

CVE DEMO

```
1. Sep 15:53 ..
0. Sep 2015 bin -> usr/bin
19. Sep 09:31 boot
21. Sep 15:50 dev
19. Sep 09:32 etc
21. Sep 15:52 home
7 30. Sep 2015 lib -> usr/lib
84 23. Jul 2015 lib64 -> usr/lib
96 1. Aug 10:01 lost+found
096 30. Sep 2015 mnt
16 21. Sep 15:52 opt
4096 12. Aug 08:15 private -> /home/encrypted
560 21. Sep 15:37 proc
7 30. Sep 15:50 root
4096 30. Sep 2015 run
0 21. Sep 15:51 sbin -> usr/bin
300 21. Sep 15:45 srv
4096 12. Aug 15:39 sys
1a 4096 23. Jul 10:25 tmp
root 4096 21. Sep 15:53
root 4096 21. Sep 15:53
root 4096 21. Sep 15:53
```



Jonathan Bateman, Hassan Alshehri, Harrison Tarsia

Why is CVE-2011-4354 important?

- OpenSSL version 0.9.8g
- Implementation vs Efficiency → The Programmers Dilemma
- TLS v1.2 → Why it enables this attack?
 - “Change Cipher Spec” comparison Left (1.2) vs Right (1.3)

660	2024/857	181:87:28.966738	54.167.185.228	129.21.104.128	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
661	2024/857	181:87:28.966738	54.167.185.228	129.21.104.128	TLSv1.2	117	Change Cipher Spec, Encrypted Handshake Message
662	2024/857	181:87:28.966738	54.167.185.228	129.21.104.128	TLSv1.2	132	Application Data
Frame 668: 192 bytes on wire (1536 bits), 192 bytes captured (1536 bits) on interface enb, id 8							
Ethernet II, Src: Apple-7a:dd:ee (88:ee:83:79:de:ed), Dst: Icy-Woan-MD_81 (88:00:5e:00:01:81)							
Internet Protocol Version 4, Src: 129.21.104.128, Dst: 54.167.185.228							
Transmission Control Protocol, Src Port: 50987, Dst Port: 443, Seq: 518, Ack: 5389, Len: 126							
Transport Layer Security							
TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange							
Content Type: Handshake (22)							
Version: TLS 1.2 (0x0303)							
Length: 70							
Handshake Protocol: Client Key Exchange							
Handshake Type: Client Key Exchange (16)							
Length: 66							
EC Diffie-Hellman Client Params							
Pubkey Length: 65							
Pubkey: 0413c472d2de11d6f729317dbd63c3c086aeae577b8d749bc481f17c15fa4776cf13246326c9f6c844e0844f59c9e58a5440a53428e4272249ab							
TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec							
Content Type: Change Cipher Spec (20)							
Version: TLS 1.2 (0x0303)							
Length: 1							
Change Cipher Spec Message							
TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message							
Content Type: Handshake (22)							
Version: TLS 1.2 (0x0303)							
Length: 40							
Handshake Protocol: Encrypted Handshake Message							

1452	2024/05/7	18:07:41.345099	129.21.104.128	54.236.104.103	TLV51.3	577 Client Hello (SNI=s-lackb.com)
1455	2024/05/7	18:07:41.369894	129.21.104.128	54.236.104.103	TLV51.3	1514 Server Hello, Change Cipher Spec
1458	2024/05/7	18:07:41.369897	54.236.104.103	129.21.104.128	TLV51.3	167 Application Data
1459	2024/05/7	18:07:41.369938	2600:9000:2514:760...	2620:8d:8000:1068:...	TLV51.2	125 Application Data
1467	2024/05/7	18:07:41.375802	129.21.104.128	54.236.104.103	TLV51.3	130 Change Cipher Spec, Application Data
1470	2024/05/7	18:07:41.380919	129.21.104.128	54.236.104.103	TLV51.3	1514 Application Data, Application Data
1474	2024/05/7	18:07:41.381013	129.21.104.128	54.236.104.103	TLV51.3	1032 Application Data
1477	2024/05/7	18:07:41.400424	54.236.104.103	129.21.104.128	TLV51.3	1593 Application Data, Application Data
1479	2024/05/7	18:07:41.403968	129.21.104.128	54.236.104.103	TLV51.3	97 Application Data
1480	2024/05/7	18:07:41.441669	44.194.124.151	129.21.104.128	TLV51.2	971 Application Data
1489	2024/05/7	18:07:41.448715	54.236.104.103	129.21.104.128	TLV51.3	409 Application Data
1491	2024/05/7	18:07:41.466091	44.194.124.151	129.21.104.128	TLV51.2	97 Encrypted Alert
1495	2024/05/7	18:07:41.562622	2607:f80e:4006:806...	2620:8d:8000:1068:...	TLV51.2	159 Application Data
1498	2024/05/7	18:07:41.566139	2607:f80e:4006:806...	2620:8d:8000:1068:...	TLV51.2	159 Application Data
1568	2024/05/7	18:07:43.639000	2607:f80e:4006:806...	2620:8d:8000:1068:...	TLV51.2	159 Application Data

```

> Frame 1467: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface em0, id 0
  Ethernet II, Src: Apple79:66:e0 (b0:be:83:79:66:e0), Dst: IETF-VRRP-VRID_01 (00:00:5e:00:01:01)
  > Internet Protocol Version 4, Src: 129.21.104.128, Dst: 54.236.104.103
  > Transmission Control Protocol, Src Port: 58989, Dst Port: 443, Seq: 1960, Ack: 4446, Len: 64
  > Transport Layer Security
    > TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
    > TLSv1.3 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
      Opaque Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 53
      Encrypted Application Data: 4fe8a61a3015e0a967c72b79524d2028e7abc46a368a074bacbb1a0561f2bc2637932c41f8aa898cf3d36d74791c7f61aa01a3d
      [Application Data Protocol: Hypertext Transfer Protocol]

```

ECDHE Overview

- k_C^i & $k_S^i \rightarrow$ private keys $\{1 \rightarrow (n-1)\}$
- $G \rightarrow$ generator
 - (initial elliptic crv point)
- Q_C^i & $Q_S^i \rightarrow$ public keys (over net)
 - $Q = [k]G$ (private key * generator)
- bug?

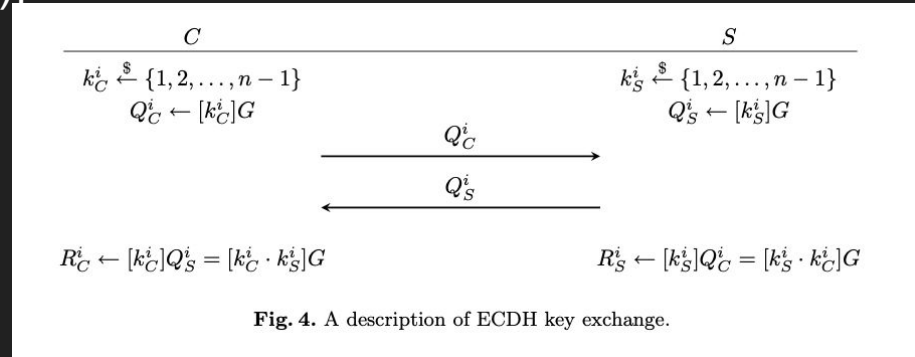


Fig. 4. A description of ECDH key exchange.

- R_C^i & $R_S^i = [k_S^i * k_C^i]G \rightarrow$ multiply personal private key by opposing public key
 - Shared secret \rightarrow both client and server have it w/o each other's private key
- Attacker: $Q_S^i Q_C^i = k_S^i * k_C^i * G * G \neq [k_S^i * k_C^i]G \rightarrow$ Very different

Attack Angle → blame devs who prioritize optimization



k_s^i
Stays the
same!

Faulty Algorithm

- The algorithm is used to compute the result of a modular reduction making sure that the quotient is exact and getting rid of the possibility of overflow.
- The problem is the logic of the algorithm is flawed.
- The algorithm is built to handle modular reduction of S by p where c is the exact quotient of the division.
- It is assumed that $S = t + c \cdot 2^{256}$ and the algorithm tries to compute the quotient $q = S/p$ based on the value of c .
- This incorrect reasoning produces erroneous results.

```

485  nist_cp_bn_0(buf, a_d + BN_NIST_256_TOP, top - BN_NIST_256_TOP, BN_NIST_256_TOP);
486
487  /*S1*/
488  nist_set_256(t_d, buf, 15, 14, 13, 12, 11, 0, 0, 0);
489  /*S2*/
490  nist_set_256(t_d2, buf, 0, 15, 14, 13, 12, 0, 0, 0);
491  if (bn_add_words(t_d, t_d, t_d2, BN_NIST_256_TOP))
492    carry = 2;
493  /* left shift */
494  {
495    register BN_ULONG *ap, t, c;
496    ap = t_d;
497    c = 0;
498    for (i = BN_NIST_256_TOP; i != 0; --i)
499    {
500      t = *ap;
501      *(ap++) = ((t << 1) | c) & BN_MASK2;
502      c = (t & BN_TBIT) ? 1 : 0;
503    }
504    if (c)
505      ++carry;
506  }
507
508  if (bn_add_words(r_d, r_d, t_d, BN_NIST_256_TOP))
509    ++carry;
510  /*S3*/
511  nist_set_256(t_d, buf, 15, 14, 0, 0, 0, 10, 9, 8);
512  if (bn_add_words(r_d, r_d, t_d, BN_NIST_256_TOP))
513    ++carry;
514  /*S4*/
515  nist_set_256(t_d, buf, 8, 13, 15, 14, 13, 11, 10, 9);
516  if (bn_add_words(r_d, r_d, t_d, BN_NIST_256_TOP))
517    ++carry;
518  /*D1*/
519  nist_set_256(t_d, buf, 10, 8, 0, 0, 0, 13, 12, 11);
520  if (bn_sub_words(r_d, r_d, t_d, BN_NIST_256_TOP))
521    carry;
522  /*D2*/
523  nist_set_256(t_d, buf, 11, 9, 0, 0, 15, 14, 13, 12);
524  if (bn_sub_words(r_d, r_d, t_d, BN_NIST_256_TOP))
525    --carry;
526  /*D3*/
527  nist_set_256(t_d, buf, 12, 0, 10, 9, 8, 15, 14, 13);
528  if (bn_sub_words(r_d, r_d, t_d, BN_NIST_256_TOP))
529    --carry;
530  /*D4*/
531  nist_set_256(t_d, buf, 13, 0, 11, 10, 9, 0, 15, 14);
532  if (bn_sub_words(r_d, r_d, t_d, BN_NIST_256_TOP))
533    --carry;
534
535  if (carry)
536  {
537    if (carry > 0)
538      bn_sub_words(r_d, r_d, _256_data + BN_NIST_256_TOP,
539        --carry, BN_NIST_256_TOP);
540    else
541    {
542      carry = -carry;
543      bn_add_words(r_d, r_d, _256_data + BN_NIST_256_TOP +
544        --carry, BN_NIST_256_TOP);
545    }
546  }
547
548  r->top = BN_NIST_256_TOP;
549  bn_correct_top(r);
550  if (BN_ucmp(r, field) >= 0)
551  {
552    bn_sub_words(r_d, r_d, _nist_p_256, BN_NIST_256_TOP);
553    bn_correct_top(r);
554  }
555  bn_check_top(r);

```

The intention is to eliminate any possibility of overflow; the assumption is that c is the exact quotient of division of S by p .

The reasoning behind the faulty algorithm is that if one writes $S = t + c \cdot 2^{256}$, then the exact quotient $q = S \div p$ is given by

1. if $c \geq 0$, then $q = c$ or $q = c + 1$,
2. if $c < 0$, then $q = c$ or $q = c - 1$

since c is small. Indeed, write $\Delta = 2^{256} - p$, then after subtracting $c \cdot p$ we obtain

$$S - c \cdot p = t + c \cdot 2^{256} - c \cdot p = t + c \cdot \Delta.$$

Since $-4 \leq c \leq 6$ and $\Delta < 2^{224}$, this shows the result is bounded by $-p < t + c \cdot \Delta < 2p$. The faulty algorithm therefore computes an incorrect result in the following cases:

- If $c \geq 0$, the algorithm fails when $t + c \cdot \Delta \geq 2^{256}$ since it computes r' only modulo 2^{256} and not as a full integer (for which the resulting algorithm would have been correct). Note that in this case the correct result would be $r' + \Delta$ and that modulo p , the correct result thus is $r' + 2^{256} \pmod{p}$.
- If $c < 0$, the algorithm fails when $t + c \cdot \Delta < 0$. The correct result then depends on whether $(t + c \cdot \Delta) \bmod 2^{256} \geq p$ or not: in the former case, the correct result is $r' - \Delta$, whereas in the latter case, the correct result is given by $r' + 2^{256} - 2\Delta$. Note that although there are two different subcases for $c < 0$, the errors $-\Delta$ and $2^{256} - 2\Delta$ are congruent modulo p , i.e. modulo p , the correct result is given by $r' - 2^{256} \pmod{p}$.

Fig. 3. A program fragment demonstrating the faulty OpenSSL modular reduction algorithm for P-256.

The Fix

- Fix in version 0.9.8r
- The issues in the algorithm are associated with the modular addition and subtraction operations.
- The algorithm involves multiple modular additions (bn_add_words) and modular subtractions (bn_sub_words).
- The fixes to the addition and subtraction are shown here on the right.
- Utilizes bitwise operations for efficient function selection based on 'carry'.

```
626 u.f = bn_sub_words;
627 if (carry > 0)
628     carry = (int)bn_sub_words(r_d,r_d,_nist_p_256[carry-1],BN_NIST_256_TOP);
629 else if (carry < 0)
630     {
631         carry = (int)bn_add_words(r_d,r_d,_nist_p_256[-carry-1],BN_NIST_256_TOP);
632         mask = 0-(size_t)carry;
633         u.p = ((size_t)bn_sub_words&mask) | ((size_t)bn_add_words&~mask);
634     }
635 else
636     carry = 1;
637
638 mask = 0-(size_t)(*u.f)(c_d,r_d,_nist_p_256[0],BN_NIST_256_TOP);
639 mask &= 0-(size_t)carry;
640 res = (BN_ULONG *)(((size_t)c_d&~mask) | ((size_t)r_d&mask));
641 nist_cp_bn(r_d, res, BN_NIST_256_TOP);
642 r->top = BN_NIST_256_TOP;
643 bn_correct_top(r);
644
645 return 1;
646 }
```

- The `SSL_OP_SINGLE_ECDH_USE` option is used to create a new key every single time when using temporary/ephemeral ECDH parameters.
- The paper states that this option must be turned on otherwise the server will “opt out” and use a static key pair for each “application instance”.
- With the option turned on, a new key pair will be generated per “handshake instance” helping to prevent the attack.

The Fix

When activated, the optimisation means a single ECDH key pair is generated during initialisation of the OpenSSL context within an application. This key pair is reused for all protocol executions thereafter; with ephemeral-static ECDH, OpenSSL has the server use a static key (i.e., a fixed k_i^S and hence Q_i^S for all i) for each OpenSSL context. Put another way, the key pair is ephemeral for each application instance and not (necessarily) per handshake instance. While this preserves forward secrecy between application instances, it violates forward secrecy within a single application instance when performing more than a single protocol execution. Interestingly, the default behaviour is the latter: to “opt out” and disable the optimisation, the application must explicitly use the `SSL_OP_SINGLE_ECDH_USE` option during initialisation of the context.

Other Changes

Takanori Yanagisawa has shown how to correctly use pre-computed values. · openssl/openssl@830b887 · GitHub

- Table structure
- Modification in the Reduction Process ('if (a>>256)')
- Improvements in carry handling
- Boundary checks

Changes are aimed at handling edge cases and improving efficiency

Fix remaining BN_nist_mod *. · openssl/openssl@299ab42 · GitHub

- Function parameter usage

Modified code:

```
'BN_nist_mod_256(BN_ULONG r[4],  
const BN_ULONG a[6][4], const  
BN_ULONG p[6][4], BN_CTX *ctx)'
```

The BN_CTX parameter indicates change in how the context is handled during modular reduction