

Neue strongSwan VPN Features

GUUG Frühjahrsfachgespräch 2015 Stuttgart

Prof. Dr. Andreas Steffen
Institute for Internet Technologies and Applications
HSR Hochschule für Technik Rapperswil
andreas.steffen@strongswan.org



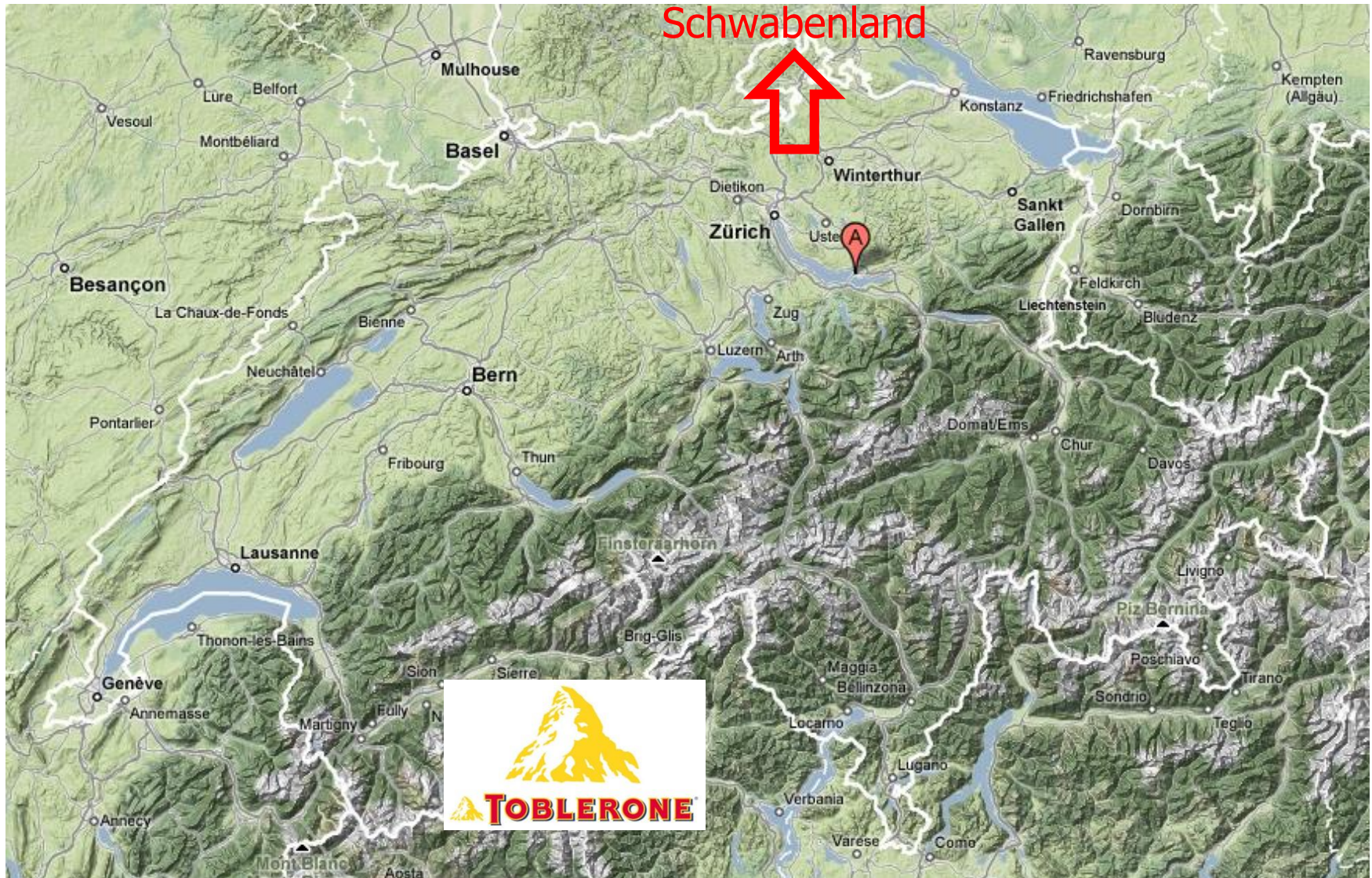
HSR

HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

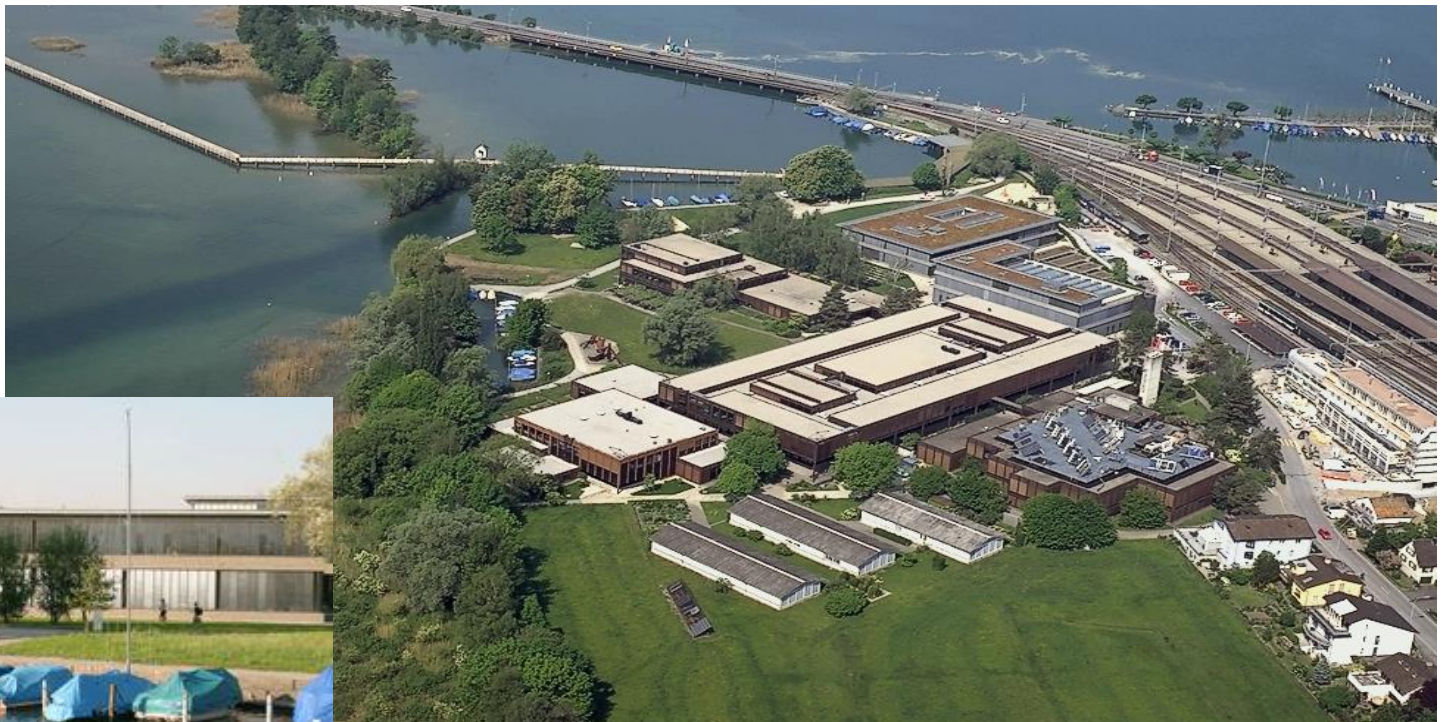


Wo um Gottes Willen liegt Rapperswil?



HSR - Hochschule für Technik Rapperswil

- Fachhochschule mit ca. 1500 Studierenden
- Studiengang Informatik (300-400 Studierende)
- Bachelor-Studium (3 Jahre), Master-Studium (+1.5 Jahre)



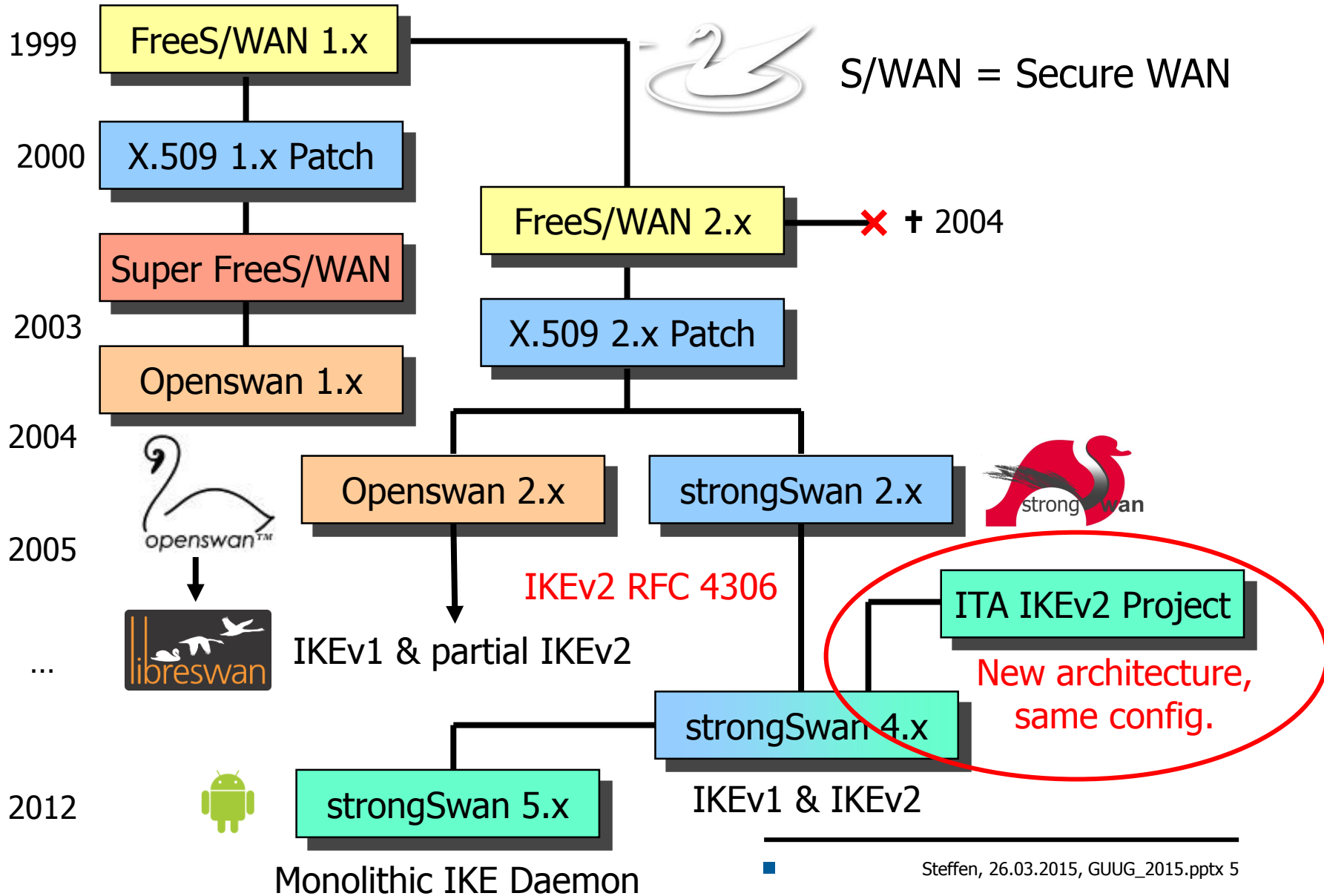
Neue strongSwan VPN Features

GUUG Frühjahrsfachgespräch 2015 Stuttgart

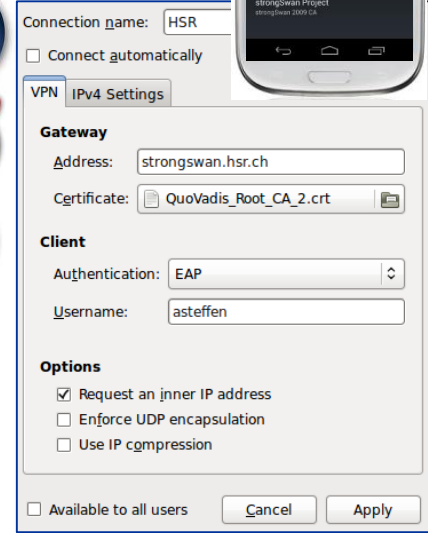
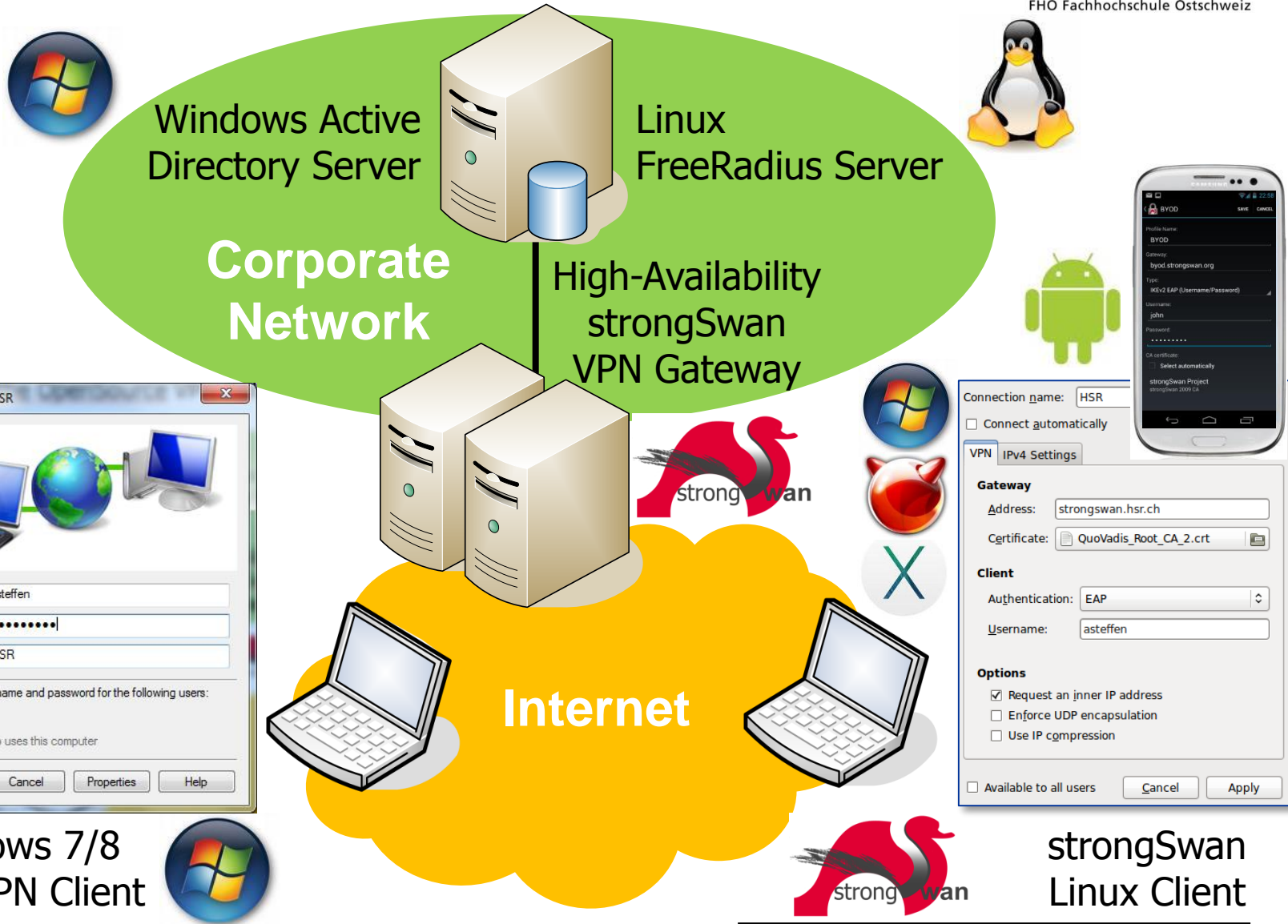
Warum und wozu ein starker Schwan?



The strongSwan Open Source VPN Project

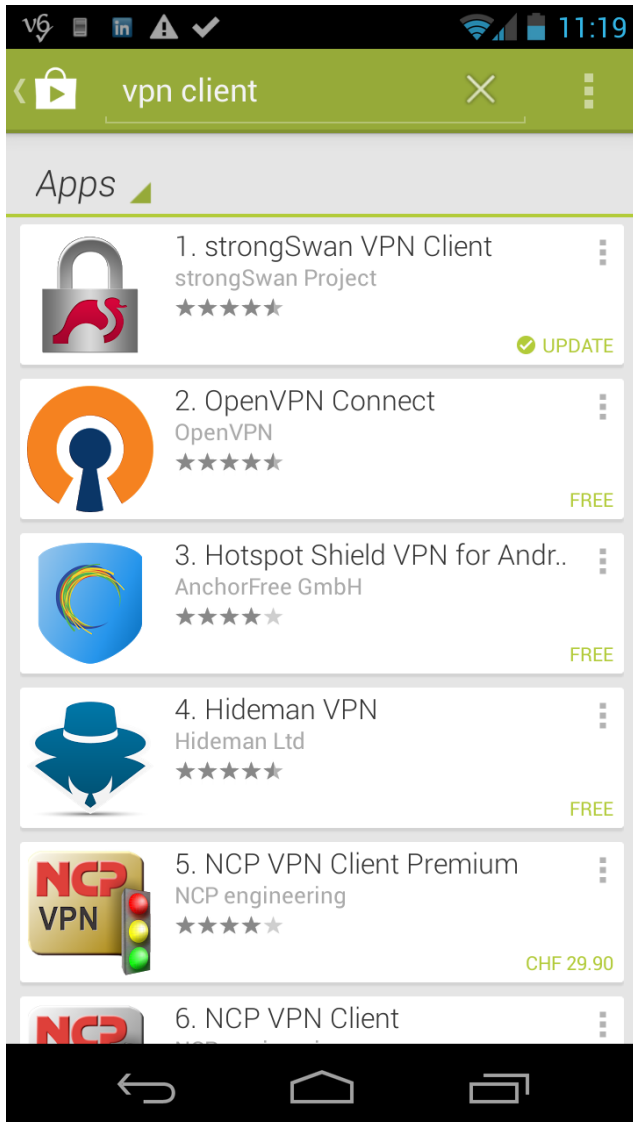


strongSwan – the OpenSource VPN Solution



- Supported Operating Systems
 - Linux 2.6.x, 3.x (optional integration into NetworkManager)
 - Android 4.x/5.x App (using libipsec userland ESP encryption)
 - OS X App (using libipsec userland ESP encryption)
 - OS X (IPsec via PFKEYv2 kernel interface)
 - FreeBSD (IPsec via PFKEYv2 kernel interface)
 - Windows 7/8 (native Windows IPsec stack, MinGW-W64 build)
- Supported Hardware Platforms (GNU autotools)
 - Intel i686/x86_64, AMD64
 - ARM, MIPS
 - PowerPC
- Supported Network Stacks
 - IPv4, IPv6
 - IPv6-in-IPv4 ESP tunnels
 - IPv4-in-IPv6 ESP tunnels

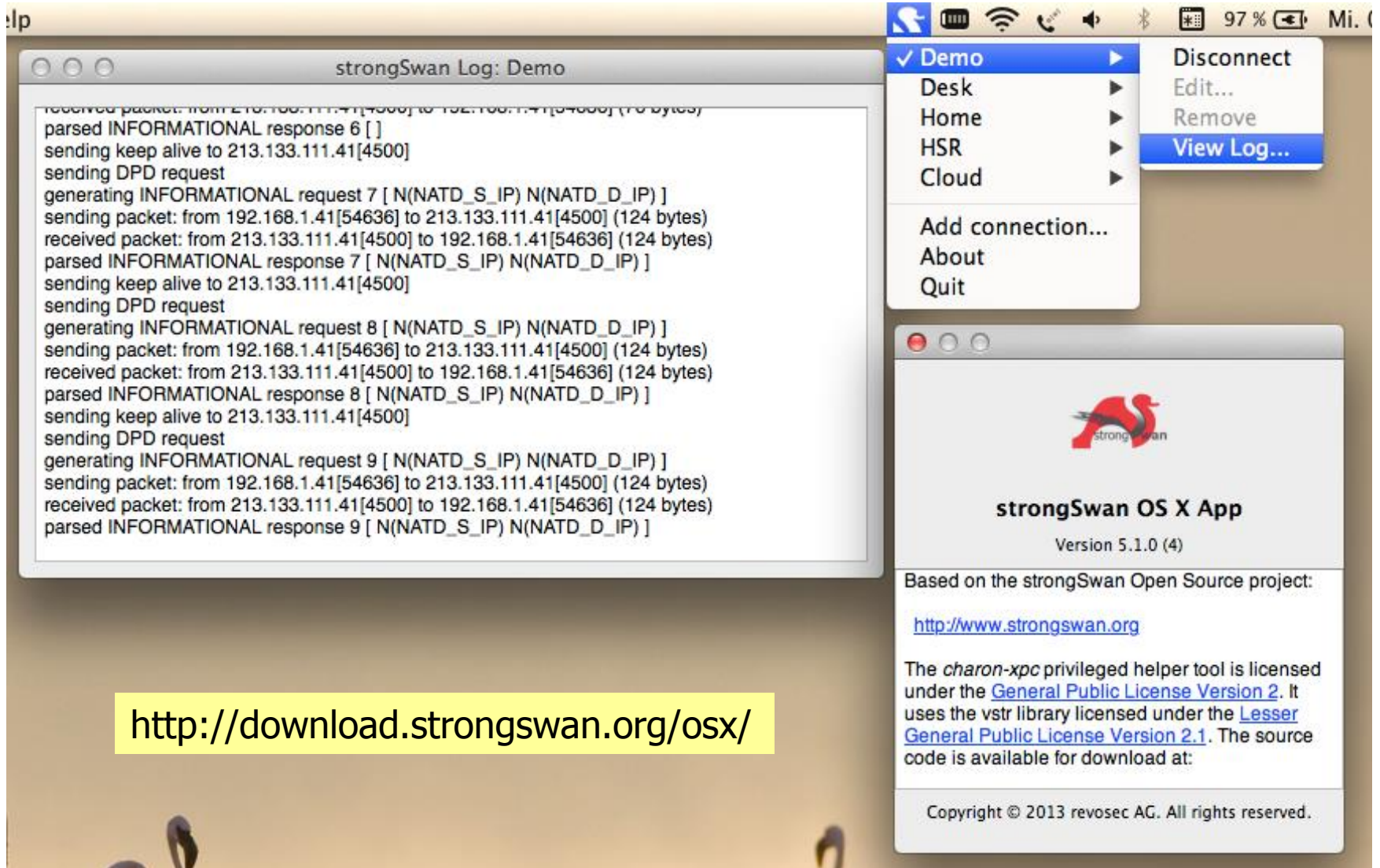
Free Download from Google Play Store



March 24, 2015:
12'619 installations

<input checked="" type="checkbox"/>	 United States	2,668	21.14%
<input checked="" type="checkbox"/>	 China	2,166	17.16%
<input checked="" type="checkbox"/>	 Germany	1,405	11.13%
<input type="checkbox"/>	 United Kingdom	578	4.58%
<input type="checkbox"/>	 Russia	484	3.84%
<input type="checkbox"/>	 Canada	325	2.58%
<input type="checkbox"/>	 France	276	2.19%
<input type="checkbox"/>	 Australia	262	2.08%
<input type="checkbox"/>	 Netherlands	249	1.97%
<input type="checkbox"/>	 Italy	235	1.86%
	 Others	3,971	31.47%

OS X App



The screenshot shows a Mac OS X desktop environment. In the foreground, there is a window titled "strongSwan OS X App" (Version 5.1.0 (4)). The window contains the strongSwan logo (a red swan) and the text "strongSwan OS X App" and "Version 5.1.0 (4)". Below this, it states "Based on the strongSwan Open Source project:" followed by the URL <http://www.strongswan.org>. It also mentions licensing: "The charon-xpc privileged helper tool is licensed under the [General Public License Version 2](#). It uses the vstr library licensed under the [Lesser General Public License Version 2.1](#). The source code is available for download at:" and "Copyright © 2013 revosec AG. All rights reserved."

In the background, a "strongSwan Log: Demo" window is open, displaying a log of network activity. The log shows a series of packets being sent and received, along with informational responses and keep-alive messages. The log entries include IP addresses and port numbers, such as 192.168.1.41 and 213.133.111.41.

A yellow highlight box is present at the bottom left of the screenshot, containing the URL: <http://download.strongswan.org/osx/>

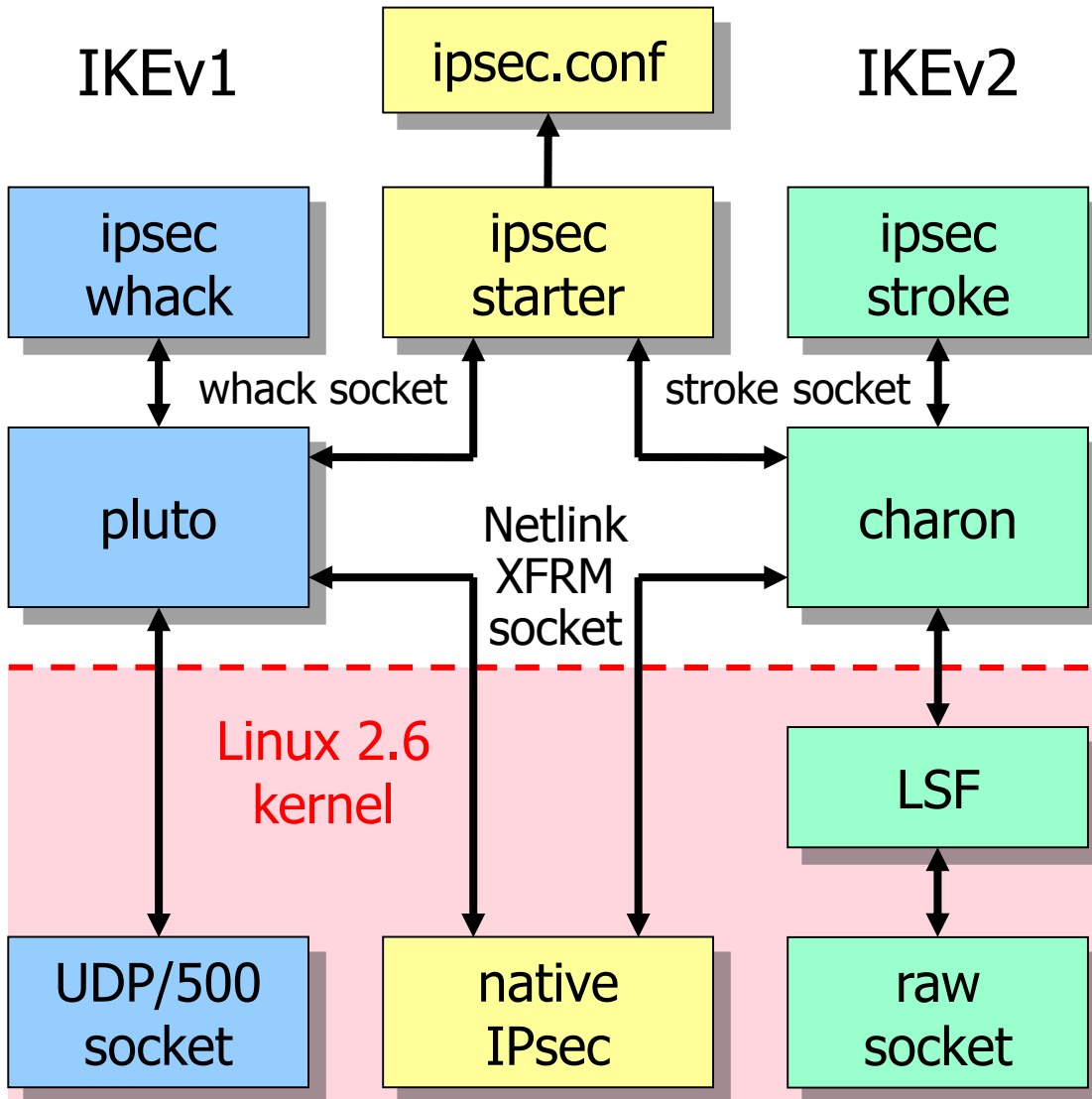
Neue strongSwan VPN Features

GUUG Frühjahrsfachgespräch 2015 Stuttgart

Evolution des strongSwan Charon IKE Dämons

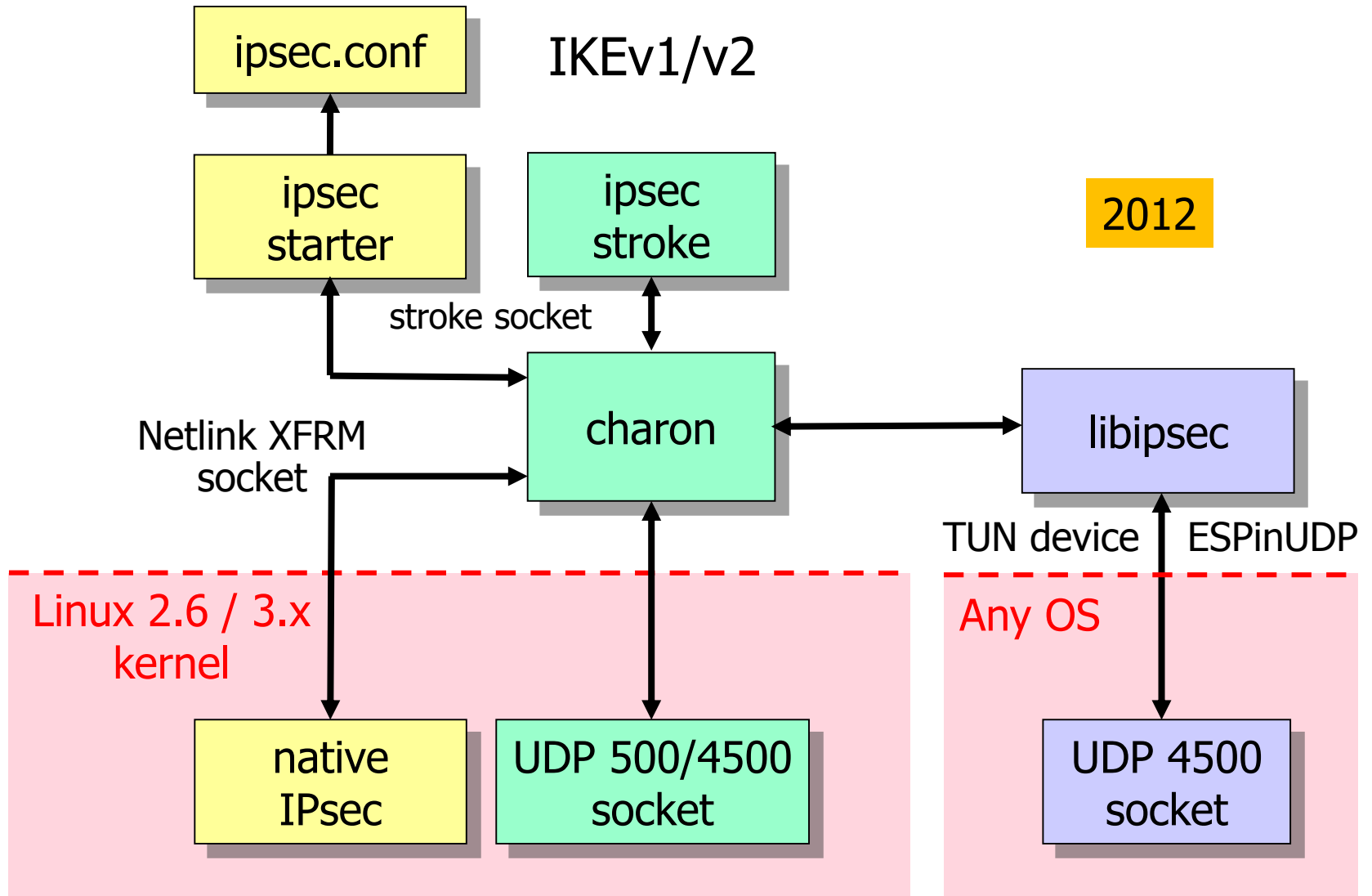


strongSwan 4.x **pluto** & **charon** Daemons

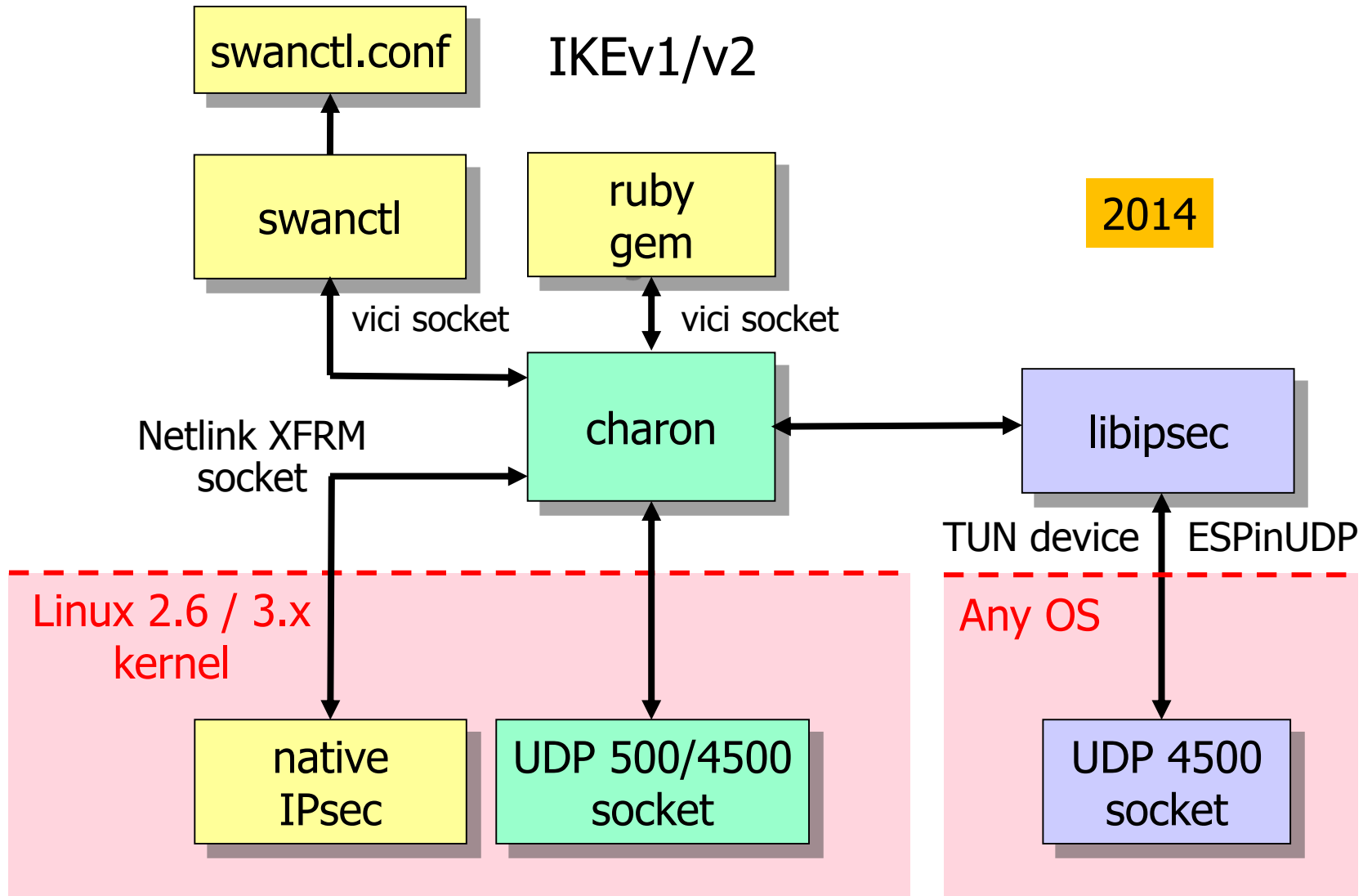


2005

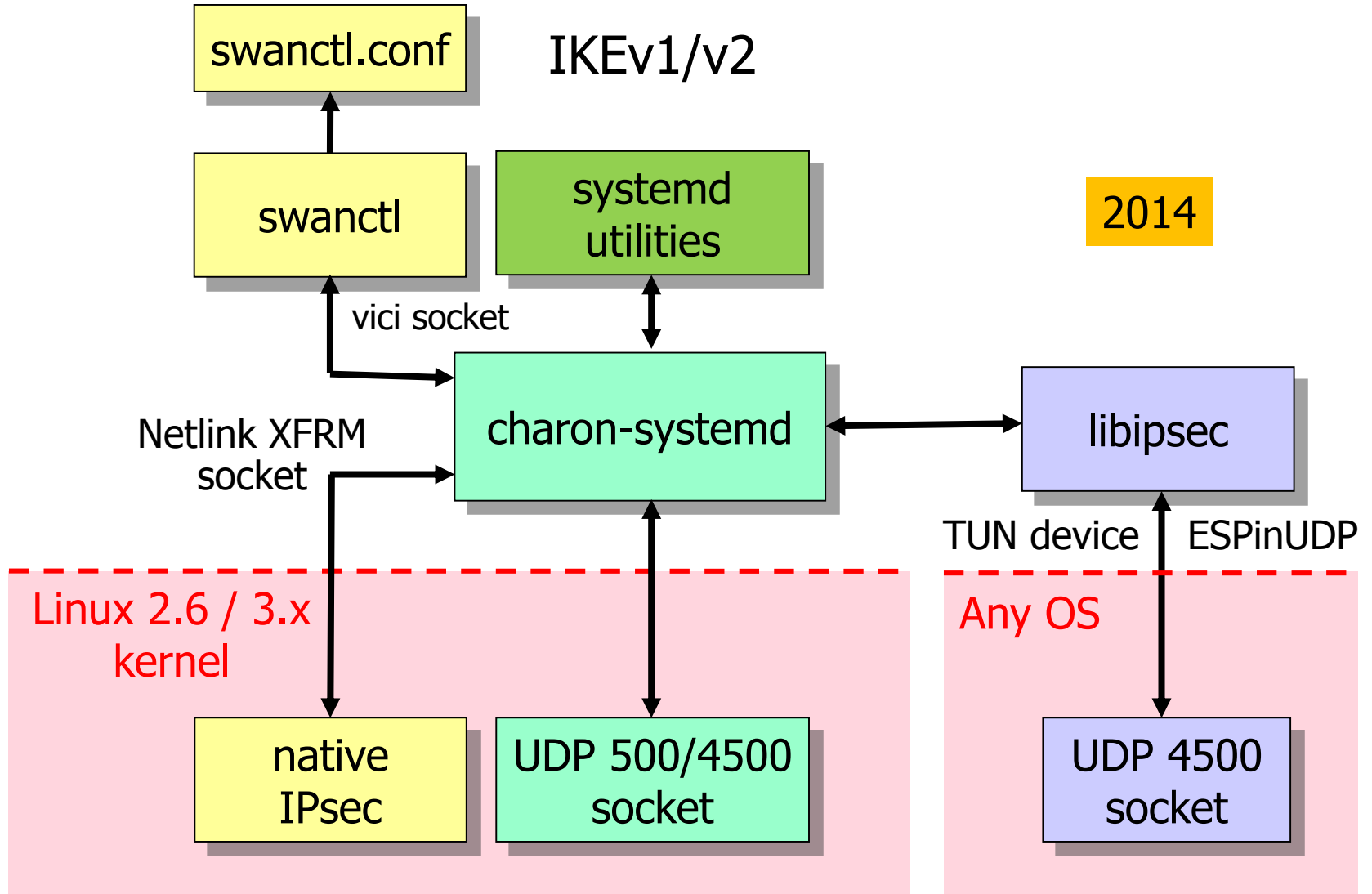
strongSwan 5.x charon Daemon



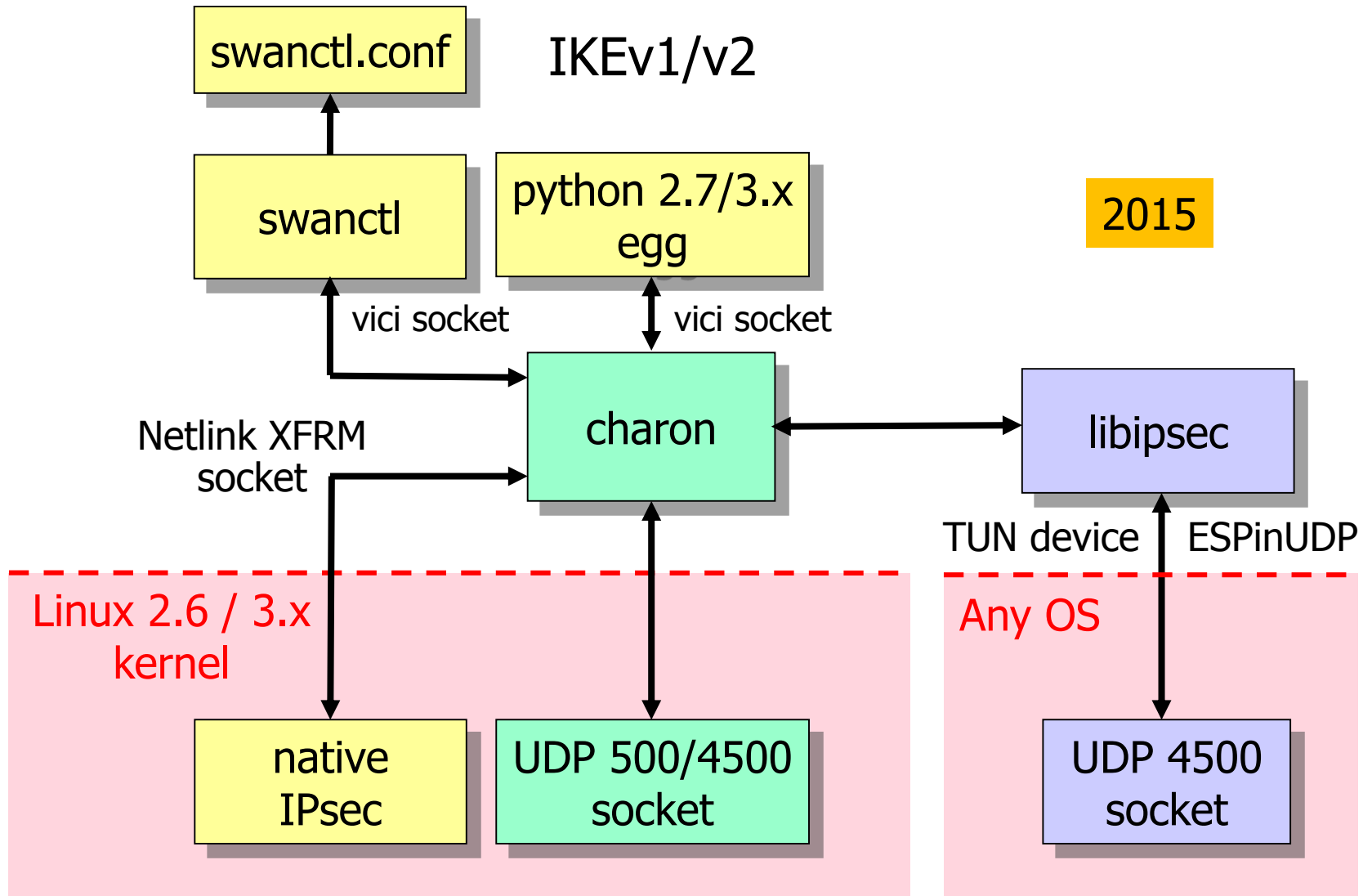
strongSwan 5.2 charon Daemon



strongSwan 5.2 charon-systemd Daemon



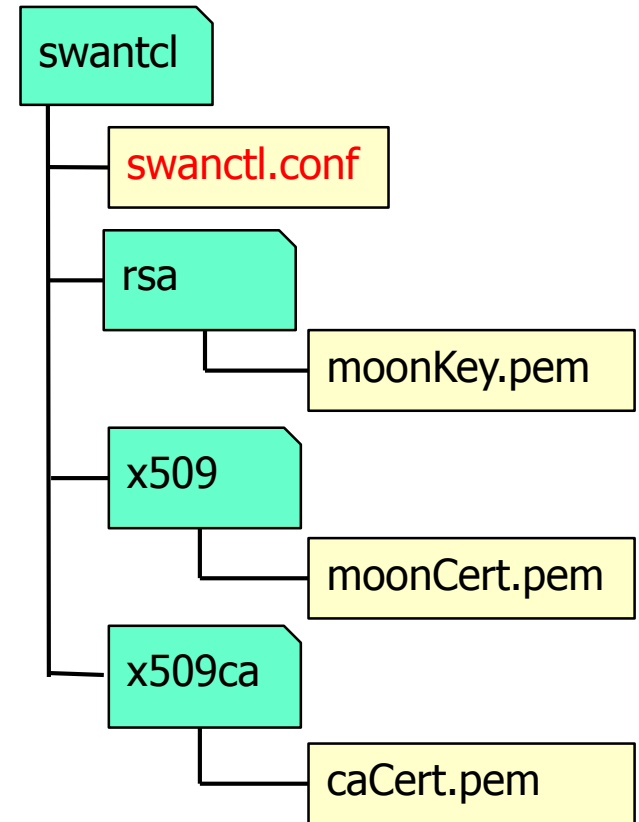
strongSwan 5.3 charon Daemon



swanctl.conf of VPN Gateway moon

```
connections {  
  rw {  
    local_addrs = 192.168.0.1  
    pools = rw_pool  
  
    local {  
      auth = pubkey  
      certs = moonCert.pem  
      id = moon.strongswan.org  
    }  
    remote {  
      auth = pubkey  
    }  
    children {  
      net {  
        local_ts = 10.1.0.0/16  
        start_action = none  
        esp_proposals = aes128gcm128-modp2048  
      }  
    }  
    version = 2  
    proposals = aes128-sha256-modp2048  
  }  
}
```

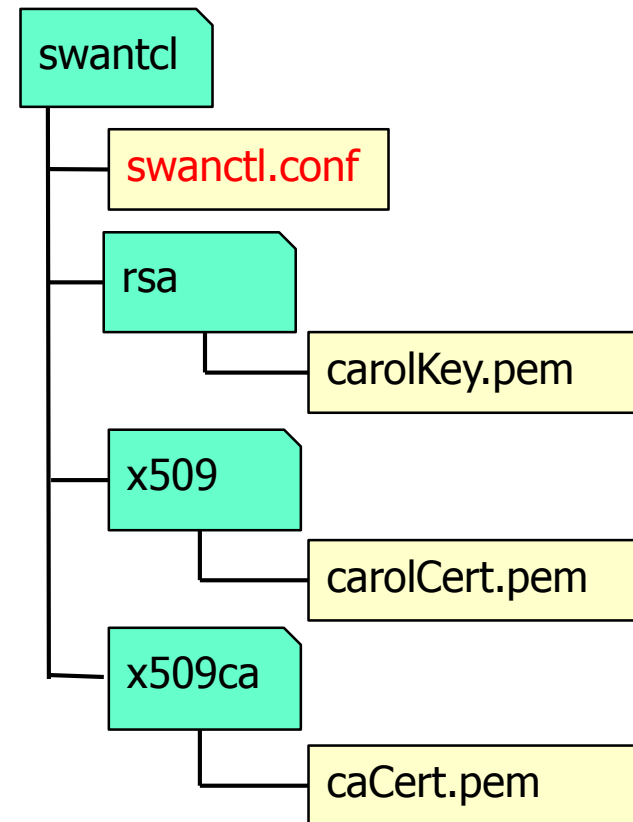
```
pools {  
  rw_pool {  
    addrs = 10.3.0.0/20  
  }  
}
```



swanctl.conf of VPN Client carol

```
connections {
  home {
    local_addrs = 192.168.0.100
    remote_addrs = 192.168.0.1
    vips = 0.0.0.0

    local {
      auth = pubkey
      certs = carolCert.pem
      id = carol@strongswan.org
    }
    remote {
      auth = pubkey
      id = moon.strongswan.org
    }
    children {
      home {
        remote_ts = 10.1.0.0/16
        start_action = none
        esp_proposals = aes128gcm128-modp2048
      }
    }
    version = 2
    proposals = aes128-sha256-modp2048
  }
}
```



swanctl - The Command Line Tool

```
moon# swanctl --load-creds
loaded x509 certificate from '/etc/swanctl/x509/moonCert.pem'
loaded x509ca certificate from '/etc/swanctl/x509ca/strongswanCert.pem'
loaded rsa key from '/etc/swanctl/rsa/moonKey.pem'

moon# swanctl --load-conns
loaded connection 'rw'
successfully loaded 1 connections, 0 unloaded

moon# swanctl --load-pools
loaded pool 'rw_pool'
successfully loaded 1 pools, 0 unloaded

carol# swanctl --initiate --child home
[IKE] initiating IKE_SA home[1] to 192.168.0.1
...
[IKE] installing new virtual IP 10.3.0.1
initiate completed successfully

carol# swanctl --terminate --ike home
...
[IKE] IKE_SA deleted
terminate completed successfully
```

swanctl - Monitoring Commands

```
moon# swanctl --list-conns
```

```
rw: IKEv2
```

```
local: 192.168.0.1
```

```
remote: %any
```

```
local public key authentication:
```

```
id: moon.strongswan.org
```

```
certs: C=CH, O=Linux strongSwan, CN=moon.strongswan.org
```

```
remote public key authentication:
```

```
net: TUNNEL
```

```
local: 10.1.0.0/16
```

```
remote: dynamic
```

```
moon# swanctl --list-sas
```

```
rw: #1, ESTABLISHED, IKEv2, b8deada3ec240a81:50af58eedcd556c7
```

```
local 'moon.strongswan.org' @ 192.168.0.1
```

```
remote 'carol@strongswan.org' @ 192.168.0.100
```

```
AES_CBC-128/HMAC_SHA2_256_128/PRF_HMAC_SHA2_256/MODP_2048
```

```
established 0s ago, rekeying in 1169s, reauth in 3259s
```

```
net: #1, reqid 1, INSTALLED, TUNNEL, ESP:AES_GCM_16-128
```

```
installed 0 ago, rekeying in 575s, expires in 660s
```

```
in c39fc9ac,      84 bytes,      1 packets,      0s ago
```

```
out c2c80483,    84 bytes,      1 packets,      0s ago
```

```
local 10.1.0.0/16
```

```
remote 10.3.0.1/32
```

Neue strongSwan VPN Features

GUUG Frühjahrsfachgespräch 2015 Stuttgart

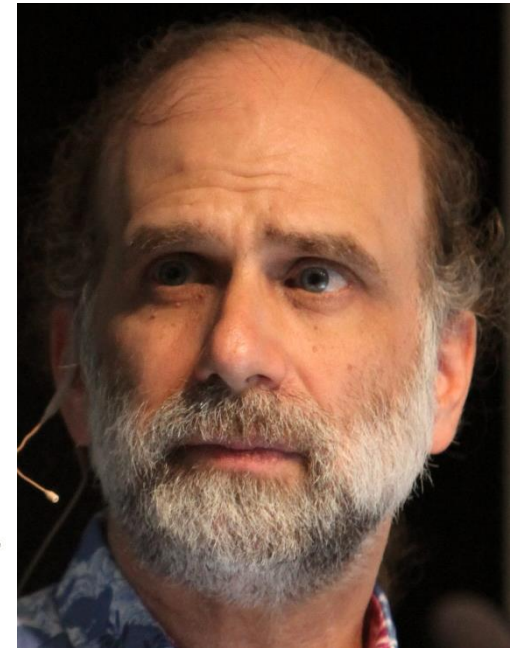
Der Schwan wird stärker!



The Snowden Documents – Fall 2013



Edward Snowden



Bruce Schneier



Laura Poitras



Glenn Greenwald

Principle of Comparative Security Strength*

Symmetric Key	RSA / DH	ECDSA / ECDH	Hash
80	1024	160	160
112	2048	224	224
128	3072	256	256
192	7680	384	384
256	15360	512	512

- NIST SP 800-57 Recommendation for Key Management: Part 1 General (Revision 3, 2012)

*cryptographic strength given in bits

Getting rid of SHA-1

- SHA-1 has a hash size of 160 bits which was supposed to give a strength of 2^{80} against collision attacks. Unfortunately SHA-1 is much weaker with the best known attack having a complexity of 2^{61} only.
- The NSA might already have found a SHA-1 collision, using it e.g. to generate fake X.509 certificates.
- IKEv2 uses SHA-1 as a hardwired algorithm to generate RSA digital signature AUTH payloads.
- RFC 7427 "Signature Authentication in IKEv2" published in January 2015 allows to negotiate SHA-2 hash algorithms and is used per default by strongSwan 5.3.0:

Hash
160
224
256
384
512

```
moon charon: 15[IKE] authentication of 'sun.strongswan.org'  
with RSA_EMSA_PKCS1_SHA256 successful
```

Can the NSA break RSA and DH faster?

- According to Lenstra's updated formula on www.keylength.com a 1024 bit RSA key or DH factor could be cracked in 2006 with an effort of **40'000'000 dollardays**.
- Due to Moore's law (factor $2^6 = 64$ in $6 \times 1.5 = 9$ years) the effort in 2015 has fallen to **625'000 dollardays**.
- Many cryptanalysts expect a major breakthrough in prime number factoring (RSA) and the computation of the discrete logarithm (DH) within the next few years.
- The NSA might already have much more efficient algorithms.
- As a precaution better use 4096 bit RSA moduli and 4096 bit DH groups.

RSA / DH
1024
2048
3072
7680
15360

Can we trust the NIST Elliptic Curves?

- The NIST Elliptic Curves are based on pseudo-Mersenne primes

```
ike=aes128-sha256-ecp256,aes192-sha384-ecp384!
```

The NIST curve parameter selection process is not documented!

- Use the European (BSI) Brainpool Elliptic Curves instead

```
ike=aes128-sha256-ecp256bp,aes192-sha384-ecp384bp!
```

RFC 6932 Brainpool Elliptic Curves for IKE, 2013.

- Drawback: Brainpool ECDH performance is 5x slower than with NIST curves since the selected primes are random.
- Use Dan Bernstein's popular Curve25519?
- ECC NUMS (Nothing Up My Sleeve) Curves, 2014
tools.ietf.org/html/draft-black-numscurves

ECDH
160
224
256
384
512

Does the NSA have a Quantum Computer?

- If a quantum computer with enough qbits becomes available, Elliptic Curve Cryptography (ECC) is going to fall first!
- As an alternative to the ECDH key exchange strongSwan can use **NTRU encryption** based on the shortest-vector problem in a high-dimensional lattice which is known to be resistant to quantum computer attacks.
- NTRU encryption has been standardized by IEEE Std 1363.1-2008. The fast algorithm is owned by Security Innovations but is available under a GPLv2 open source license.
- NTRU encryption has been implemented by the strongSwan **ntru** plugin:

```
ike=aes128-sha256-ntru256,aes192-sha384-ntru384!
```

ECDH
160
224
256
384
512

Post-Quantum Digital Signatures?

- NTRU signature has been shown to leak the private key over time so that the key can be determined after a few thousand signatures.
- The most promising candidate is BLISS (Bimodal Lattice Signature Scheme, 2013) in its enhanced BLISS-B form ("Accelerating Bliss: the geometry of ternary polynomials" by Léo Ducas, 2014).

Scheme	Strength	Signature Size
BLISS-I	128 bits	≈ 5800 bits
BLISS-III	160 bits	≈ 6200 bits
BLISS-IV	192 bits	≈ 6800 bits

ECDSA
160
224
256
384
512

- BLISS-B signatures have been implemented by the strongSwan 5.3.0 **bliss** plugin:

```
moon charon: 14[IKE] authentication of 'sun@strongswan.org'  
with BLISS_WITH_SHA256 successful
```

Generating BLISS Keys and Certificates I

```
# Generate a BLISS-IV private CA key
pki --gen --type bliss --size 4 --outform pem > caKey.pem
secret key generation succeeded after 1 trial

# Generate a self-signed BLISS-IV CA certificate
pki --self --type bliss --in caKey.der --ca --lifetime 3653 \
    --dn "C=CH, O=Demo, CN=ISS Root CA" --outform pem > caCert.der

# Print info on BLISS-IV CA certificate
pki --print --in caCert.pem
cert:      X509
subject:   "C=CH, O=Demo, CN=BLISS Root CA"
issuer:    "C=CH, O=Demo, CN=BLISS Root CA"
validity:  not before Mar 15 17:58:01 2015, ok
           not after  Mar 15 17:58:01 2025, ok (expires in 3652 days)
serial:    55:9c:dd:7d:32:89:99:a8
flags:     CA CRLSign self-signed
subjKeyId: 47:bd:9e:5e:a8:58:ce:60:14:73:f3:54:7c:e8:28:10:7b:e6:c7:65
pubkey:    BLISS 192 bits strength
keyid:     1c:a7:5c:94:d1:ee:f6:c7:94:21:18:e5:ef:89:b3:c3:64:42:24:97
subjkey:   47:bd:9e:5e:a8:58:ce:60:14:73:f3:54:7c:e8:28:10:7b:e6:c7:65
```

Generating BLISS Keys and Certificates II

```
# Generate a BLISS-I private key for server moon
pki --gen --type bliss --size 1 > moonKey.der
secret key generation succeeded after 1 trial

# Generate a self-signed PKCS #10 certificate request for server moon
pki --req --type bliss --in moonKey.der --san moon.strongswan.org \
    --dn "C=CH, O=Demo, CN=moon.strongswan.org" > moonReq.der

# Generate a BLISS-I host certificate signed by the BLISS-IV CA key
pki --issue --type pkcs10 --in moonReq.der --flag serverAuth \
    --cacert caCert.pem --caKey caKey.pem \
    --crl http://crl.strongswan.org/bliss.crl > moonCert.der

# Generate an empty CRL signed by the BLISS-IV CA key
pki --signcrl --cacert caCert.pem --caKey caKey.pem \
    --lifetime 15 > bliss.crl
```

Generating BLISS Keys and Certificates III

Print info on BLISS-I server certificate

```
pki --print --in moonCert.der
```

```
cert:          X509
subject:       "C=CH, O=Demo, CN=moon.strongswan.org"
issuer:        "C=CH, O=Demo, CN=BLISS Root CA"
validity:      not before Mar 15 18:04:00 2015, ok
                not after  Mar 14 18:04:00 2018, ok (expires in 1094 days)
serial:        43:63:44:f0:7f:2f:aa:dc
altNames:      moon.strongswan.org
flags:         serverAuth
CRL URIs:      http://crl.strongswan.org/bliss.crl
authkeyId:     47:bd:9e:5e:a8:58:ce:60:14:73:f3:54:7c:e8:28:10:7b:e6:c7:65
subjkeyId:     cb:b5:c3:d5:00:ba:bb:90:ec:80:99:05:68:72:ae:3b:04:f8:9b:5f
pubkey:        BLISS 128 bits strength
```

Print info on the BLISS CRL

```
pki --print -type crl --in bliss.crl
```

```
cert:          X509_CRL
issuer:        "C=CH, O=Demo, CN=BLISS Root CA"
validity:      not before Mar 25 11:58:25 2015, ok
                not after  Apr 24 12:58:25 2015, ok (expires in 29 days)
serial:        01
authkeyId:     47:bd:9e:5e:a8:58:ce:60:14:73:f3:54:7c:e8:28:10:7b:e6:c7:65
```


Neue strongSwan VPN Features

GUUG Frühjahrsfachgespräch 2015 Stuttgart

Wie zerlegt man einen Schwan?



HSR

HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz



Fixing IKE Message Fragmentation Problems

- IKE messages are transported in UDP datagrams. If the message size exceeds the MTU then the UDP datagrams are split up into IP fragments. This is always the case with CERT payloads containing X.509 certificates with 2048 bit RSA keys.
- Often IP fragments are discarded by routers and firewalls on the way between the VPN endpoints so that IKE negotiation fails.
- RFC 7383 "IKEv2 Message Fragmentation" solves the problem:

```
Mar 15 12:18:03 carol charon:
  sending end entity cert "C=CH, O=Demo, OU=BLISS I, CN=sun.strongswan.org"
  generating IKE_AUTH request 1 [ IDi CERT N(INIT_CONTACT) CERTREQ Idr
                                AUTH CPRQ(ADDR) SA TSi TSr ]
  splitting IKE message with length of 3232 bytes into 3 fragments
  generating IKE_AUTH request 1 [ EF ]
  generating IKE_AUTH request 1 [ EF ]
  generating IKE_AUTH request 1 [ EF ]
  sending packet: from 192.168.0.100[4500] to 192.168.0.1[4500] (1460 bytes)
  sending packet: from 192.168.0.100[4500] to 192.168.0.1[4500] (1460 bytes)
  sending packet: from 192.168.0.100[4500] to 192.168.0.1[4500] ( 452 bytes)
```

Configuring IKE Message Fragmentation

- IKE fragmentation is enabled by adding the line

```
fragmentation=yes
```

to the connection definition in [ipsec.conf](#) or [swanctl.conf](#).

- RFC 7383 "IKEv2 Message Fragmentation" defines a default MTU of **576 bytes** for IPv4 and **1280 bytes** for IPv6. The size of the IKE fragments is automatically computed to fit into an IPv4 or IPv6 packet with the given MTU.
- The fragmentation size can be set manually in [strongswan.conf](#):

```
charon {  
    fragment_size = 1500  
}
```

which assumes an MTU of 1500 bytes.

- This also works for IKEv1 where the proprietary Microsoft fragmentation scheme is used.

Danke für Ihre Aufmerksamkeit!

Fragen?

www.strongswan.org

