rya.nc

Ryan Castellucci's blog

Stupid certificate tricks

Sometimes I do things for no real reasons other than "because I can" and/or "it amuses me". For example, embedding a snarky message into my HTTPS certificate.

Before I get into how this works, take a moment to admire the results:

```
$ openssl s_client -connect rya.nc:443 2>/dev/null</dev/null | sed -n
/BEGIN/,/END/p | egrep --color '^//.*|$'
-----BEGIN CERTIFICATE-----
MIIFIjCCBAqgAwIBAgIDFZnFMA0GCSqGSIb3DQEBBQUAMDwxCzAJBgNVBAYTAlVT
MRcwFQYDVQQKEw5HZW9UcnVzdCwgSW5jLjEUMBIGA1UEAxMLUmFwaWRTU0wgQ0Ew
HhcNMTQxMDA2MDQzMjU1WhcNMTUxMDExMDk0MzA0WjCBuTEpMCcGA1UEBRMgaWcv
T0FzbjFySm5Bd3lzNURwL0g2LUF4UzIvNDJsOC0xEzARBgNVBAsTCkdUODg5OTk4
NTkxMTAvBgNVBAsTKFNlZSB3d3cucmFwaWRzc2wuY29tL3Jlc291cmNlcy9jcHMg
KGMpMTQxLzAtBgNVBAsTJkRvbWFpbiBDb250cm9sIFZhbGlkYXRlZCAtIFJhcGlk
U1NMKFIpMRMwEQYDVQQDEwp3d3cucnlhLm5jMIIBIjANBgkqhkiG9w0BAQEFAAOC
AQ8AMIIBCgKCAQEApiiX3wNKiyVkpjz3hlVgacTpn5Du6q8JwECCIUd3j3Fs1yXo
//When/cryptography/is/outlawed/bayl/bhgynjf/jvyy/unir/cevinpl//
/6u56qRe3fvqVb9EkdZqtgtYv6akC5s5t3BoPFyrM3UEMugrAX7q6EGPl4k2kWgz
HhLq4IrRENGileaWuLrkuEIyqwattiM8DGm9tqOlnWZ5zBRcEpfviZKLRrdQncSS
ZqtfXA7HWBIPluLDIUgM1YRlfiiTvATAL7DrqNqWKIlsq7JZe6jnCkuRJoR2a0BA
guDEul/ksF351jTHPc5pFiVGeFL13D7vDO0KpwIDAQABo4IBrTCCAakwHwYDVR0j
BBgwFoAUa2k9ahhCSt2PAmU5/TUkhniRFjAwDgYDVR0PAQH/BAQDAgWgMB0GA1Ud
JQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjAdBgNVHREEFjAUggp3d3cucnlhLm5j
ggZyeWEubmMwQwYDVR0fBDwwOjA4oDagNIYyaHR0cDovL3JhcGlkc3NsLWNybC5n
ZW90cnVzdC5jb20vY3Jscy9yYXBpZHNzbC5jcmwwHQYDVR0OBBYEFIPyKl6gmhN9
2byRrgbuvFN0U+p+MAwGA1UdEwEB/wQCMAAweAYIKwYBBQUHAQEEbDBqMC0GCCsG
AQUFBzABhiFodHRwOi8vcmFwaWRzc2wtb2NzcC5nZW90cnVzdC5jb20wOQYIKwYB
BQUHMAKGLWh0dHA6Ly9yYXBpZHNzbC1haWEuZ2VvdHJ1c3QuY29tL3JhcGlkc3Ns
```

```
LmNydDBMBgNVHSAERTBDMEEGCmCGSAGG+EUBBzYwMzAxBggrBgEFBQcCARYlaHR0
cDovL3d3dy5nZW90cnVzdC5jb20vcmVzb3VyY2VzL2NwczANBgkqhkiG9w0BAQUF
AAOCAQEAYCxc5LD/M7tz54pxEvluYHX/peL0u7KaKRNPrVXrAqAVsu4oeO6egXga
zqOrICSzCIkgdo4BhBOelLKj+GJgrWUI+p0NWZkL1zhgOdZw+AVapDEkuXt27Wgg
WXyVjR8XPDf3ZXP651+Rthk+pMfdofX8SWOyWPFg94KxBYqG9/v4XQxsEBY8D/m4
ZHDY1nnUwWsnr7NfiZATZRs2SV67yVYGcFz4kK+AY5gcFYpbhDMnMrnBD7tqWw2Y
KkbnUrnichGuJlDg8R8fdxIWF8y8WnT8g7kV37nrYdvVAIhpx7okeayAkqBLVfUY
9oK928THRTdSkrqBpEuqH6j6geT11g==
-----END CERTIFICATE-----
```

In case you're wondering, "bayl bhgynjf jvyy unir cevinpl" is "only outlaws will have privacy" run through ROT13.

This is a the PEM (Base64) formatted representation of the certificate. Decoded, it represents the following:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1415621 (0x1599c5)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=GeoTrust, Inc., CN=RapidSSL CA
        Validity
            Not Before: Oct  6 04:32:55 2014 GMT
            Not After : Oct 11 09:43:04 2015 GMT
        Subject: serialNumber=ig/OAsn1rJnAwys5Dp/H6-AxS2/42l8-,
                 OU=GT88999859, OU=See www.rapidssl.com/resources/cps
(c)14,
                 OU=Domain Control Validated - RapidSSL(R),
CN=www.rya.nc
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a6:28:97:df:03:4a:8b:25:64:a6:3c:f7:86:55:
                    60:69:c4:e9:9f:90:ee:ea:af:09:c0:40:82:21:47:
                    77:8f:71:6c:d7:25:e8:ff:f5:a1:7a:7f:dc:af:2a:
                    6d:a2:0a:da:a6:1c:bf:8a:cf:e8:ba:d9:5a:c1:e7:
                    7f:6d:ac:a5:fd:b8:60:ca:78:df:fe:3b:f2:cb:fb:
                    a7:8a:bf:dc:7a:f8:a7:a6:5f:ff:ff:ab:b9:ea:a4:
                    5e:dd:fb:ea:55:bf:44:91:d6:6a:b6:0b:58:bf:a6:
```

```
                    a4:0b:9b:39:b7:70:68:3c:5c:ab:33:75:04:32:e8:
                    2b:01:7e:ea:e8:41:8f:97:89:36:91:68:33:1e:12:
                    ea:e0:8a:d1:10:d1:a2:95:e6:96:b8:ba:e4:b8:42:
                    32:ab:06:ad:b6:23:3c:0c:69:bd:b6:a3:a5:9d:66:
                    79:cc:14:5c:12:97:ef:89:92:8b:46:b7:50:9d:c4:
                    92:66:ab:5f:5c:0e:c7:58:12:0f:96:e2:c3:21:48:
                    0c:d5:84:65:7e:28:93:bc:04:c0:2f:b0:eb:a8:da:
                    96:28:89:6c:ab:b2:59:7b:a8:e7:0a:4b:91:26:84:
                    76:6b:40:40:82:e0:c4:ba:5f:e4:b0:5d:f9:d6:34:
                    c7:3d:ce:69:16:25:46:78:52:f5:dc:3e:ef:0c:ed:
                    0a:a7
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:

keyid:6B:69:3D:6A:18:42:4A:DD:8F:02:65:39:FD:35:24:86:78:91:16:30

            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client
Authentication
            X509v3 Subject Alternative Name:
                DNS:www.rya.nc, DNS:rya.nc
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://rapidssl-crl.geotrust.com/crls/rapidssl.crl

            X509v3 Subject Key Identifier:

83:F2:2A:5E:A0:9A:13:7D:D9:BC:91:AE:06:EE:BC:53:74:53:EA:7E
            X509v3 Basic Constraints: critical
                CA:FALSE
            Authority Information Access:
                OCSP - URI:http://rapidssl-ocsp.geotrust.com
                CA Issuers - URI:http://rapidssl-
aia.geotrust.com/rapidssl.crt

            X509v3 Certificate Policies:
                Policy: 2.16.840.1.113733.1.7.54
                  CPS: http://www.geotrust.com/resources/cps

    Signature Algorithm: sha1WithRSAEncryption
        60:2c:5c:e4:b0:ff:33:bb:73:e7:8a:71:12:f9:6e:60:75:ff:
        a5:e2:f4:bb:b2:9a:29:13:4f:ad:55:eb:02:a0:15:b2:ee:28:
        78:ee:9e:81:78:1a:ce:a3:ab:20:24:b3:08:89:20:76:8e:01:
        84:13:9e:94:b2:a3:f8:62:60:ad:65:08:fa:9d:0d:59:99:0b:
        d7:38:60:39:d6:70:f8:05:5a:a4:31:24:b9:7b:76:ed:68:20:
        59:7c:95:8d:1f:17:3c:37:f7:65:73:fa:e7:5f:91:b6:19:3e:
        a4:c7:dd:a1:f5:fc:49:63:b2:58:f1:60:f7:82:b1:05:8a:86:
        f7:fb:f8:5d:0c:6c:10:16:3c:0f:f9:b8:64:70:d8:d6:79:d4:
        c1:6b:27:af:b3:5f:89:90:13:65:1b:36:49:5e:bb:c9:56:06:
        70:5c:f8:90:af:80:63:98:1c:15:8a:5b:84:33:27:32:b9:c1:
        0f:bb:6a:5b:0d:98:2a:46:e7:52:b9:e2:72:11:ae:26:50:e0:
        f1:1f:1f:77:12:16:17:cc:bc:5a:74:fc:83:b9:15:df:b9:eb:
        61:db:d5:00:88:69:c7:ba:24:79:ac:80:92:a0:4b:55:f5:18:
        f6:82:bd:db:c4:c7:45:37:52:92:ba:81:a4:4b:aa:1f:a8:fa:
        81:e4:f5:d6
```

The section of the modulus containing the message is highlighted. The finer points of RSA are beyond the scope of this post, but it's important to know that the modulus, *n*, is the product of two secret prime numbers of the same length, *p* and *q*. Normally they would be chosen at random, but in this case that's been tweaked.

```python
import gmpy
from Crypto.PublicKey import RSA
from binascii import hexlify, unhexlify
from base64 import b64encode as b64e, b64decode as b64d

def replace_at(orig, replace, offset):
    return orig[0:offset] + replace + orig[offset+len(replace):]

msg_b64 = '//When/cryptography/is/outlawed/bayl/bhgynjf/jvyy/unir/cevinp
msg_bytes = b64d(msg_b64)

# key parameters
keybits, e = 2048, 65537

while True:
    # generate initial random key
    rsa = RSA.generate(keybits)
    # get modulus as bytes
    n_bytes = unhexlify(str(hex(rsa.n))[2:-1])
    p = rsa.p

    # splice in the message, may need to play
    # with the offset for it to come out right
    n_tmp_bytes = replace_at(n_bytes, msg_bytes, 36)

    # convert the modulus bytes back into an integer
    n_tmp = gmpy.mpz(hexlify(n_tmp_bytes), 16)

    # value to start search for a new prime q from
    q_tmp = n_tmp / p
    q_new = q_tmp.next_prime()

    # recompute key components based on new q
    n_new = p * q_new
    phi = (p-1) * (q_new-1)
    # let the loop repeat until phi is coprime to e, then compute d
    if gmpy.gcd(e, phi) == 1:
        d = gmpy.invert(e, phi)
        rsa = RSA.construct((long(n_new), long(e), long(d), long(p), lon
        print rsa.exportKey()
        break
```

The original p is kept, n_{tmp} is created by splicing the encoded message into n, and $q_{tmp} = n_{tmp} \div p$. The problem is that q_{tmp} is vanishingly unlikely to be an integer, let alone a prime number. This is resolved by finding q_{new}, the first prime number larger than q_{tmp}, using the `next_prime` function from Python's gmpy library. When $n_{new} = p \cdot q_{new}$ is computed, it will differ from n_{tmp} by $p \cdot (q_{new} - q_{tmp})$. Since the gaps between prime numbers are relatively small even for very large numbers, only the last half of the digits (give or take a few) of the modulus will be disturbed by the correction, leaving the message intact so long as it wasn't too long. If you're confused as to why this is so, think back to the multiplication method you probably learned as a child.

Running the code gives an RSA private key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA8JyOXdiRCVOZmoEF9k01100IS7t2Uci9OrsXG6Nbs2RAQXBw
//When/cryptography/is/outlawed/bayl/bhgynjf/jvyy/unir/cevinpl//
D9F96ghtF/AT5N8BCaodDKv4oAkuH9Vxl/BvYCkjBMRY3LurPCjRfylHTQRH0Zgu
wL3u8QuP24xTbx8kuRBT4mdRxLD6QEwJaF+ObYbw5ShxEJpgFmXqi3woGIHQCP1Y
p+wmSMHbS5SIGDnOdLAWbMfVZC+IE/z00VSPrl0R+Vt1LBzNML9Mll1Uot2Uln5G
wBScJtUpqSr9Dei3d170y9GZW0JVDzVe+4XEmwIDAQABAoIBAQDjl8LWpC50uv41
dkvUca43DGeHczf1HkNYFXZDL19jLbXV8G0CwC5RODbf/esb9tZhgBnyTL1gWI6P
kdEoRcHxYAE2I+YEjmIYbt9I0DjWnPO/3Vffd5J52CSRGwdGW2aY5K97uAOCJYza
kcRUKxq+w8qbDLrdeCr9ycJ4XOxTvJ4WAuQUB8MlOWgtpNS1bFRZ+tzndMhCSvWv
E4656RZf7W/iQKcBaDlPAfdcf2iUt7ZaXv3cKSygaJ2srpmx0vSd1gWK3wDK/xrt
nsXZoJBuXcIDSDGy7t0dcG3MHXDVCZmdvEYRUXmyoe+WoW0XEwak6/FmcTgad+Rz
WrKZf/uBAoGBAPEXwTG+SZ9fvz9Byc6EJX+LEOXBvYpVXOw17PymMK3HEGI3UBNJ
6yAViNWNqpXE928zrEZv+f+f62Heuf+WQ3DQVMdM1QxFpj5RVf5WievpqUaxGjQJ
yKU/x63oiqjNPogYreT5HN56ljsh7pJJ68EgCDmHIt55A8NEn3rR7TbbAoGBAP99
LxGhEN5ZgjWBgrcGmR7CRoOWhSJFGygpCGYKCynOfg66VLqmuoEijVAAgxmZo8u8
mGSg+6IuwsA335tkBiOgLMMvNZE5hyljqeSVlTkCaR7ZbEkHiIpKlQg7+N3NEBpf
t7qWLZMhASGCpuzwET1rrMAbTARlia34j2IWJrVBAoGBAK4ltYRj6jQ36iIcOFR3
OcrePe9oOawxqvRoo21/8gukjd4UDEBSlYdQZs2zDfQvGXf2wEsE2XVfI5xHUN0g
wkg8A/EOO5oouUOsZsxX4DpLRt3sUXwjUQ6kemzRW09BKhkOkpWhp8vAisHd6cE7
qhKPO8GqLnK6wRAMgpIqDwofAoGAVGdz7FwMqZhihvCxUWvxnBLMnt5UP10bOqpL
pwI8a+RXCuCN61f3l3/ltX9l0EhMr5svsVbpqsvN9RjAW6Kw0IYzI4xuIvshZxAQ
6X5tXPcp6VIlDv9ZIW7AS4cckZIUdtIWbaL9jXTC3eI+6VnqKCNxX8nk1DMDSCEs
pVfyE8ECgYBCHuv1NmZDWvV8xhYC8wk0M7kekZriH2s1MXolb76yjd0Nusf0BX1o
F0+HzV6BfxioV6Jk6sggfM0aO1q0B1ZxBO3Y2g4NMPpcHu8BDnpTKPQy60N5ifv7
Y8ZQbDunXhEAdobNhqwe8pDNVyI/KwrX5efjAeQUwlcCYvEMtVMo6A==
-----END RSA PRIVATE KEY-----
```

A CSR (Certificate Signing Request) can be generated from the private key like this:

```
$ openssl req -new -key tmp.key -subj '/C=US/CN=www.example.com'
-----BEGIN CERTIFICATE REQUEST-----
MIICbDCCAVQCAQAwJzELMAkGA1UEBhMCVVMxGDAWBgNVBAMMD3d3dy5leGFtcGxl
LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAPCcjl3YkQlTmZqB
BfZNNddNCEu7dlHIvTq7FxujW7NkQEFwcP/1oXp/3K8qbaIK2qYcv4rP6LrZWsHn
f22spf24YMp43/478sv7p4q/3Hr4p6Zf/w/RfeoIbRfwE+TfAQmqHQyr+KAJLh/V
cZfwb2ApIwTEWNy7qzwo0X8pR00ER9GYLsC97vELj9uMU28fJLkQU+JnUcSw+kBM
CWhfjm2G8OUocRCaYBZl6ot8KBiB0Aj9WKfsJkjB20uUiBg5znSwFmzH1WQviBP8
9NFUj65dEflbdSwczTC/TJZdVKLdlJZ+RsAUnCbVKakq/Q3ot3de9MvRmVtCVQ81
XvuFxJsCAwEAAaAAMA0GCSqGSIb3DQEBBQUAA4IBAQA42dokgfBLUVXLw5hqicwe
```

```
vTvvZZqizWCjHLpqiOGLOJzQ4DU2OcNadae4KDNZpJ2jEMRUK+zaG91Qq+++YN/E
9dkMDZ10DQQuisyZelidDP/ppifNmFxJzRQnyHEFrM5JfxBHxZIBNJ/PXxTSQfxS
qgNj/2W4PHd0kvMPv9DU/xqmy4eVkwfruPzdx6Om2+45w9pii4rUsPhHBZ3NTJFG
lWCl8SJHkWtwCB2ClGZTETMz/FcxiM1r9PTpTiWD3oZ7ldOJ2IoJ0HB8MqEQUDBT
9K0hvpDxvMFOsY0m7DlAGb0PFv62esnLDYct0r8AgWlBioh/Sk9PlAFvew496sv5
-----END CERTIFICATE REQUEST-----
```

Notice that the message is no longer visible. With base64 every three bytes are represented by four characters so a byte may be encoded differently depending on its position within its grouping. The additional fields in the CSR changed the offset of the modulus, but the CA (Certificate Authority) will add more fields when signing it, so I'll worry about it once I know what the CA will add.

```
-----BEGIN CERTIFICATE-----
MIICyjCCAbICCQCSfbRHPi0dFzANBgkqhkiG9w0BAQUFADAnMQswCQYDVQQGEwJV
UzEYMBYGA1UEAwwPd3d3LmV4YW1wbGUuY29tMB4XDTE1MDQxNzAyMjcwM1oXDTE2
MDQxNjAyMjcwM1owJzELMAkGA1UEBhMCVVMxGDAWBgNVBAMMD3d3dy5leGFtcGxl
LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAPCcjl3YkQlTmZqB
BfZNNddNCEu7dlHIvTq7FxujW7NkQEFwcP/1oXp/3K8qbaIK2qYcv4rP6LrZWsHn
f22spf24YMp43/478sv7p4q/3Hr4p6Zf/w/RfeoIbRfwE+TfAQmqHQyr+KAJLh/V
cZfwb2ApIwTEWNy7qzwo0X8pR00ER9GYLsC97vELj9uMU28fJLkQU+JnUcSw+kBM
CWhfjm2G8OUocRCaYBZl6ot8KBiB0Aj9WKfsJkjB20uUiBg5znSwFmzH1WQviBP8
9NFUj65dEflbdSwczTC/TJZdVKLdlJZ+RsAUnCbVKakq/Q3ot3de9MvRmVtCVQ81
XvuFxJsCAwEAATANBgkqhkiG9w0BAQUFAAOCAQEAvCIILMlPnBL0phWKNNn3pWyv
5vZZ0g2T/MH4Tn7zd7ED39abjWe9kkThwPYG3GGhaxVoTEu0ftQfyAy32nZTwnU4
KdMBPkjvXkJQtPxsPx5O0mcizVPPX4lqrkozoUWrUTMvbkFytlnqgK1ekG0OprYi
GUjIVnxDrAqosrWYdoMU7OQM0r/OH4v7Bq6Nfk6zA5eCZqtC4ZQ/thz3POv6OUfS
hZs2fWSdfUf2BLQVuGro74Uin7T+ww56ivE02hOT2QLN6rjOcdLKKM7HHkke99kF
ZYh7S1YTUdsyZMV8DMSIlA/wcYDZr2CJNerg0NS6kiLCtO+cB5K3bvvzwav8Fw==
-----END CERTIFICATE-----
```

Still no message. Fortunately many CAs offer free re-issues. After regenerating the key and CSR a few times with different offsets, 59 turns out to be a winner.

```
-----BEGIN CERTIFICATE-----
MIICyjCCAbICCQCour4qG5JTdTANBgkqhkiG9w0BAQUFADAnMQswCQYDVQQGEwJV
UzEYMBYGA1UEAwwPd3d3LmV4YW1wbGUuY29tMB4XDTE1MDQxNzAzMTcxNFoXDTE2
MDQxNjAzMTcxNFowJzELMAkGA1UEBhMCVVMxGDAWBgNVBAMMD3d3dy5leGFtcGxl
LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALhGe8Ac/+f3Zig3
LfFlCvV419kOe50SL1JxgtF4x279KEkQLIJLE2OreO/u5DWEQCBGf9DUu514YsWG
//When/cryptography/is/outlawed/bayl/bhgynjf/jvyy/unir/cevinpl//
AeKvaZcayiiNMT7WEPan2IEqhB9BiHkB8sQ/hOWldr8Sn5N/MEgzFMs6vy9GqqAQ
```

```
B5X35iLaz2H5kyTuiVyhg7JoMYt/0tt0NvYp3IXPJTWoReUoaPZTJMuxAtUGiqOa
WXfVGlVjtaOauXtGaiI19EL2OQZYEZLwYigFRJK7VmDZ1EDtqn6eGjCPXGCtwm0t
wSrYYFUCAwEAATANBgkqhkiG9w0BAQUFAAOCAQEAlXu7E40uY7CL1lUR3Mf//KsV
sC1WkIYHWAncMrCO99wvDwsLVMcC0eFpC+q6bHpgjdyLR/Rux3YHqghNMdfjr4YJ
IrkKq91RjXVoyj13kuNkEFZTwP2YsTDcSZHv9bkgaXsdHNIs+o+O/Z9hUsLfCMBO
2O313ldDXn/mMLlIYO+pSdwShY+X61Ui79gCsoSkfz2SpB7VTQ6eRDzfXNDDCmVG
sMH0rKbl377c8JYxrALDQ06rt5DuRe06dV+OZcqu2SsCGyr6FcxA4Inckc/H5KPG
ga0Sbg5VTSJT2FdjKefcufSglf8/bFnS8hnPXVYSLboMGLB64EV4W707zI28uA==
-----END CERTIFICATE-----
```

So, is this safe to use? I couldn't come up with any attacks that this key generation method introduces, but I didn't try very hard. I certainly don't recommend using it in a commercial setting.

Posted by Ryan Castellucci on 2015-04-04. Tags: crypto ssl tls rsa python