A Systematic Analysis of the Juniper Dual EC Incident

Jacob Maskiewicz

ACM CCS 2016

with Stephen Checkoway, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham

University of Illinois at Chicago, University of California San Diego, Johns Hopkins University, University of Pennsylvania, Comsecuris

- Administrative Access (CVE-2015-7755)
- VPN Decryption (CVE-2015-7756)

https://twitter.com/thegrugq/status/677611480041127936

Administrative Access Backdoor

```
ADD     R0, R5, #0x44
LDR     R1, =aSUnSU ; "<<< %s(un='%s') = %u"
BL      strcmp
CMP     R0, #0
BNE     loc_13DC78
MOV     R0, #0xFFFFFFFD
LDMDB   R11, {R4-R8,R11,SP,PC}
```

Extra check in `auth_admin_internal` allows admin login using password:

`<<< %s(un='%s') = %u`

ARM disassembly from H.D. Moore — https://community.rapid7.com/community/infosec/blog/2015/12/20/cve-2015-7755-juniper-screenos-authentication-backdoor

Changed constants in an H.D. Moore diff

P-256 Weierstraß b

5AC635D8AA3A93E7B3EBBD557 … CC53B0F63BCE3C3E27D2604B

P-256 P x coord

6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F … 6

P-256 field order

FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551

bad: 9585320EEAF81044F20D55030A035B11BECE81C785E6C933E4A8A131F6578107

good: 2c55e5e45edf713dc43475effe8813a60326a64d9ba3d2e39cb639b0f3b0ad10

nist: c97445f45cdef9f0d3e05e1e585fc297235b82b5be8ff3efca67c59852018192

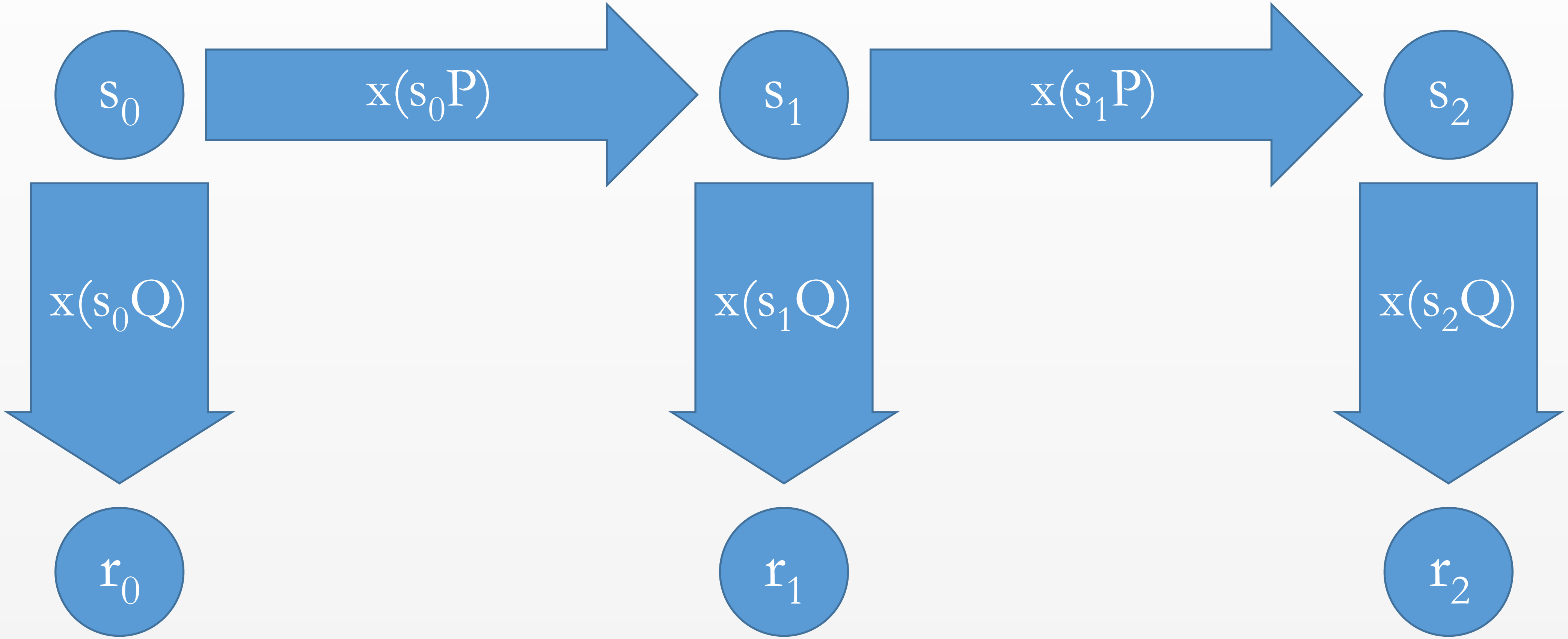
Reverse engineering shows changed values are x coords for Dual EC point Q

https://cseweb.ucsd.edu/~hovav/dist/rwc16.pdf

Dual EC DRBG History

- Early 2000s: Created by the NSA and pushed towards standardization
- 2004: Published as part of ANSI x9.82 part 3 draft
- 2004: RSA makes Dual EC the default CSPRNG in BSAFE ($10mil)
- 2005: Standardized in NIST SP 800-90 draft
- 2007: Shumow and Ferguson demonstrate theoretical backdoor attack
- 2013: Snowden documents lead to renewed interest in Dual EC
- 2014: Practical attacks on TLS using Dual EC demonstrated
- 2014: NIST removes Dual EC from list of approved PRNGs

Dual EC DRBG

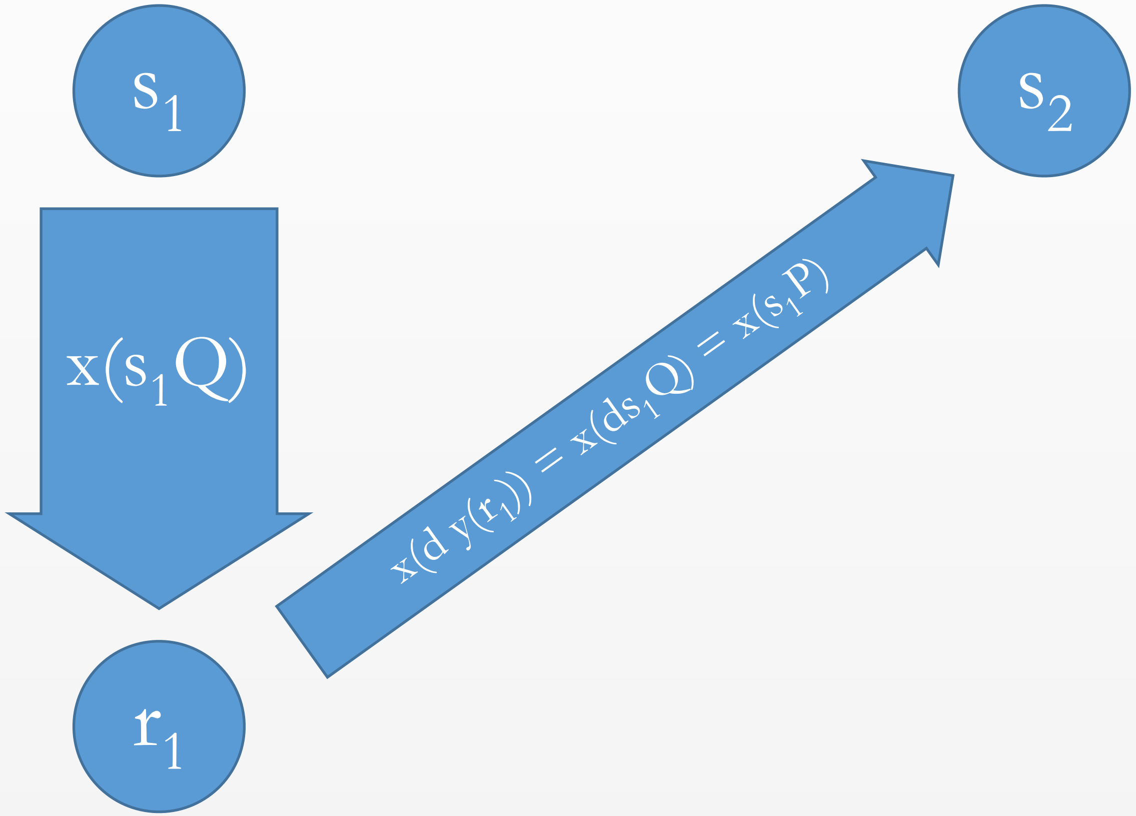


Note: r is actually the 30 least significant bytes of the x value

Dual EC DRBG Backdoor

Assume an attacker who knows $\log_Q P$ aka

d st. $P = dQ$



How to find $\log_Q P$

Disclaimer: Without more information, given P and Q there is not a way to tell if they were generated safely

- Solve the discrete log problem
- Be in charge of the official curve parameters
  - Fix Q, d, define $P = dQ$
  - Fix P, e, define $Q = eP$, compute $d = e^{-1}$
- Use your own curve parameters

e.g. the NSA

Juniper's use of Dual EC

- ScreenOS is only FIPS validated for ANSI x9.31, not Dual EC

The following product families do utilize Dual_EC_DRBG, but do not use the pre-defined points cited by NIST:

1. ScreenOS*

* ScreenOS does make use of the Dual_EC_DRBG standard, but is designed to not use Dual_EC_DRBG as its primary random number generator. ScreenOS uses it in a way that should not be vulnerable to the possible issue that has been brought to light. Instead of using the NIST recommended curve points it uses self-generated basis points and then takes the output as an input to FIPS/ANSI X.9.31 PRNG, which is the random number generator used in ScreenOS cryptographic operations.

Questions

- Why does a change in Q result in a passive VPN Decryption vulnerability?
- We doesn’t Juniper’s use of X9.31 protect their system against a compromise of Q?
- What is the history of the PRNG code in ScreenOS?
- How was Juniper’s Q value generated?
- Is the version of ScreenOS with Juniper’s Q vulnerable to attack?

We can explore the answers to these questions
using forensic reverse engineering

ScreenOS RNG

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}
```

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}
```

Conditional Reseed

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

Generate Dual EC Output

Copy to prng internals

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}
```

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}
```

Generate output with new key

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}
```

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}
```

Global Variable

Always true

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

Global Variable

Set to 32

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}
```

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

ScreenOS RNG

```c
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}
```

Never Runs

Uses same buffer

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on analysis of the apparent function of each symbol

Internet Key Exchange (IKE) protocol

- Used to establish keys for VPN session
- Two major versions, IKEv1 and v2
- Both use two phases:
  - Phase 1 establishes keys to encrypt the phase 2 handshake
  - Phase 2 establishes keys for IPSec (or other encapsulated protocol)
- Both phases present nonces and use a Diffie-Hellman key exchange

IKE Phase 1 Handshake

- Header
- Payload: Security Association
  - Contains details about which cipher suites to use
- Payload: Key Exchange
  - Contains DH key exchange data, g^x
- Payload: Nonce
  - Contains 8-128 byte random value
- Other payloads
  - Vendor info, identification, etc.

IKE Phase 1 Handshake

- Header
- Payload: Security Association
  - Contains details about which cipher suites to use
- Payload: Key Exchange
  - Contains DH key exchange data, g^x
- Payload: Nonce
  - Contains 8-128 byte random value
- Other payloads
  - Vendor info, identification, etc.

ScreenOS x comes directly from Dual EC

ScreenOS uses 32-byte nonce from Dual EC

ScreenOS Multiple Handshake Key Recovery Attack

Key Exchange value generated before Nonce means we need to see multiple handshakes

$$s_0 \rightarrow r_0$$

$$\downarrow$$

$$s_1 \rightarrow r_1$$

$$\downarrow$$

$$s_2 \rightarrow r_2$$

$$\downarrow$$

$$s_3 \rightarrow r_3$$

ScreenOS Multiple Handshake Key Recovery Attack

Key Exchange value generated before Nonce means we need to see multiple handshakes

$s_0 \rightarrow r_0$

$\downarrow$

$s_1 \rightarrow r_1$

$\downarrow$

$s_2 \rightarrow r_2$

$\downarrow$

$s_3 \rightarrow r_3$

IKE Handshake 1

KE = $g^{\wedge}r_0$

Nonce = r_1

IKE Handshake 2

KE = $g^{\wedge}r_2$

Nonce = r_3

ScreenOS Multiple Handshake Key Recovery Attack

Key Exchange value generated before Nonce means we need to see multiple handshakes

$s_0 \rightarrow r_0$

$\downarrow$

$s_1 \rightarrow r_1$

$\downarrow$

$s_2 \rightarrow r_2$

$\downarrow$

$s_3 \rightarrow r_3$

IKE Handshake 1

KE = $g^{\wedge}r_0$

Nonce = r_1

backdoor

IKE Handshake 2

KE = $g^{\wedge}r_2$

Nonce = r_3

$s_2 \rightarrow r_2$

$\downarrow$

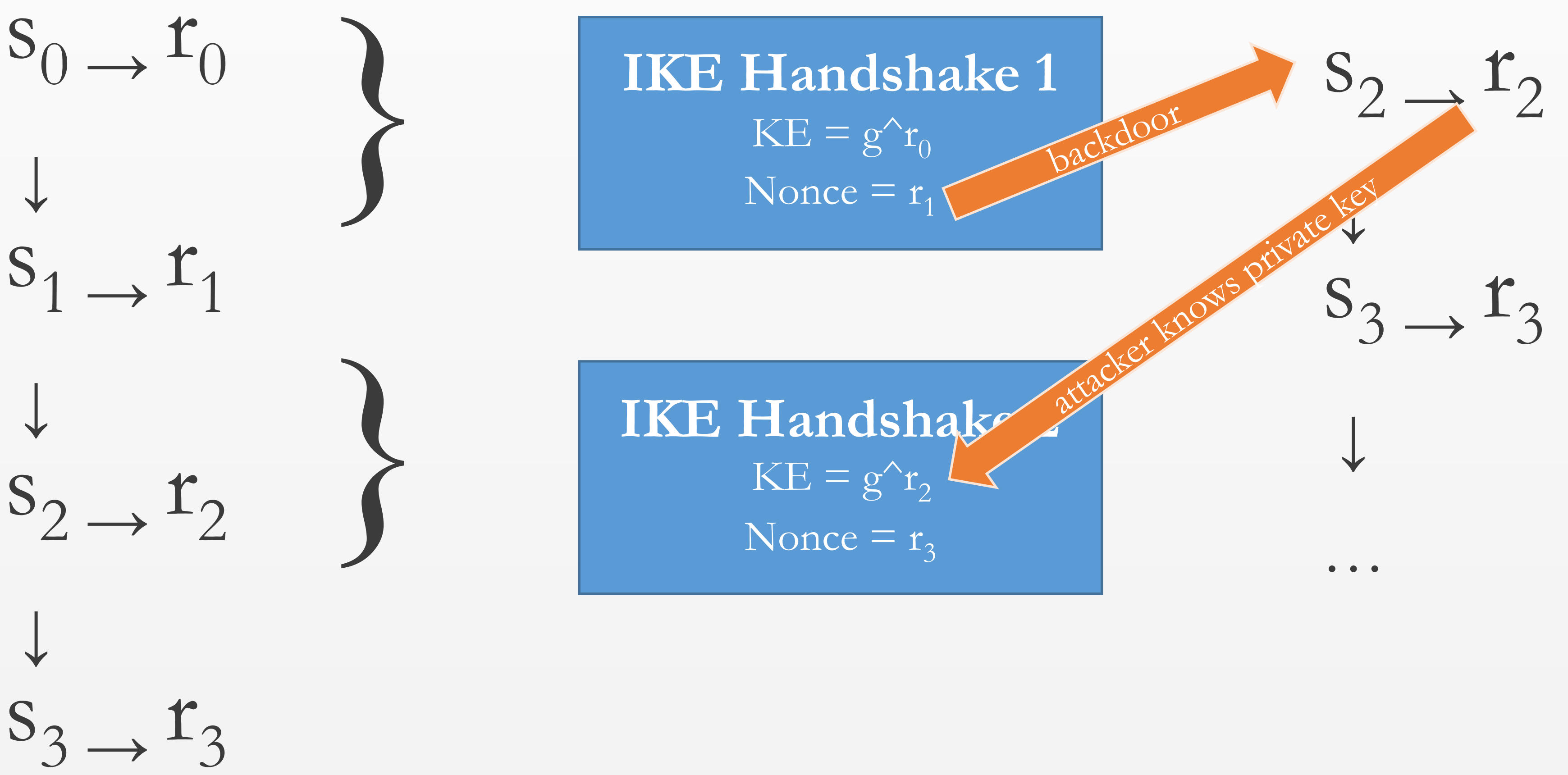
$s_3 \rightarrow r_3$

$\downarrow$

…

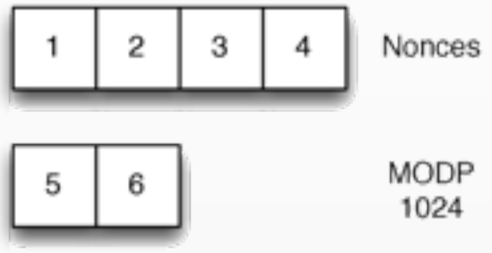
ScreenOS Multiple Handshake Key Recovery Attack

Key Exchange value generated before Nonce means we need to see multiple handshakes

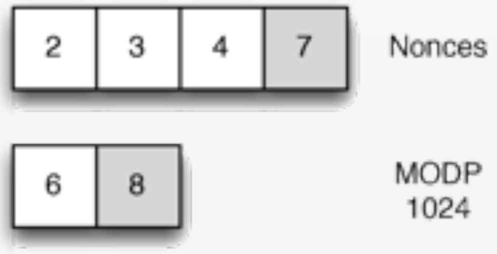


Nonce Queues

- There are queues for each of:
  - Nonces
  - MODP DH groups
    - 768, 1024, 1536, and 2048 bit
  - ECP DH groups
    - 256 and 384 bit
- Filled in background process
- Nonces **always** generated before keys



At system startup



After a DH exchange

ScreenOS Single Handshake Key Recovery Attack

$s_0 \rightarrow r_0$

$\downarrow$

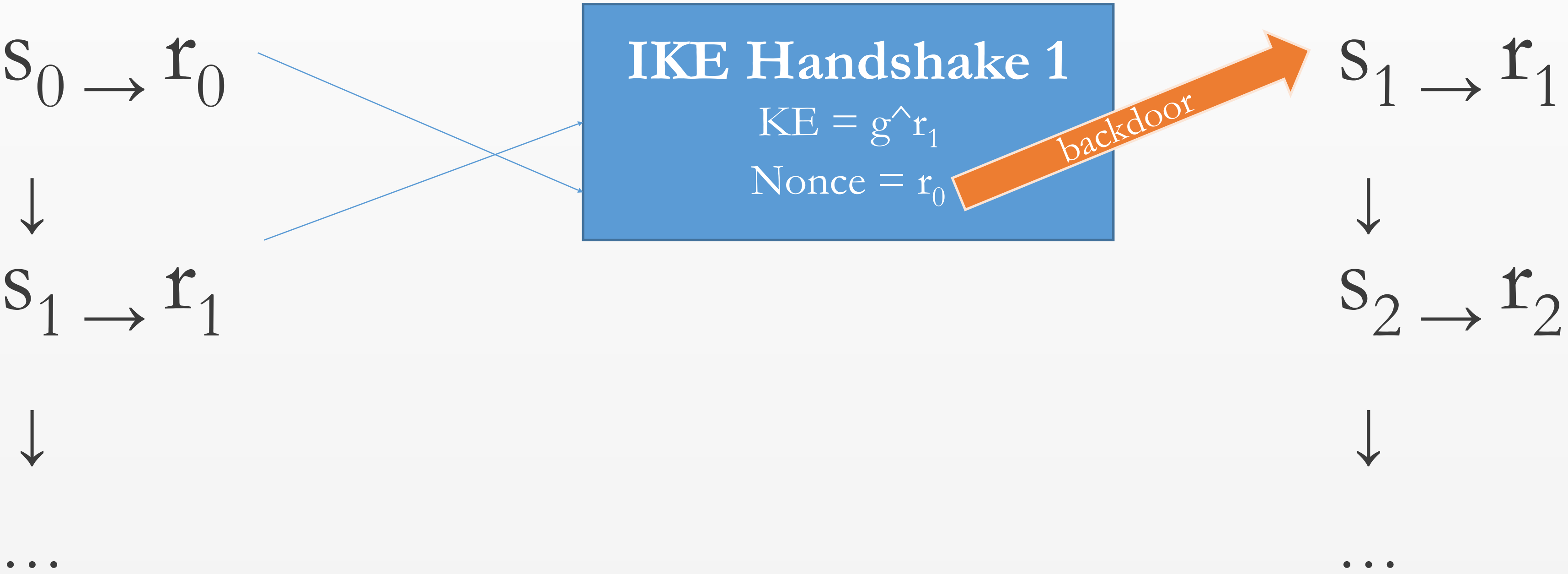
$s_1 \rightarrow r_1$

$\downarrow$

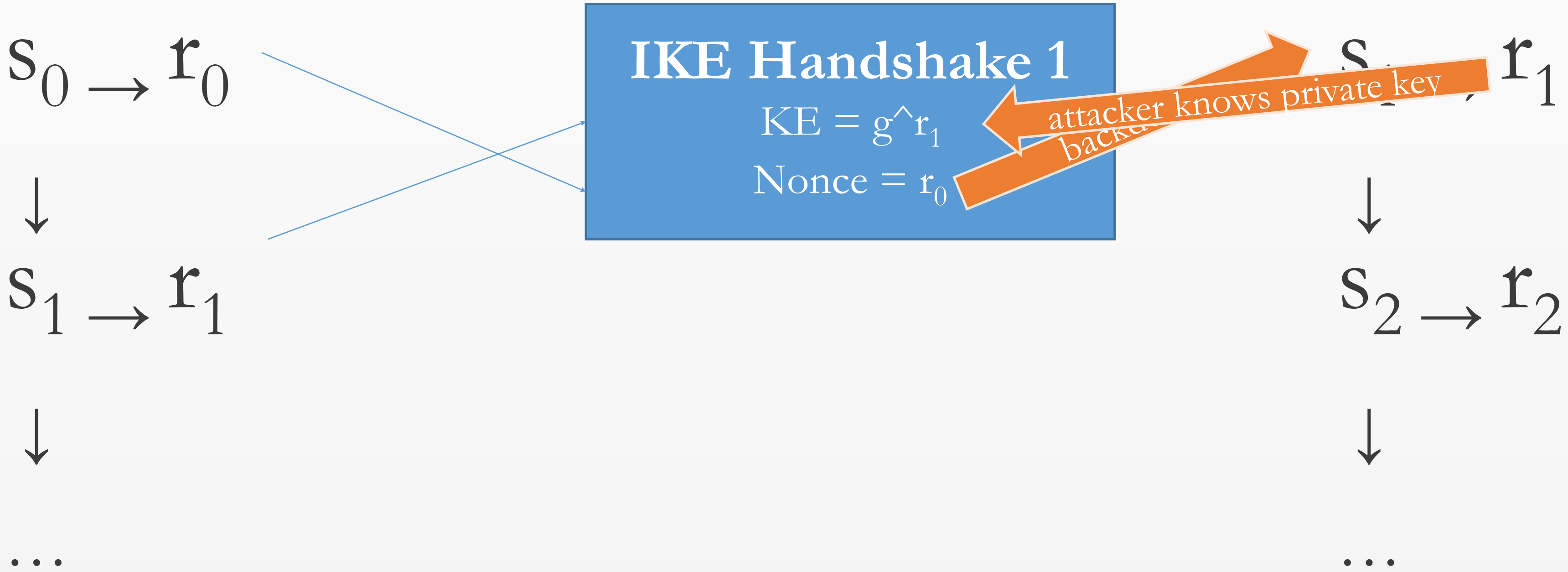
…

ScreenOS Single Handshake Key Recovery Attack

ScreenOS Single Handshake Key Recovery Attack



ScreenOS Single Handshake Key Recovery Attack



Caveats

- Many scenarios can downgrade single handshake attack to multiple handshake attack:
  - Fast connections exhaust queue
  - Non-DH phase 2 exchanges
  - Multiple DH queues at different rates (figure 2 in the paper)

Proof of Concept

- Purchased a Netscreen SSG 550M
- Created a modified firmware with our own Q (for which we know the discrete log d)
- Generated VPN connections in several configurations
  - IKEv1 with PSK
  - IKEv1 with RSA cert
  - IKEv2 with PSK

Did it Work?

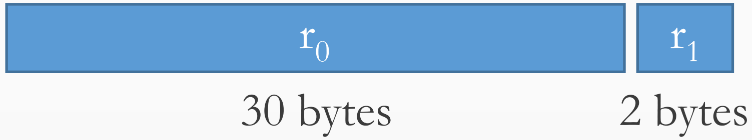
- Attack worked on:
  - ~~IKEv1 with PSK~~ (attacker needs PSK)
  - IKEv1 with RSA cert
  - IKEv2 with PSK
  - Should work on IKEv2 with cert

Version History

- ScreenOS 6.1.0r7 (last 6.1 version)
  - ANSI x9.31
    - Seeded by Interrupts
    - Reseeds every 10k calls
  - DH Queues
  - 20-byte IKE nonces

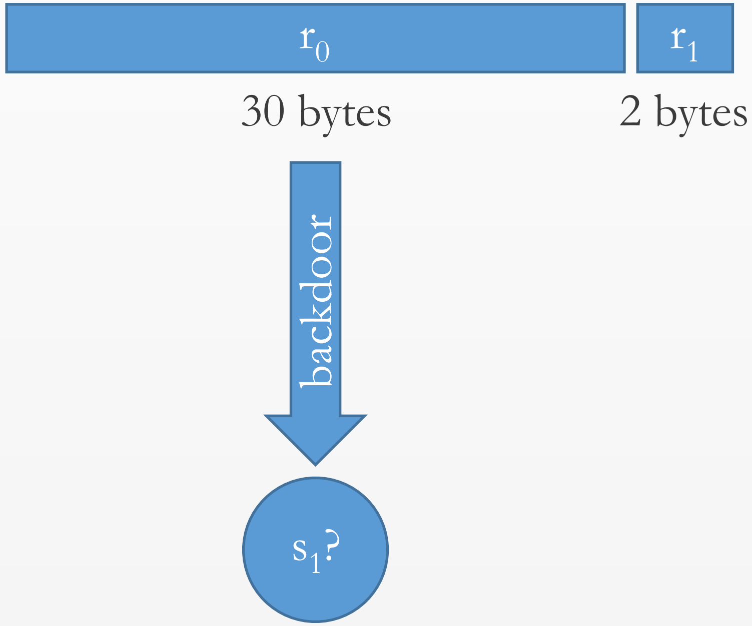
- ScreenOS 6.2.0r0 (first 6.2 version)
  - DualEC → ANSI x9.31
    - Reseed Bug exposes DualEC
    - Reseeds every call
  - Nonce Queues before DH Queues
  - 32-byte nonces

32-Byte Nonces



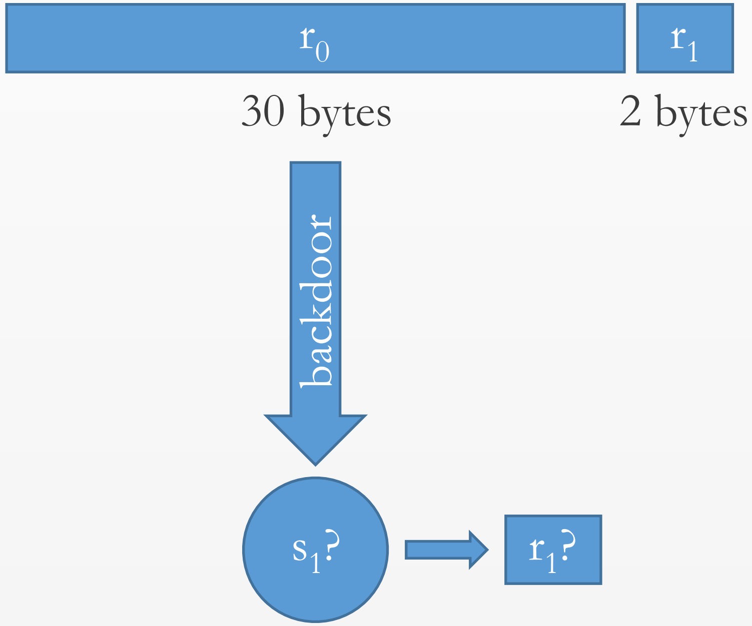
- 32-byte Dual EC outputs actually facilitate the attack:
  - Use first 30 bytes to recover 2^{15} possible states
  - For each possible state, generate a value and test against last 2 bytes

32-Byte Nonces



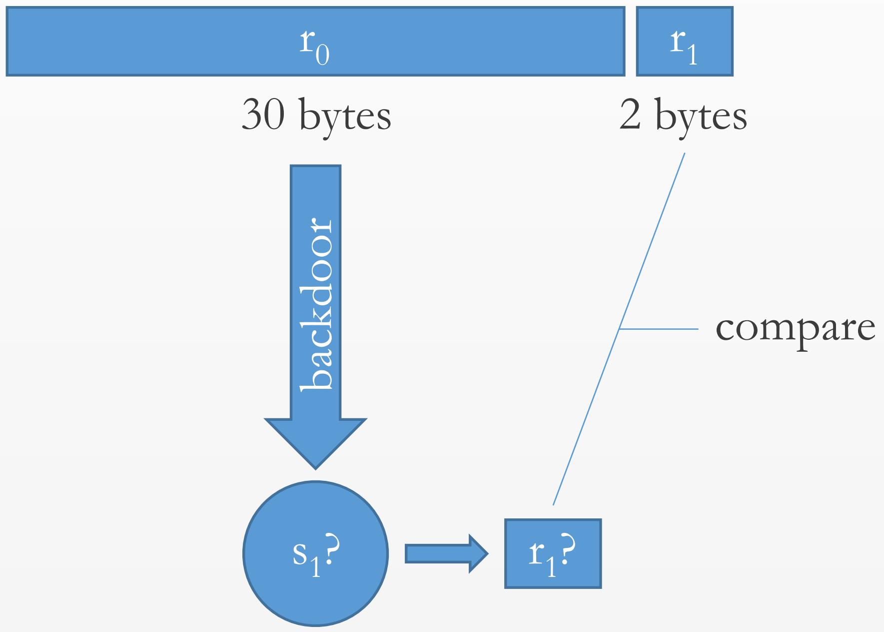
- 32-byte Dual EC outputs actually facilitate the attack:
  - Use first 30 bytes to recover 2^{15} possible states
  - For each possible state, generate a value and test against last 2 bytes

32-Byte Nonces



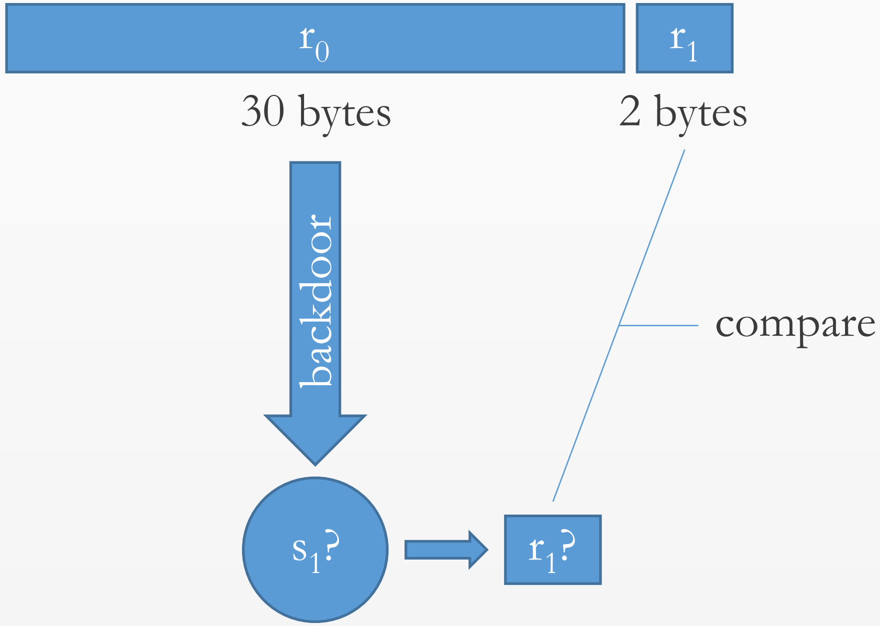
- 32-byte Dual EC outputs actually facilitate the attack:
  - Use first 30 bytes to recover 2^{15} possible states
  - For each possible state, generate a value and test against last 2 bytes

32-Byte Nonces



- 32-byte Dual EC outputs actually facilitate the attack:
  - Use first 30 bytes to recover 2^{15} possible states
  - For each possible state, generate a value and test against last 2 bytes

32-Byte Nonces



- 32-byte Dual EC outputs actually facilitate the attack:
  - Use first 30 bytes to recover 2^{15} possible states
  - For each possible state, generate a value and test against last 2 bytes
- Results in 1-3 possible states in practice
- Attack is impractical with 20-byte nonce

Version History

- ScreenOS 6.1.0r7 (last 6.1 version)
  - ANSI x9.31
    - Seeded by Interrupts
    - Reseeds every 10k calls
  - DH Queues
  - 20-byte IKE nonces

- ScreenOS 6.2.0r0 (first 6.2 version)
  - DualEC → ANSI x9.31
    - Reseed Bug exposes DualEC
    - Reseeds every call
  - Nonce Queues before DH Queues
  - 32-byte nonces

Attacker changed constant in 6.2.0r15

```
      5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B
      6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C296
      FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551
bad:  9585320EEAF81044F20D55030A035B11BECE81C785E6C933E4A8A131F6578107
good: 2c55e5e45edf713dc43475effe8813a60326a64d9ba3d2e39cb639b0f3b0ad10
```

https://cseweb.ucsd.edu/~hovav/dist/rwc16.pdf

Version History

- ScreenOS 6.1.0r7 (last 6.1 version)
  - ANSI x9.31
    - Seeded by Interrupts
    - Reseeds every 10k calls
  - DH Queues
  - 20-byte IKE nonces

- ScreenOS 6.2.0r0 (first 6.2 version)
  - DualEC → ANSI x9.31
    - Reseed Bug exposes DualEC
    - Reseeds every call
  - Nonce Queues before DH Queues
  - 32-byte nonces

Completely Passive Attack

Enabled in single point release

Juniper's "fix" was to reinstate their original Q value. After our work, they removed Dual EC completely.

Answers

- ~~Why does a change in Q result in a passive VPN Decryption vulnerability?~~
- ~~We doesn't Juniper's use of X9.31 protect their system against a compromise of Q?~~
- ~~What is the history of the PRNG code in ScreenOS?~~
- How was Juniper's Q value generated?
- ~~Is the version of ScreenOS with Juniper's Q vulnerable to attack?~~

Questions?

ScreenOS Timeline

- 6.1.0r7 – ANSI generator
- 6.2.0r1 – DualEC with bugs and Juniper's Q
- 6.2.0r15 – Q changed to unknown attacker's value (12 Sept. 2012)
- 6.3.0r17 – SSH Backdoor introduced (25 April 2014?)
- 6.3.0r19b and 6.3.0.r12b – Rebuilt with backdoors removed (Dec. 2015)
- 6.3.0r22 – Dual EC removed and replaced (April 2016)