IPsec

real end-to-end security without VPNs

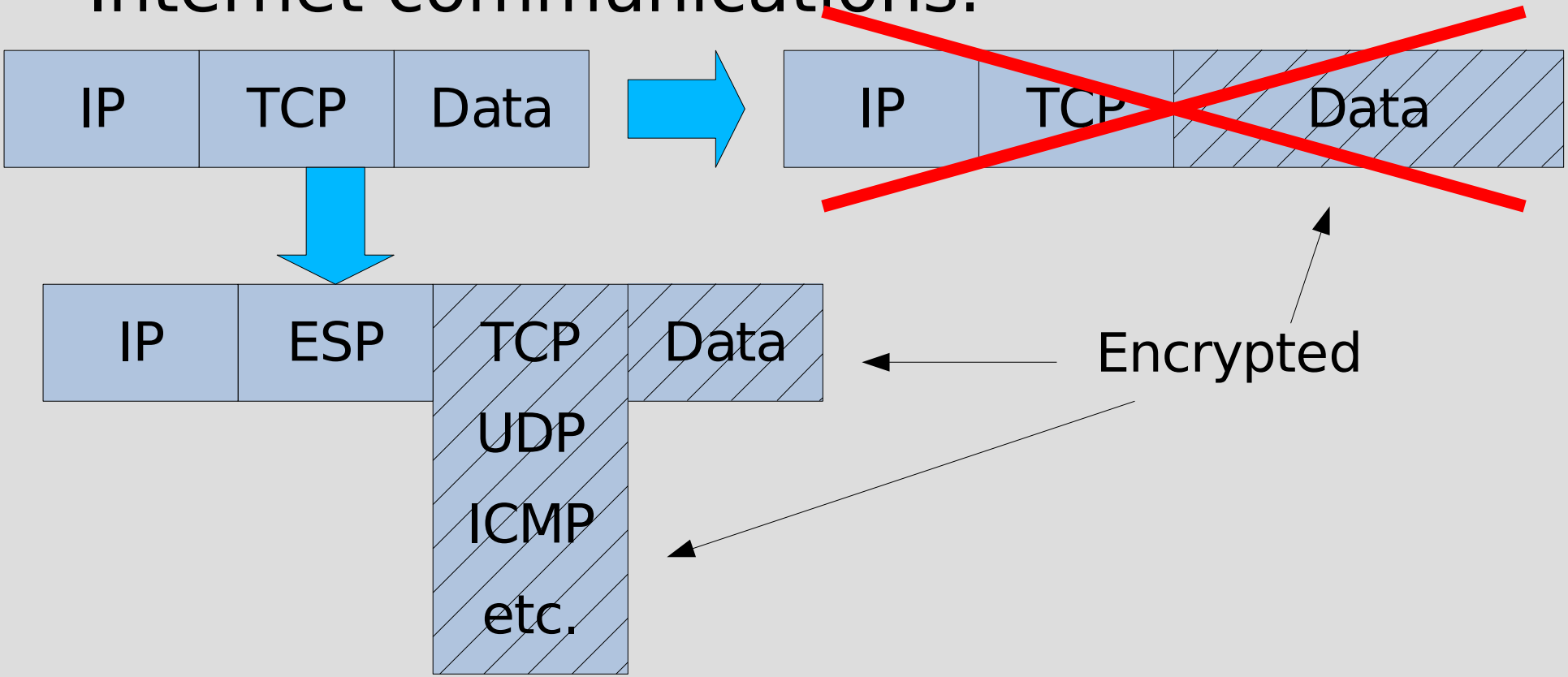
FUKT Computer Society
Teddy Hogeborn
Björn Påhlsson

Who are we?

- FUKT Computer Society
  - Unix system since 1995
  - Using IPsec since 2003
- FUKT is a society/club for computer enthusiasts
  - A computer system with many servers
  - Holds lectures ☺
  - Meet, experiment and tinker
  - A lot of (almost 100) members
  - Had some nice rooms until recently

What is IPsec?

A more basic and conceptually cleaner way to encrypt and authenticate Internet communications.



- IPsec is simpler on a conceptual level, like speaking Navajo* instead of using Enigma.
- It’s still hard to learn Navajo

* http://www.navajocodetalkers.org/

Is IPsec some special Linux thing?

No.

- First standard in 1995
- Third standard in 2005

- Supported for a long time
  - Unix
  - Mac OS X
  - Windows NT and later (2000, XP, etc.)
  - Most other operating systems

Why is IPsec not more widely used?

- Complex to configure
  - People prefer simpler methods
- Old key management standard “IKE” *very* complex
  - Phase 1/Phase 2/Main mode/Quick mode/ Aggressive mode/
  - 4 different methods, 8 ways to do it
  - New “IKEv2” standard is better
    - Not widely supported yet

What will be covered in this lecture?

- Why use IPsec?
- How IPsec works, in theory...
- ...and practice
  - (in Debian with Racoon and an early release of OpenIKEv2)

Why use IPsec?

- Why encryption/authentication?
- Single point of failure?
- What's wrong with using VPNs?
- Why not use an SSH tunnel?
- Why not use TLS?
- Monitoring and external firewalls?
- How is IPsec better?
  - Pros and cons of IPsec

Why encryption/ authentication?

- Encryption is important
  - Privacy from traffic sniffers
- Authentication is important
  - Protection from spoofing and man-in-the-middle attacks
- Examples:
  - Protect NFS, NIS, SMB...
  - Targeted security (lazy sysadmins)
    - Forced encryption for all applications
- Also on the local network!

Single point of failure?

- IPsec is only meant for
  - Encryption
  - Authentication of hosts
  - Replay protection
  - Traffic flow confidentiality
- *Not* meant for authorization or user authentication
  - Left to individual services and programs
- Extra security can easily be added
  - Beyond the scope of this talk

What's wrong with VPNs?

- Why a tunnel?
  - Two identical IP headers
- Does not protect individual hosts
- Creates stereotypical network design
  - Makes you fall into a design pattern which may not be appropriate
- End-to-end principle* states:
  - Intelligence at the end points
  - As little intelligence as possible in between
  - No host should *rely* on something else to protect it from "bad" traffic.

*http://www.reed.com/Papers/endtoend.pdf

Why not use SSH tunnels?

- SSH is a remote login system
  - SSH tunnels not a general tunneling mechanism
- Tunnels always alongside a login
  - Needs a remote user
- Authenticates *users,* not machines
- Only passwords and its own unique public key system

Why not use TLS?

(Used to be called SSL, as used in HTTPS)

- Only for TCP traffic
- Can only use X.509 certificates
- Support must exist in all applications
  - A lot of work to configure all of them
- Both client and server programs need access to their respective private keys
  - If a single host key is used, *all* server programs need access to it.

Brute force *works*, but…

- You *can* tunnel *anything* over *anything*
- TLS over TCP over IP over PPP over Telnet over TCP over IP…

- Rainbows are pretty, but not elegant
- …nor efficient

What about monitoring and firewalls?

- The goal of encryption is to *defeat* packet sniffing
  - Any encrypted traffic will bypass any external firewalls and can not be monitored
- IPsec potentially encrypts everything
  - The end hosts must secure themselves
- Some statistical analysis can still be done
  - IPsec has features to make even this difficult, since the person who makes the analysis might not be you

How is IPsec better?

- Can encrypt everything (TCP, UDP, ICMP, SCTP, etc.)
- Does not require application support
  - I now run Telnet every day
- Offloads key management to a separate application
  - Can most often use at least either pre-shared keys or X.509 certificates
  - Can be upgraded and replaced over time
- Does not need to be handled and managed – it’s just there, invisible

Downsides

- Need to be configured on both sides.
- Must be configured on the Operating System level
  - can not be changed or accessed by users
- Is CPU-bound – can be slow
  - Not compared to other encryption methods
    - Algorithms can be selected and tuned
  - *We* encrypt everything…
- Adds some size overhead to packets
  - Not much compared to other methods

Security Architecture for Control Networks using IPsec and KINK, Nobuo Okabe et al.
http://www.taca.jp/docs/saint2005/12_nobuo_okabe.pdf
Vincent Roy, Benchmarks for Native IPsec in the 2.6 Kernel, Linux Journal, October 2004
http://www.linuxjournal.com/article/7840

Can IPsec be of use in WLANs?

Yes, certainly.

- To IPsec, a WLAN is nothing special and is treated like any other network medium.
- WLANs, on the other hand, might need IPsec more than wire-based networks do, since WLANs are inherently less secure.
- If IPsec is used for *everything*, you could turn off WPA etc, since IPsec is better anyway.

- Won't give authority control
  - A Network Access Server with EAP-IKEv2 support can fix this

What about IPv6?

Yes, IPsec can do IPv6.

- IPsec is *designed* to be used with both IPv4 and IPv6.
- The IPv6 standards *mandates* IPsec support.

Why bother with IPsec if it is so complex?

- It *is* getting better all the time.
- Since you only need to configure it once per host to secure *all IP traffic*, it is worth the effort.
- You don't really need to understand all of it to get it to work.
  - We are proof of this. ☺

How *does* IPsec work?

- Much like a stateful firewall
- Security Policy Database (SPD)
  - Like a firewall rule list
- Security Association Database (SAD)
  - Like a stateful firewall's list of current connection states

Security Policy Database (SPD)

- Much like a firewall rule list
- Lists what packets should be encrypted or not
- Can specify by addresses, protocols and/or port numbers

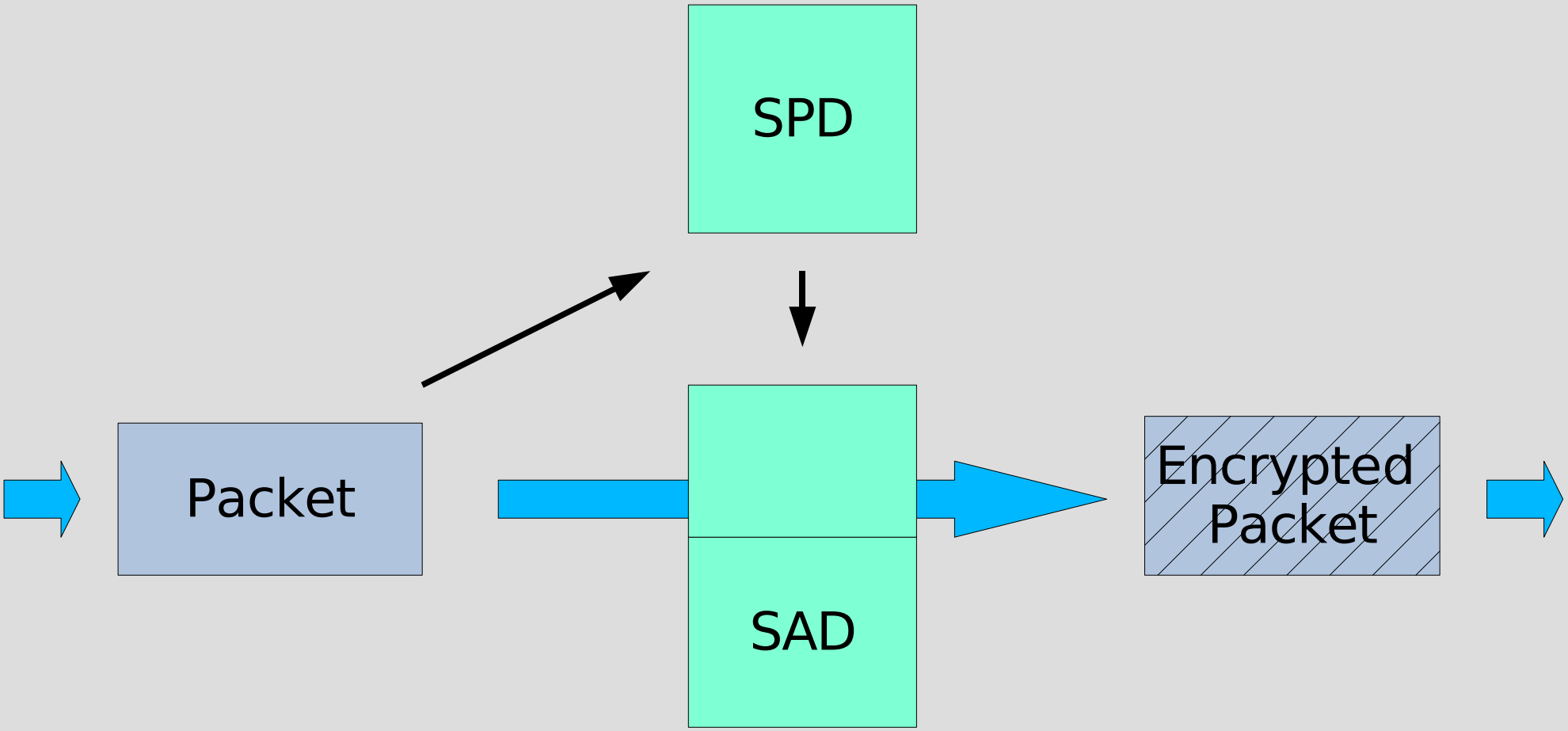
Security Association Database (SAD)

- Much like a stateful firewall's list of current connection states
- Each Security Association (SA) has:
  - Encryption and hash algorithm
  - Session keys
  - Maximum lifetime and byte counts
- An SA *must* exist for packets to be encrypted. The SPD rules does nothing but indicate the need for an SA.

Sequence of events
(first outgoing packet)

- An outgoing packet is created locally
- The SPD is searched for a match
- An SA is created to match the parameters specified in the SPD
  - Any existing matching SA is reused
- Packet is encrypted, signed, padded, etc. according to the SA settings
- Packet is sent

Sequence of events
(first outgoing packet)



SPD Entry Found

- SA *not* created by the Kernel
  - only holds SPD and already created SAs
- A Key management daemon is signaled
  - Daemon uses IKE or IKEv2 protocol for key negotiation
    - UDP port 500
- Common Daemons are Racoon, ISAKMPd, Pluto, ...

A working connection

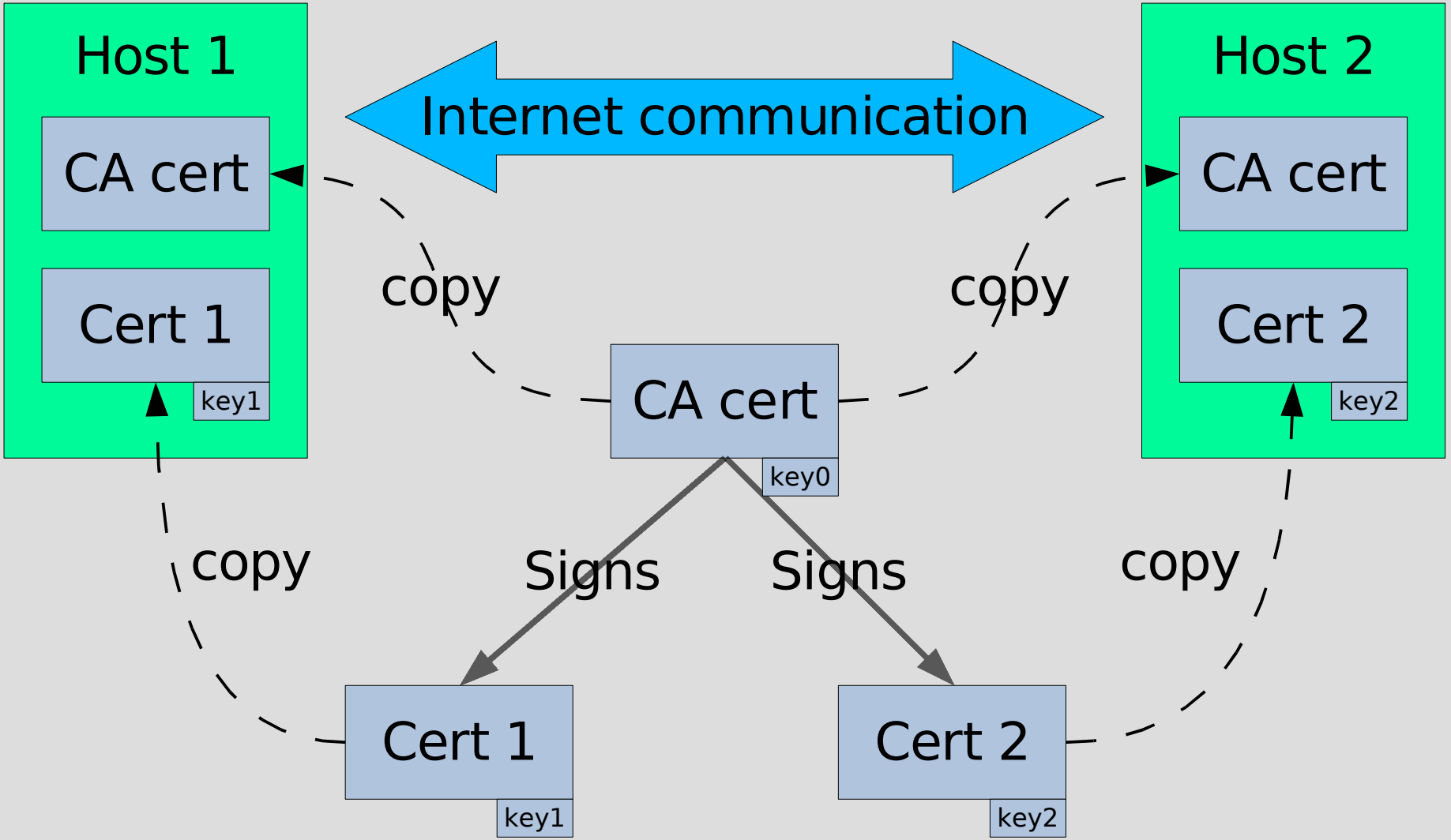
- Two SPD and SA entries on each host
  - one for outgoing, one for incoming
- Similarly configured key management daemons
  - Need to agree to use same algorithms, authentication method, etc.
- Negotiation phase need to be setup
  - can be same as for the SA
- Need access to authentication data
  - PSK, Private certificate

Practical examples

- Have:
  - Two Debian computers
  - One certificate for the *Certificate Authority*, and one signed certificate for each host
  - Working network between
    - On same local network or using the global Internet to a different continent does not matter – any connection works
- Want:
  - End to end IPsec
    - Transport mode, not Tunnel mode (VPN)

X.509 Certificates?

(*very* briefly)



X.509 Certificate programs

- These are graphical programs to manage X.509 certificates:
  - TinyCA
    - http://tinyca.sm-zone.net/
  - XCA
    - http://www.hohnstaedt.de/xca.html

X.509 is a nightmare itself

- If you don't believe us:
  - *Everything you Never Wanted to Know about PKI but were Forced to Find Out*
    - http://www.cs.auckland.ac.nz/~pgut001/pubs/pkitutorial.pdf
  - *Generating X.509 Certificates*
    - From "*The official IPsec Howto for Linux*"
      - http://www.ipsec-howto.org/x595.html
  - *The Open–source PKI Book*
    - http://ospkibook.sourceforge.net/docs/OSPKI-2.4.7/OSPKI-html/ospki-book.htm

*(PKI = Public Key Infrastructure)

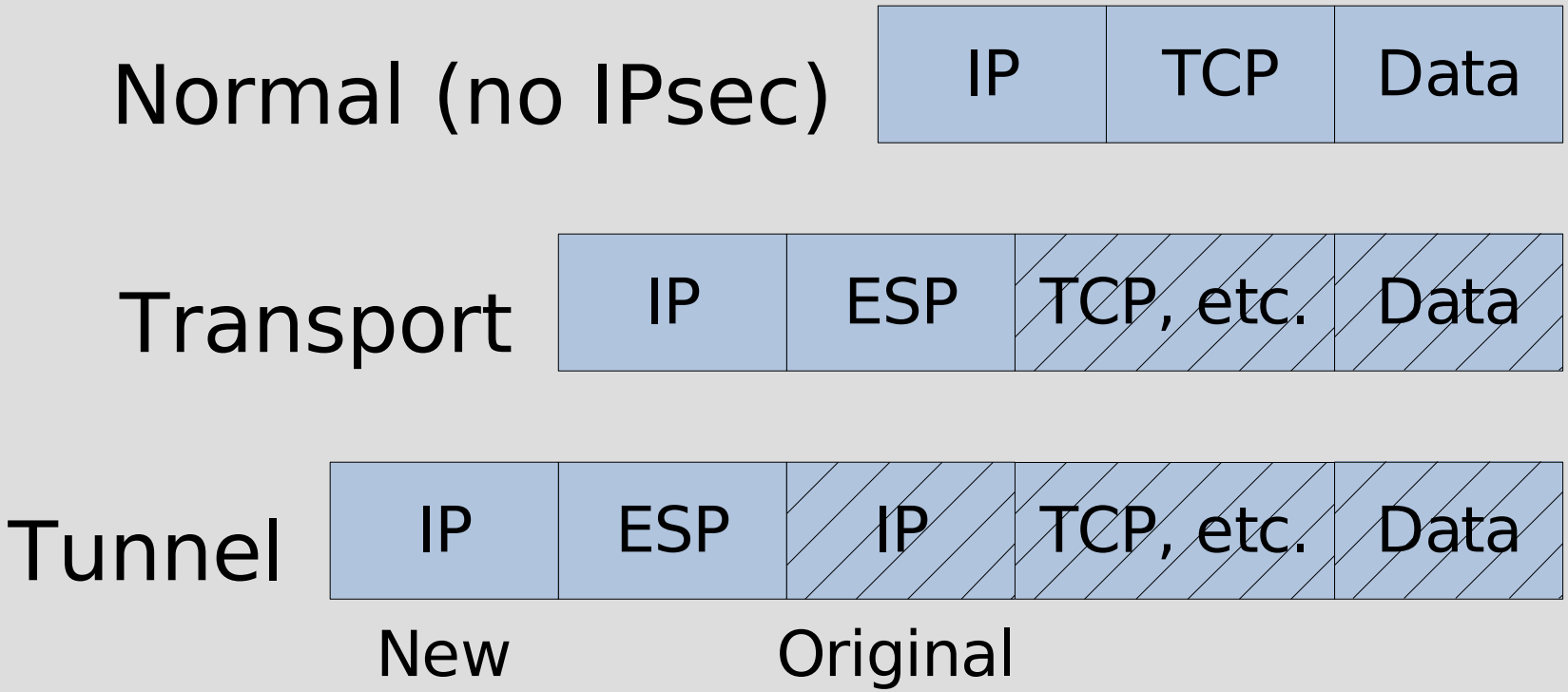
Recap

- IPsec encrypts and authenticates
- Rule list called SPD, Security Policy Database
- An ongoing connection is an SA, a Security Association
  - Contains session key
- An SA is most often created by a key negotiation daemon
  - Typically uses IKE or IKEv2 to negotiate
- Kept in kernel

Odds and ends

- IPsec packets use the ESP protocol
  - Will show up as “ESP” in a sniffer, not TCP or UDP, etc.
- IPsec also has an “AH” protocol.
  - Don’t use it; unnecessary complexity
- IPsec has a VPN feature called “Tunnel mode”
  - Don’t use tunnel mode if you don’t need it
  - We are using “transport mode”
- Minor complications with startup

Transport & Tunnel mode



Racoon – an IKE daemon

- Install
  - aptitude install racoon
- Configure
  - SPD
    - /etc/ipsec-tools.conf
  - Racoon
    - /etc/racoon/racoon.conf
- Testing to see if IPsec works
  - setkey -PD (SPD)
  - setkey -D (SAD)

/etc/ipsec-tools.conf

```
spdadd 193.11.177.97 193.11.177.88
  any -P out ipsec
  esp/transport//require;

spdadd 193.11.177.88 193.11.177.97
  any -P in ipsec
  esp/transport//require;
```

/etc/racoon/racoon.conf

```
remote 193.11.177.88 {
        exchange_mode main;
        certificate_type x509 "tharkun-cert.pem" "tharkun-key.pem";
        my_identifier asn1dn;
        verify_identifier on;
        peers_identifier asn1dn "C=SE, ST=Blekinge, L=Karlskrona, O=FUKT
   Computer Society, CN=murvel.fukt.bsnet.se";
        proposal {
                encryption_algorithm 3des;
                hash_algorithm sha1;
                authentication_method rsasig;
                dh_group modp2048;
        }
}
sainfo address 193.11.177.97 any address 193.11.177.88 any {
        pfs_group modp2048;
        encryption_algorithm aes;
        authentication_algorithm hmac_sha1;
        compression_algorithm deflate;
}
```

Say what?

- IKE *is* complex
- IKE key management daemons are written by people who could understand the first standards
- Racoon is one of the easier ones!

IKEv2 to the rescue

- One method with 2 steps instead of 8 different possible methods in IKEv1.
  - IKE_INIT, IKE_AUTH
- Some denial of service protection

OpenIKEv2 – an IKEv2 daemon

- Working example of IKEv2 (still in testing)
- Configuration file with clear examples and sections for Peer, My ID, etc.
- No Debian package yet ☹
- Version 0.93
- http://openikev2.sourceforge.net/
- Compile and install manually
  - Left as an exercise for the reader

/etc/openikev2/openikev2.conf (policy section)

```
policies{
    policy{
        src_selector = 193.11.177.97/32
        dst_selector = 193.11.177.88/32
        ipsec_proto = esp
    }
}
#OpenIKEv2 creates SPD entries
    #This section replaces /etc/ipsec-tools.conf
    #Any protocol, any port, transport mode
    #Creates two SPD entries, one in each
      direction
```

/etc/openikev2/openikev2.conf (peer section)

```
peer{
    peer_id{
        id_type = ipaddr
        id = 193.11.177.88
    }
    peer_id{
        id_type = der_asn1_dn
        id = "/etc/openikev2/certs/murvel-cert.pem"
    }
...
```

/etc/openikev2/openikev2.conf
(peer/ike section)

```
...
        ike{
                my_id{
                        id_type = der_asn1_dn
                        id = "/etc/openikev2/certs/tharkun.crt"
                }
                proposal{
                        encr = {aes256, aes192, aes128, 3des}
                        integ = {hmac_sha1}
                        prf = {sha1}
                        dh = {2, 1}
                }
                reauth_time = 600
                authentication_method = cert
                my_certificates = { "/etc/openikev2/certs/tharkun" }
                        # This will read the files “tharkun.key” and “tharkun.crt”
                ca_certificates = { "/etc/openikev2/certs/fukt" }
                        # This will read the file “fukt.crt”
        }

...
```

/etc/openikev2/openikev2.conf
(peer/ipsec section)

```
    ipsec{
        esp_proposal{
            encr = {aes256, aes192, aes128, 3des}
            integ = {hmac_sha1}
        }
        lifetime_soft = 500
        lifetime_hard = 800
        max_bytes_soft = 1000000
        max_bytes_hard = 1200000
    }
}
# End of “peer” section
# End of file
```

END OF LINE