IEEE P802.11
Wireless LANs

|  |
| --- |
| Addressing A Side-Channel Attack on SAE |
| Date: 2014-05-12 |
| Author(s): |
| Name | Affiliation | Address | Phone | email |
| Dan Harkins | Aruba Networks | 1322 Crossman ave, Sunnyvale, CA 94089 | +1 408 227 4500 | dharkins at aruba networks dot com |
|  |  |  |  |  |

Abstract

The “hunting and pecking” technique used by SAE to find a password element can be open to side channel attack. This submission addresses that problem in a backwards-compatible manner.

***Modify section 3.3 as indicated:***

LDPC low-density parity check

LED light-emitting diode

LGR legendre symbol

LFSR linear feedback shift register

LLC logical link control

***Modify section 11.3.4.2.2 as indicated:***

In order to minimizethe possibility of side-channel attacks that attempt to determine the number of

interactions of the “hunting-and-pecking” loop required for a given <password, STA-A-MAC, STA-BMAC>

tuple, implementations should perform at least k iterations regardless of whether PWE is discovered

or not. The value k may be set to any non-negative value and should be set to a sufficiently large number to

effectively guarantee the discovery of PWE in less than k iterations. If PWE is discovered in less than k

iterations a random “password” can be used in subsequent iterations to further obfuscate the true cost of

discovering PWE .

NOTE—The probability that one requires more than n iterations of the “hunting and pecking” loop to find PWE is

roughly (r/2p)n, which rapidly approaches zero (0) as n increases.

Algorithmically this process is described as follows:

found = 0;

counter = 1

z = len(p )

base = password

do {

pwd -seed = H(MAX(STA-A-MAC, STA-B-MAC) || MIN(STA-A-MAC, STA-B-MAC),

base || counter )

pwd -value = KDF-z(pwd -seed , “SAE Hunting and Pecking”, p )

if (pwd-value < p )

then

 if y2 = pwd-value3 + a\*pwd-value + b is a quadratic residue modulo p

then

if (found = 0)

then

x = pwd-value

save = pwd-seed

found = 1

base = new-random-number

fi

fi

fi

counter = counter + 1

} while ((counter <= k) or (found=0))

y = sqrt(x3 + ax + b) modulo p

if (LSB(save) = LSB(y)

then

 PWE = (x,y)

else

 PWE= (x, p-y)

fi

Checking whether a value is a quadradic residue modulo a prime can leak information that can be used in launching a side-channel attack. Therefore, to further address the possibility of a side-channel attack, it is recommended that a blinding technique be used in determining the quadratic residocity of a value modulo a prime.

The blinding technique involves multiplication of the value with a random number so the value being checked for quadratic residocity modulo a prime can take on all numbers between 1 and p-1 with equal probability. The blinded value is multiplied by a quadratic residue or quadratic non-residue depending on the value of a coin flip and the result is checked whether the result is a quadratic residue or quadratic non-residue, respectively.

This technique involves creation of a quadratic residue, *qr*, and quadratic non-residue, *qnr*, prior to beginning of the hunting-and-pecking loop. These values can be chosen at random by checking their legendre symbol:

do {

qr = random()

while ( LGR(qr, p) == -1)

do {

qnr = random()

while ( LGR(qnr, p) == 1)

The blinding technique of determining whether a value, *v*, is a quadratic residue modulo a prime, *p*, is then:

is\_quadratic\_residue (v, p) {

r = (random() modulo (p – 1) + 1

num = (v \* r \* r) modulo p

if (LSB(r) = 1)

then

num = (num \* qr) modulo p

if (LGR(num, p) = 1)

then

return true

fi

else

num = (num \* qnr) modulo p

if (LGR(num, p) == -1)

then

return true

fi

fi

return false

}

The values *qr* and *qnr* may be used for all loops in the hunting-and-pecking process but a new value for *r* must be generated each time a quadratic residue is checked.**References:**

draft-irtf-cfrg-dragonfly-04.txt