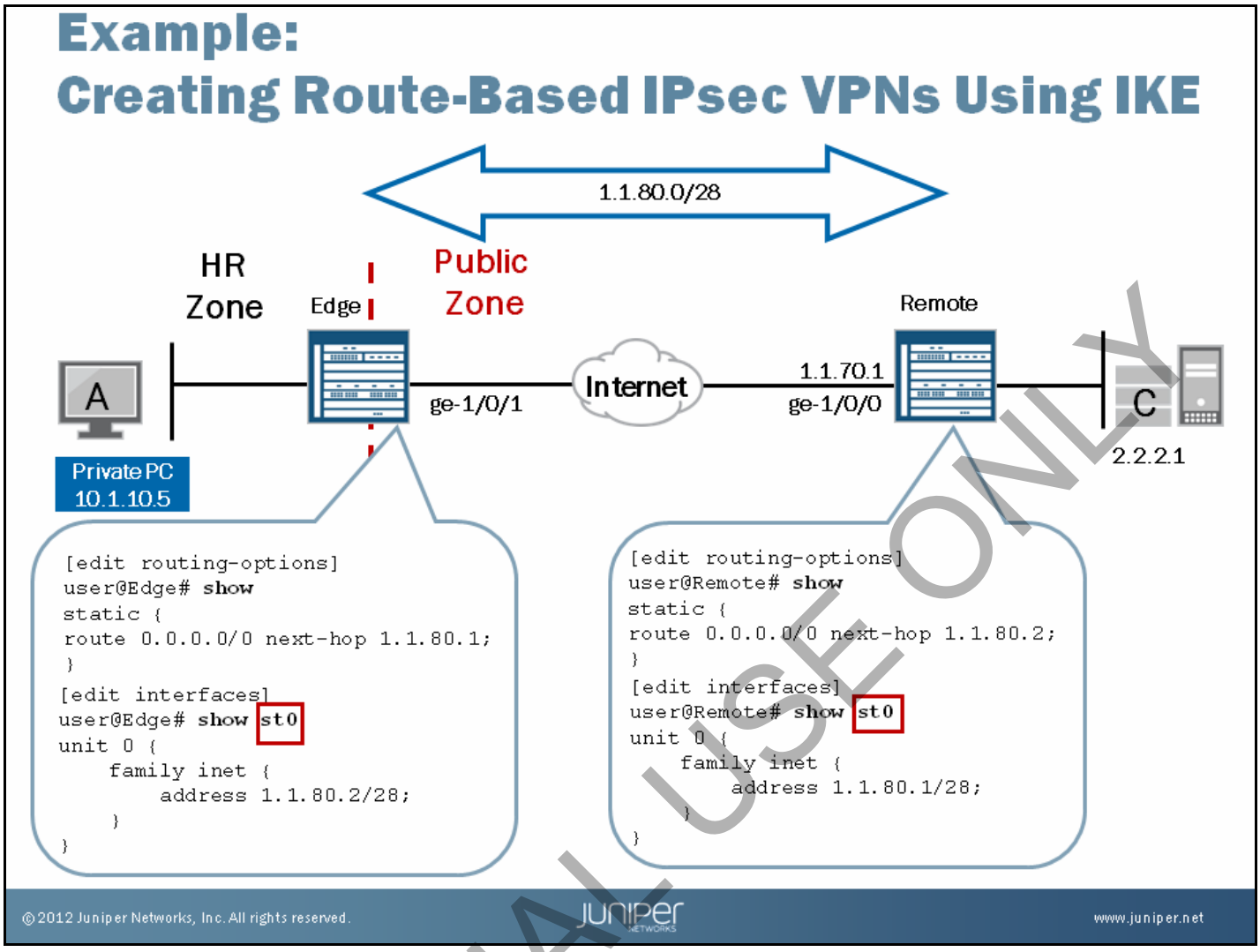
Note that you should repeat the configuration illustrated for the *`Edge`* device on the *`Remote`* device.



Monitoring Policy-Based IPsec VPNs

Once you finish and commit all the configurations, you must ensure that the tunnels are working properly by following the order of how IPsec works. First ensure that IKE Phase 1 is working properly, then ensure that IKE Phase 2 is working properly. To check that IKE Phase 1 functions properly, check whether the SAs are formed. Similarly, you perform IKE Phase 2 checking by viewing the resulting SAs.

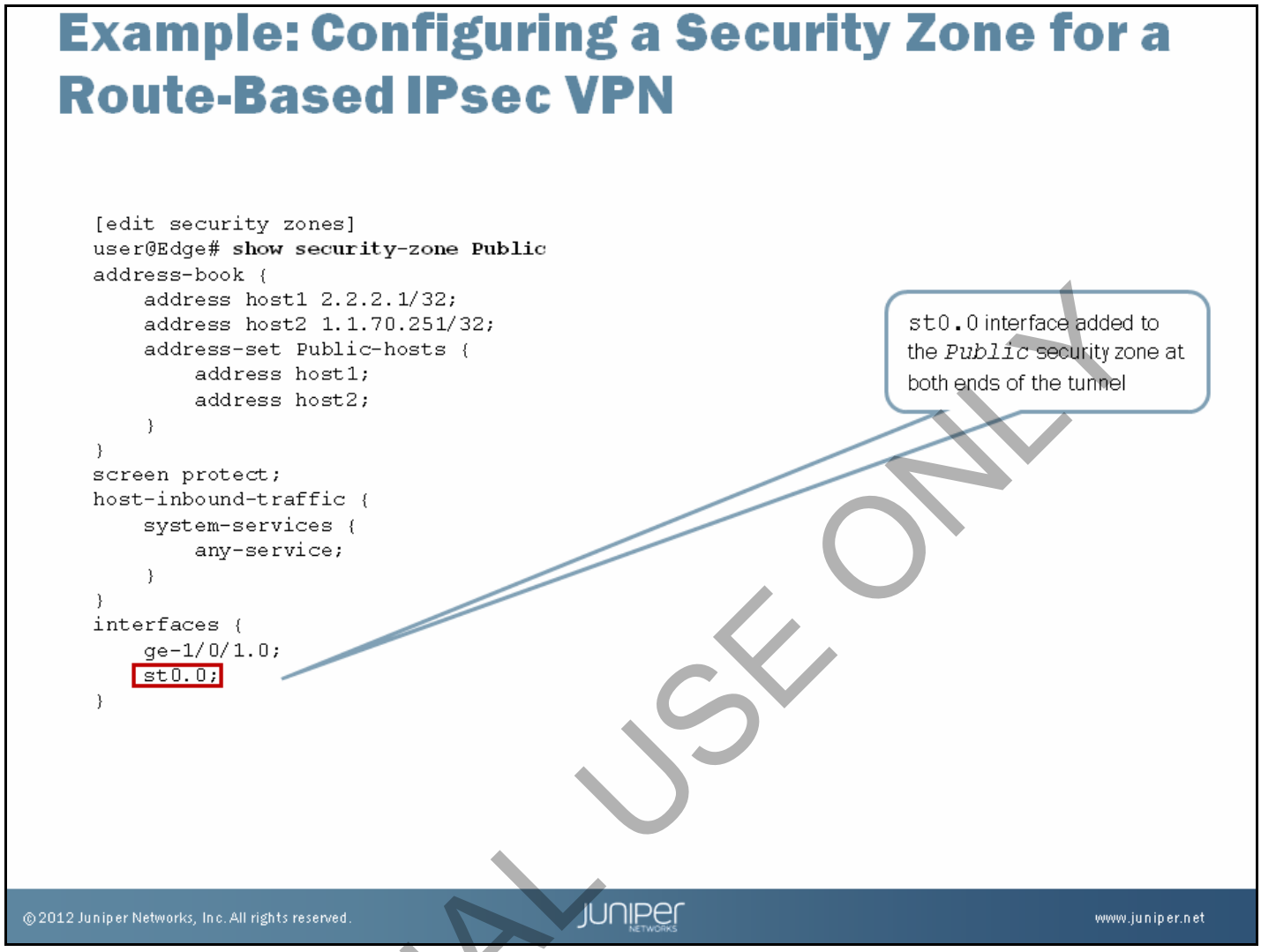
The slide illustrates both commands with the resulting outputs. Also, you can view the IPsec statistics that specify the number of transit packet bytes that the device has encrypted and decrypted.



Example: Creating Route-Based IPsec VPNs Using IKE

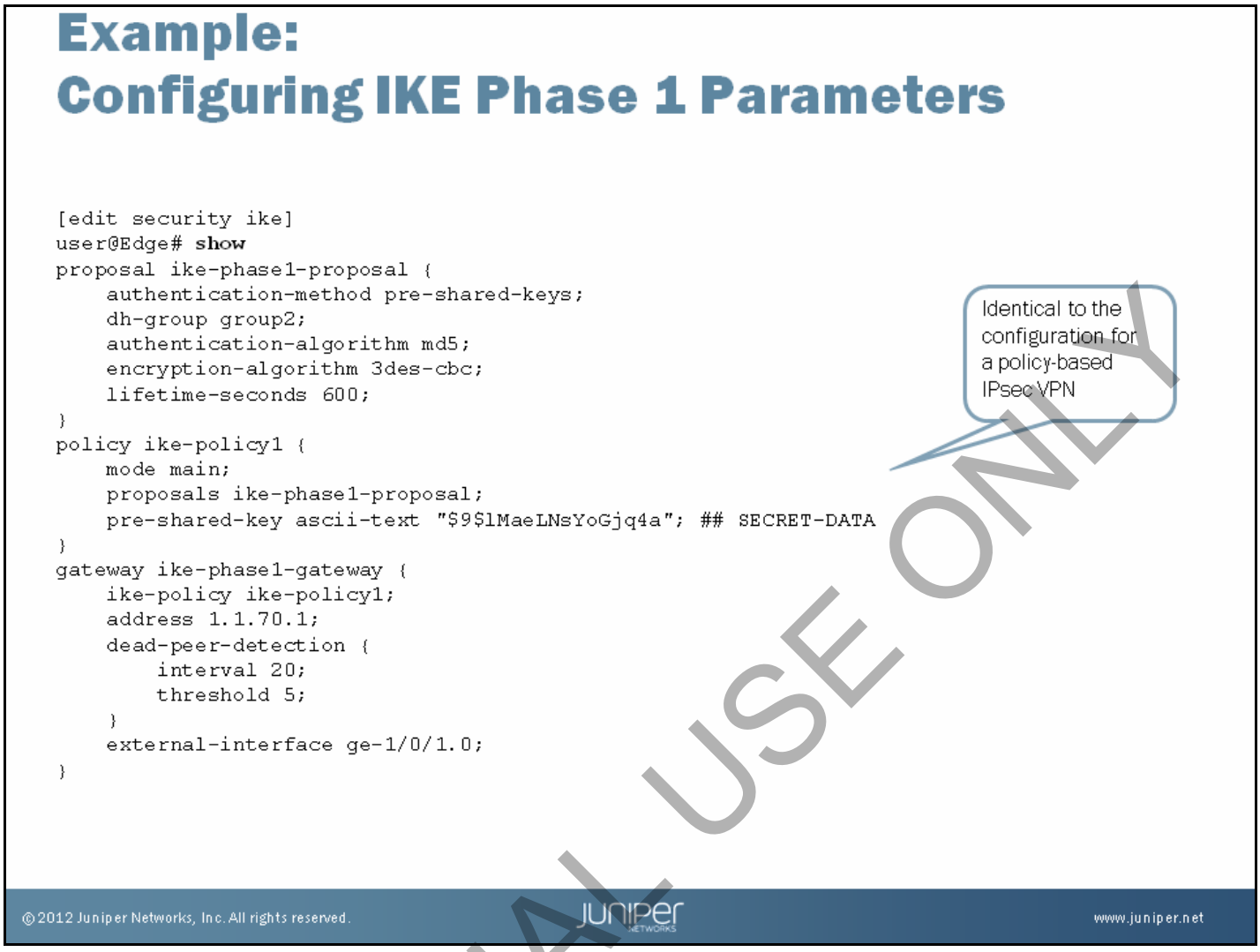
Consider another example: In this case you need to set up the IPsec tunnel using the route-based method and IKE. Recall that a route-based VPN requires only one tunnel between Junos security platforms, while a policy-based VPN sets up a tunnel for every new flow.

You must ensure that both ends of the VPN tunnel have a secure tunnel interface configured—in our case it is the st0.0 interface, with IP address 1.1.80.0/28. Furthermore, you must ensure that each of the devices has a valid route referring to the st0.0 interface. In our case we are using static route 0.0.0.0/0.



Example: Configuring a Security Zone for a Route-Based IPsec VPN

Once you configure interface st0.0, you must add it to the corresponding security zone. In our case, we must add it to the *Public* security zone. Also note that although the slide provides the configuration for the *Edge* device, you must also repeat it for the *Remote* device.

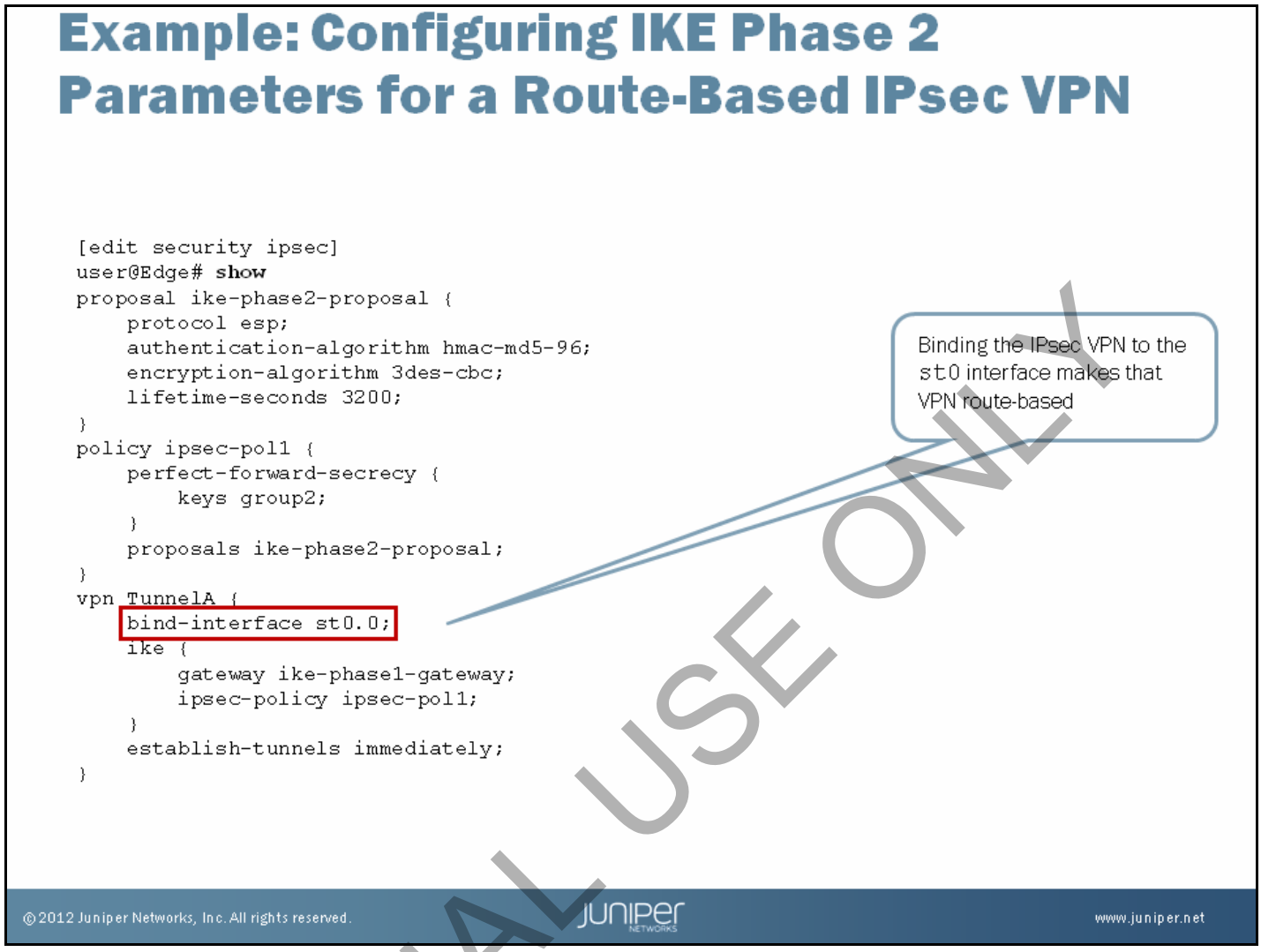


Example: Configuring IKE Phase 1 Parameters

The slide illustrates the configuration of the following parameters for IKE Phase 1:

1. *The proposal for IKE Phase 1*: Recall that this step is optional because you can use the Junos predefined proposals (the choices are `basic`, `compatible`, or `standard`). We named the configured proposal *`ike-phase1-proposal`*. We decided to use authentication algorithm `md5`, encryption algorithm `3des-cbc`, a DH key exchange of `group2`, and preshared keys as the authentication method.
2. *A policy, called `ike-policy1`*: Within the policy, we specified the mode that IKE Phase 1 will use—the main mode, in this case. We referred to the proposal *`ike-phase1-proposal`*, and we specified the preshared key.
3. *A gateway, called `ike-phase1-gateway`*: Within the `gateway` stanza we referred to the policy *`ike-policy1`*, specified the address of the peer *`Remote`* (1.1.70.1), and specified the external interface IKE will use to establish the tunnel (ge-1/0/1.0). Also, we decided to use DPD so that a peer will send a DPD request packet to another peer if it does not hear from it for 20 seconds. Suppose that it is *`Edge`* that sends the DPD request packet to *`Remote`*. After sending the DPD request packet, *`Edge`* considers *`Remote`* to be unavailable after five sequences of waiting for 20 seconds.

Note that you must repeat the illustrated configuration at the *`Remote`* device, defining the appropriate external interface and gateway address.



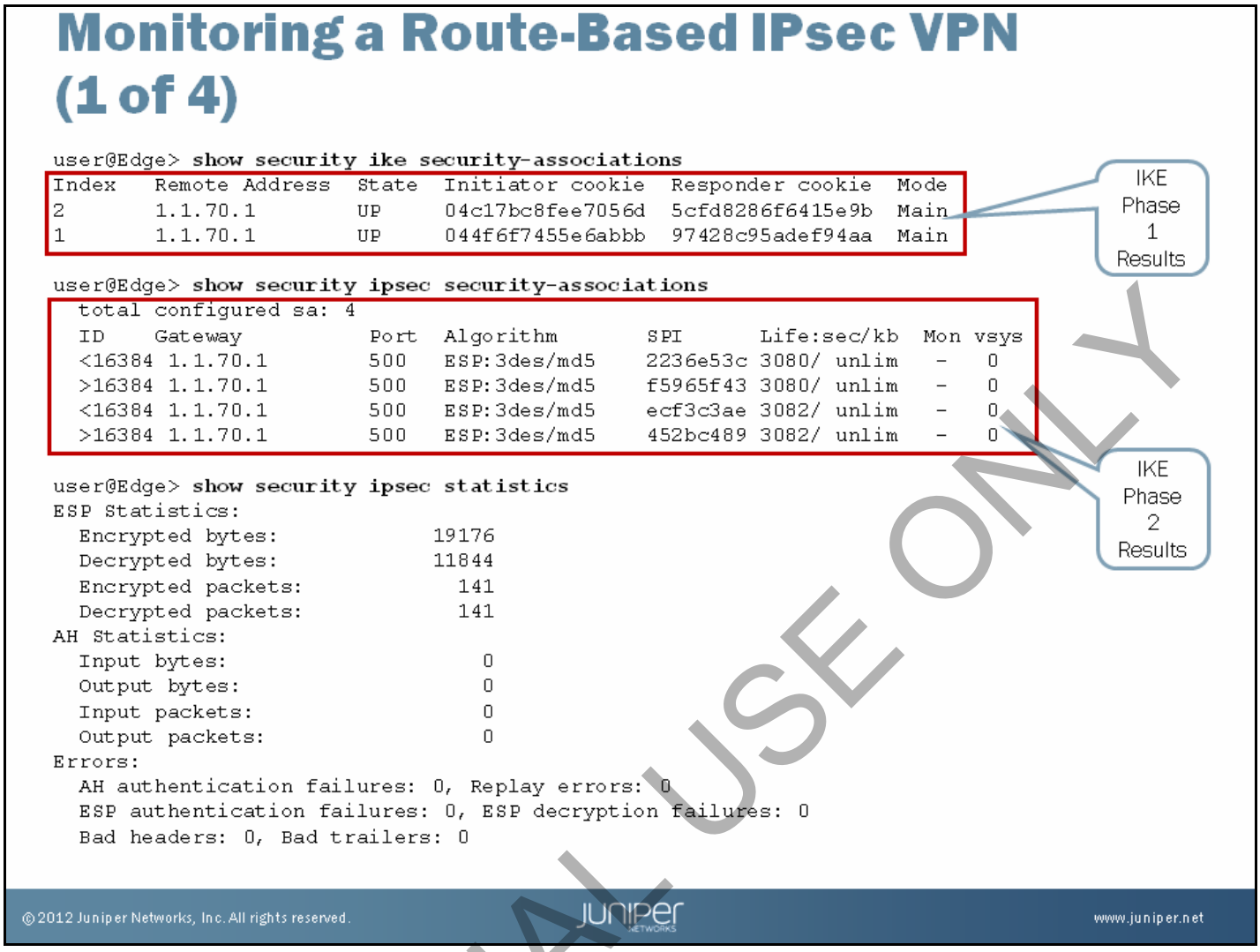
Example: Configuring IKE Phase 2 Parameters for a Route-Based IPsec VPN

The slide illustrates configuration of IKE Phase 2 parameters for our example. Our configuration consists of the following:

1. *A proposal for IKE Phase 2:* Recall that this step is optional, because you can use the Junos predefined proposals (the choices are `basic`, `compatible`, or `standard`). We named the configured proposal *`ike-phase2-proposal`*. We decided to use authentication algorithm `hmac-md5-96`, encryption algorithm `3des-cbc`, and the ESP protocol.
2. A policy, named *`ipsec-pol1`:* Within the policy we referred to the proposal *`ike-phase2-proposal`*, and we specified that IPsec will use DH `group2` as its PFS.
3. *A VPN tunnel, called `TunnelA`:* Within the tunnel we referred to the gateway *`ike-phase1-gateway`* and the IKE Phase 2 policy *`ipsec-pol1`*. We also specified that tunnels should establish immediately. Furthermore, we bound the st0.0 interface to the tunnel.

When you compare the configuration on the slide to the policy-based IPsec IKE Phase 2 configuration, you will notice that the only difference between the two is the statement binding interface st0.0 to the tunnel.

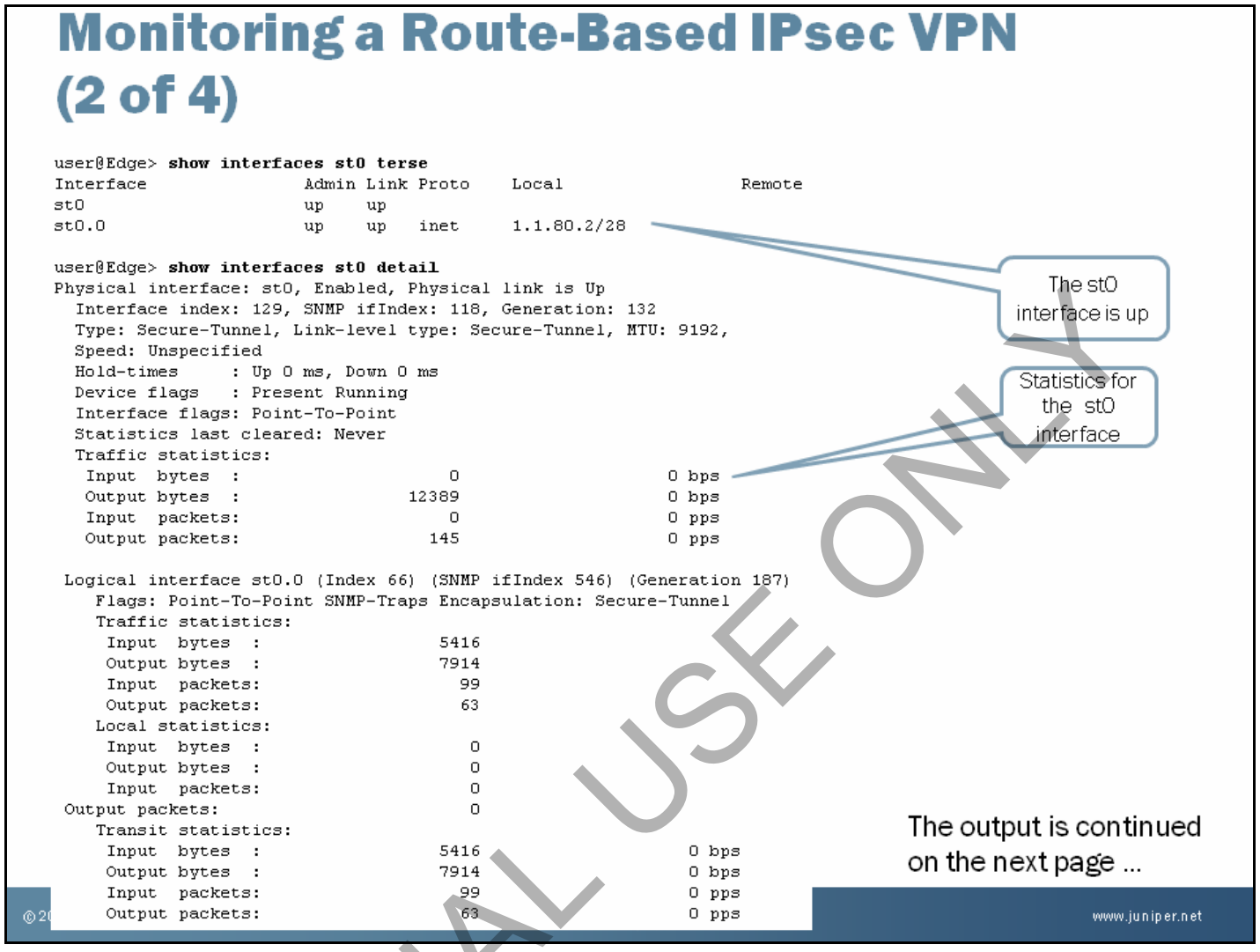
Note that we must also repeat the configuration illustrated for the *`Edge`* device at the *`Remote`* device.



Monitoring a Route-Based IPsec VPN: Part 1

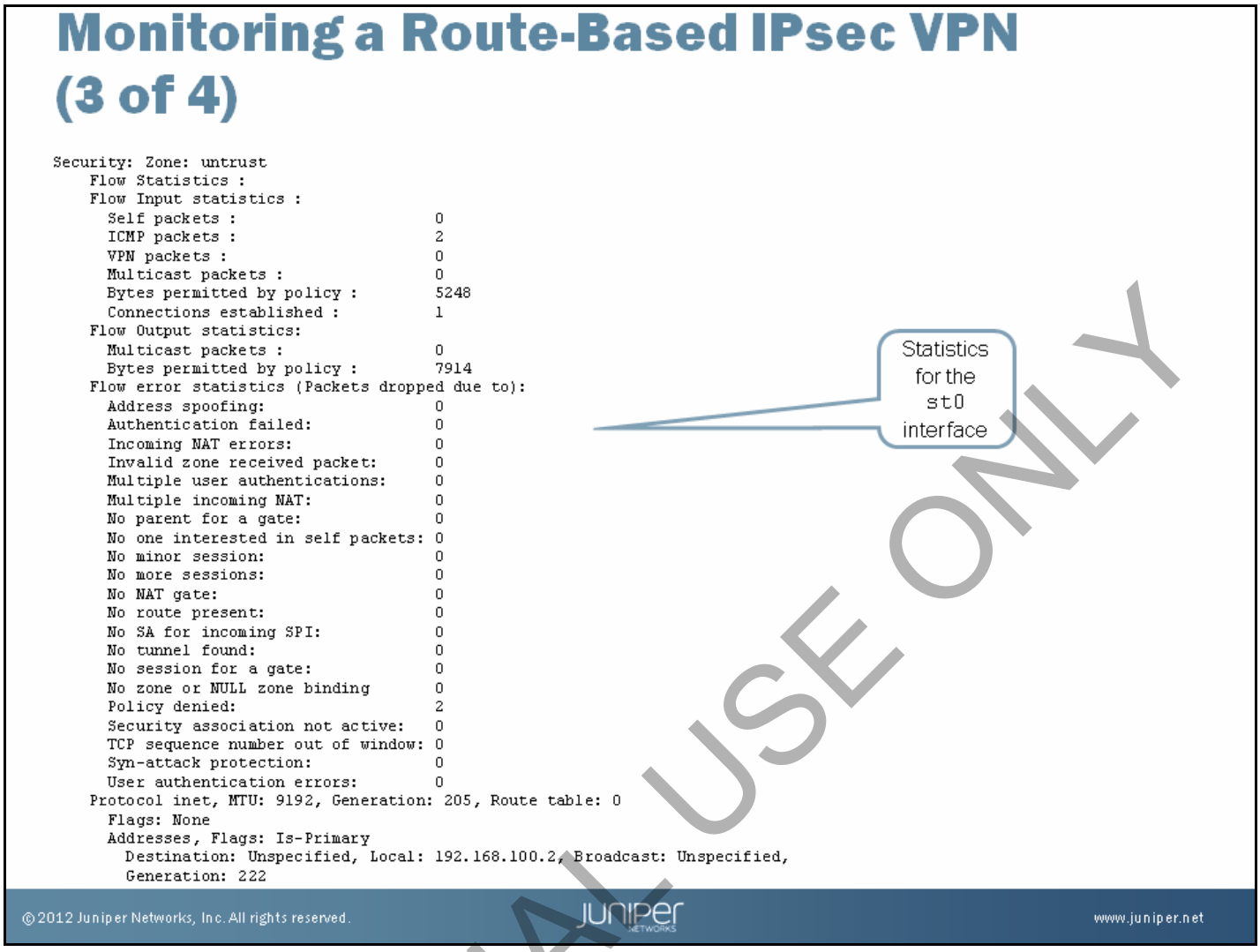
Once you finish and commit all the configurations, you must ensure that the tunnels work properly by following the order of how IPsec works. First ensure that IKE Phase 1 works properly, then ensure that IKE Phase 2 works properly. To check that IKE Phase 1 functions properly, you must check whether the SAs form. Similarly, you perform IKE Phase 2 checking by viewing the resulting SAs.

The slide illustrates both commands with the resulting outputs. Also, you can view IPsec statistics that specify the number of transit packet bytes that the device encrypts and decrypts.



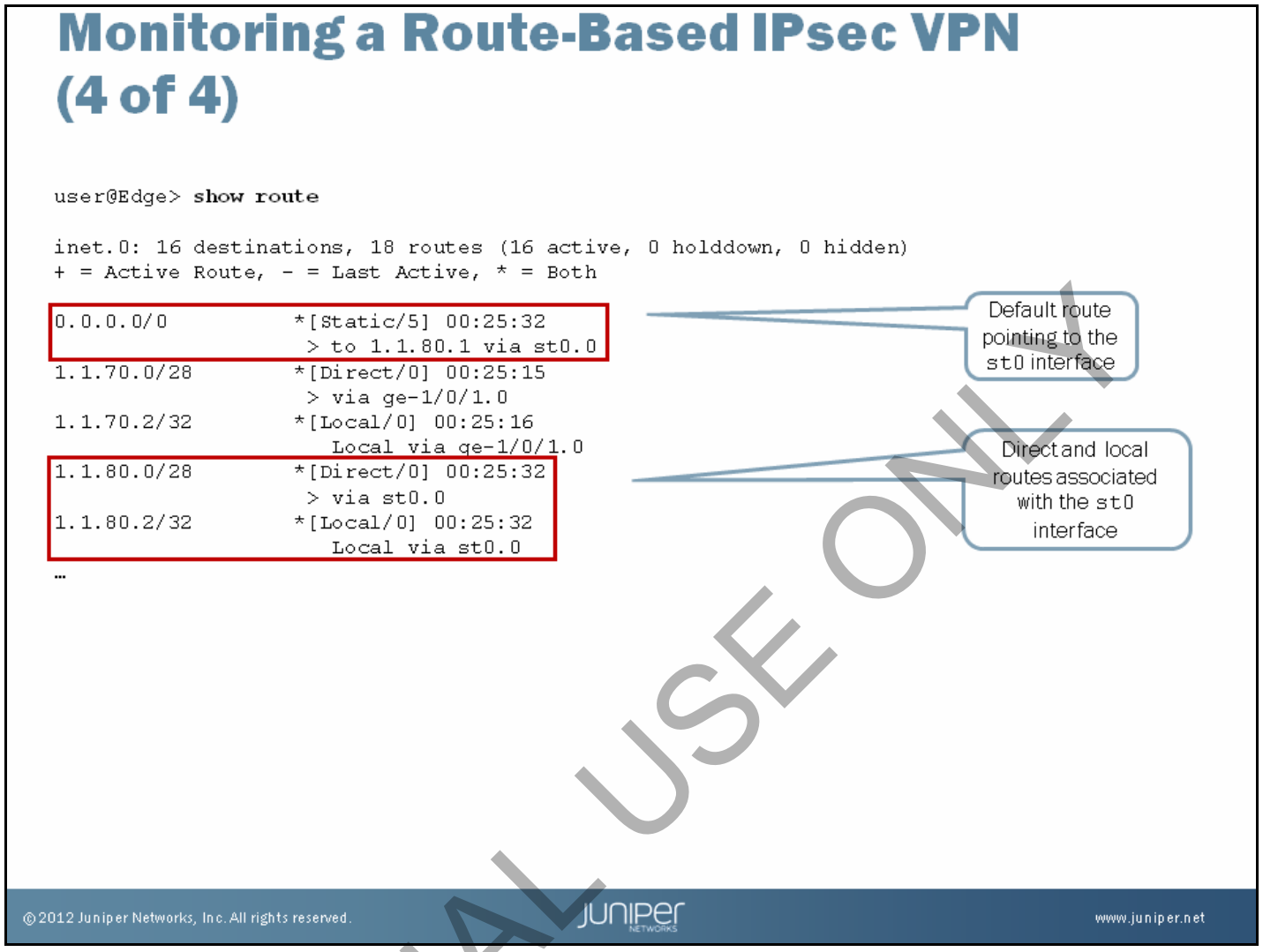
Monitoring a Route-Based IPsec VPN: Part 2

One of the differentiating points of a route-based IPsec VPN is that it uses the st0 interface. Therefore, you can use the **`show interfaces st0.x`** command to view whether the interface is up as well as how much information transits through it. If there is a problem in establishing the route-based IPsec tunnel, the st0 interface is not in the `up` state. The slide illustrates the results of the **`show interface st0 detail`** command for the *Edge* device.



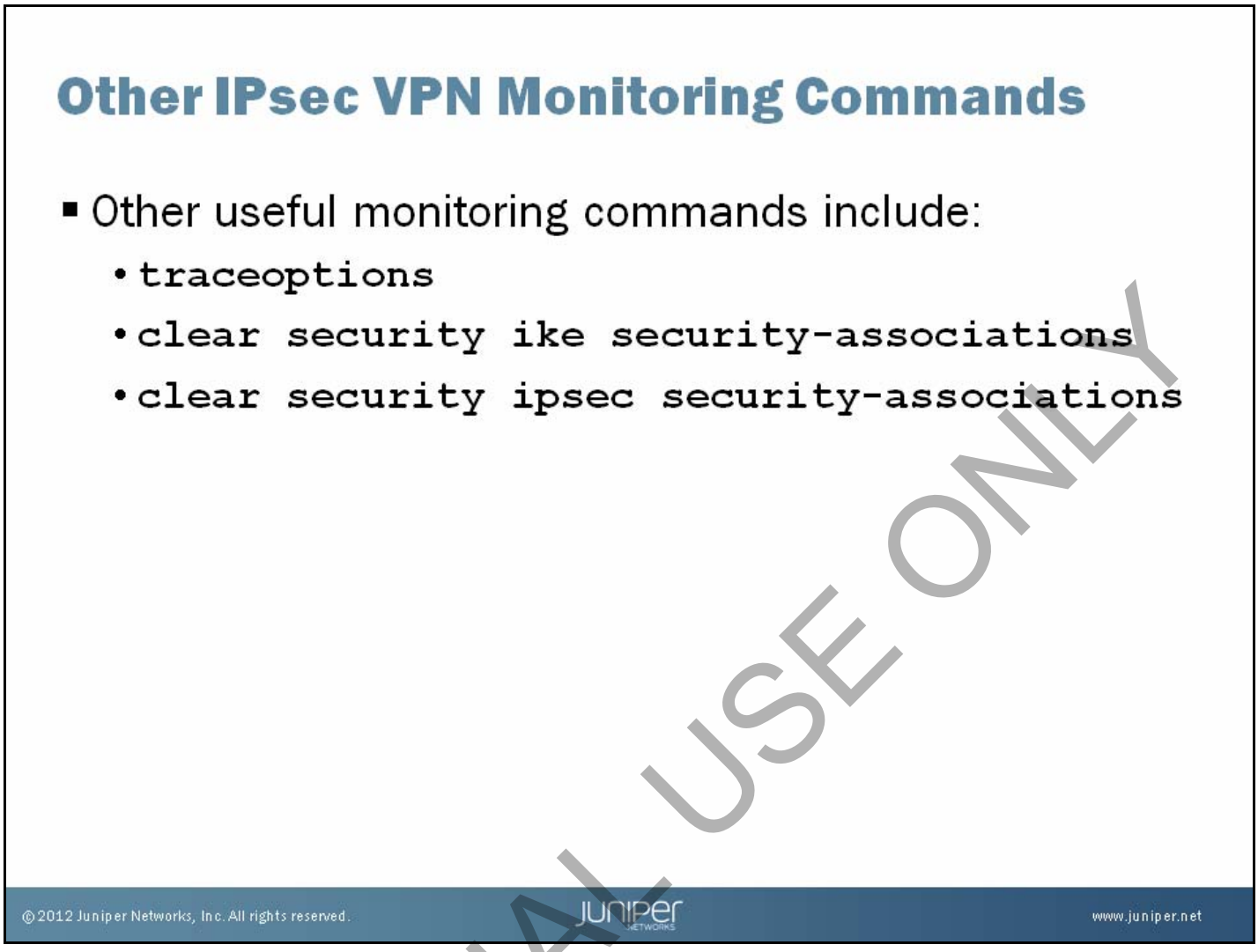
Monitoring a Route-Based IPsec VPN: Part 3

The slide is the continuation of the **`show interfaces st0 detail`** command from the previous slide. It provides statistical information for the st0 interface, including flow input, flow output, and flow error statistics.



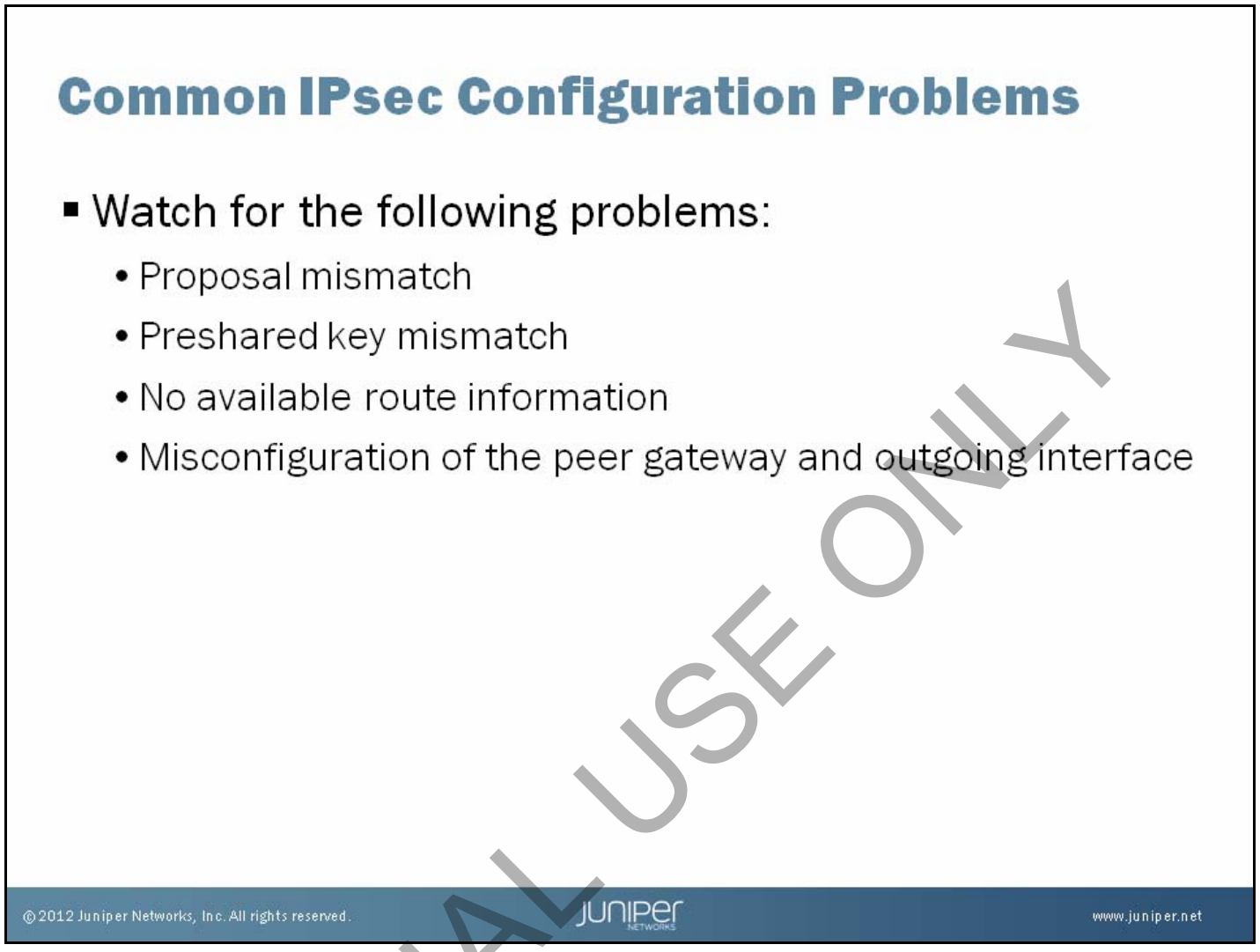
Monitoring a Route-Based IPsec VPN: Part 4

As we work with a route-based IPsec VPN, it is useful to check the routing table entries, ensuring that an active route referring to the st0 interface exists. In our case, the 0.0.0.0/0 default route, using interface st0.0 as its next hop, is active.



Other IPsec VPN Monitoring Commands

You can enable `traceoptions` to debug IKE Phase 1 and Phase 2. Also, you can clear IKE Phase 1 and Phase 2 SAs and statistics using the **`clear security ike security-associations`** and **`clear security ipsec security-associations`** operational commands.



Common IPsec Configuration Problems

You should be aware of the following common problems when configuring IPsec VPNs:

- *Proposal mismatch:* The IKE Phase 1 proposal lists configured on each side do not agree. In this case, the initiator of the tunnel sees retransmissions and a retransmission limit indicator. The problem is evident at the destination gateway (the responder). The responder rejects all proposals sent by the initiator.
- *Preshared key mismatch*: The keys do not match.
- *No route information is available*: To establish a gateway, you must either configure an explicit route or a default route (or use a dynamic routing protocol) to be used to reach the remote gateway.
- *The destination gateway is misconfigured*: It might happen that the destination gateway (responder) does not recognize the incoming request as originating with a valid peer gateway. Any of the following misconfigurations could cause this problem:
  - The peer gateway is not configured correctly;
  - The outgoing interface is not the right one; or
  - A proposal mismatch exists.