Asymmetric Cryptography Part 2
RPISEC

Avi Weinstock (`aweinstock`),
Jacques Becker (`off_by_1_error`), Ethan Wright (`eth4`)

November 1, 2019

RSA Recap

```
from codecs import encode
from gmpy import invert, next_prime
import os
d = 0
while d == 0:
    p = next_prime(int(encode(os.urandom(1024/8), ’hex’), 16))
    q = next_prime(int(encode(os.urandom(1024/8), ’hex’), 16))
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537
    d = invert(e, phi)

message = int(encode(’hello’, ’hex’), 16)
ciphertext = pow(message, e, n)
assert pow(ciphertext, d, n) == message
```

Example RSA encryption/decryption

- $n = p * q$
- $\varphi(n) = (p-1) * (q-1)$
- $e * d \equiv 1 \pmod{\varphi(n)}$
- $\texttt{encrypt}(x) = x^e \% n$
- $\texttt{decrypt}(x) = x^d \% n$
- $\texttt{pow(x, k, n)} = x^k \% n$

- Public: $(n, e) = (667, 3)$
- Message `"hi"`, encoded as $7 * 26 + 8 = 190$
- `pow(190, 3, 667) == 239`

Example RSA encryption/decryption

- $n = p * q$
- $\varphi(n) = (p-1) * (q-1)$
- $e * d \equiv 1 \pmod{\varphi(n)}$
- $\texttt{encrypt}(x) = x^e \% n$
- $\texttt{decrypt}(x) = x^d \% n$
- $\texttt{pow(x, k, n)} = x^k \% n$

- Public: $(n, e) = (667, 3)$
- Message `"hi"`, encoded as $7 * 26 + 8 = 190$
- `pow(190, 3, 667) == 239`
- Private: $(p, q, d) = (23, 29, 411)$
- `(3 * 411) % (22 * 28) == 1`
- `pow(239, 411, 23*29) == 190`

Why to use e = 65537

- $2^{16+1} = 65537_{10} = 10000000000000001_2$
- It's prime, so $\texttt{invert}(65537, \varphi(n))$ is more likely to exist
- It mitigates multiple attacks:
  - Cube root
  - Hastad's broadcast
  - Coppersmith's short pad
- Since it only has 2 bits set, it's efficient to compute via repeated squaring:
  $m^{2^{16}+1} = m^{2^{16}} * m = (m^{2^8} * m^{2^8}) * m = ((m^{2^4} * m^{2^4}) * (m^{2^4} * m^{2^4})) * m = \ldots$

Extended Euclidean Algorithm

```
from gmpy import gcd
def eea(x, y):
    r, s, t = x, 1, 0
    R, S, T = y, 0, 1
    while R > 0:
        q = r/R
        new = r-q*R, s-q*S, t-q*T
        r, s, t = R, S, T
        R, S, T = new
    assert gcd(x, y) == r # gcd from euclidean algorithm
    assert r == x*s + y*t # s and t are the bezout coefficients
    xinvy = s + y*(s < 0) # modular inverse from bezout coefficients
    yinvx = t + x*(t < 0)
    if r == 1:
        assert (x * xinvy) % y == 1
        assert (y * yinvx) % x == 1
    return (r, s, t, xinvy, yinvx)
```

Chinese Remainder Theorem - Statement

- $\forall \vec{a}, \vec{n}((\forall i, j(i \neq j \rightarrow \gcd(n_i, n_j) = 1)) \rightarrow \exists x \forall i (x \equiv a_i \pmod{n_i}))$
- For a system of equations of the form $x \equiv a_i \pmod{n_i}$
- if each (n_i, n_j) pair are relatively prime
- there is a (unique) solution x for the system of equations

Chinese Remainder Theorem - Code

- $\forall \vec{a}, \vec{n}((\forall i, j(i \neq j \rightarrow \gcd(n_i, n_j) = 1)) \rightarrow \exists x \forall i(x \equiv a_i \pmod{n_i}))$
-
```python
from eea import eea
from gmpy import gcd
from itertools import combinations
def crt(eqns):
    assert len(eqns) >= 2
    assert [gcd(eqns[i][1], eqns[j][1]) == 1 for (i, j) in combinations(range(len(eqns)), 2)]
    a0, n0 = eqns[0]
    a1, n1 = eqns[1]
    _, m0, m1, _, _ = eea(n0, n1)
    assert m0*n0 + m1*n1 == 1
    x = a0*m1*n1 + a1*m0*n0
    if len(eqns) > 2:
        x = crt([(x, n0*n1)]+eqns[2:])
    for (a, n) in eqns:
        assert x % n == a % n
    return x
```

Chinese Remainder Theorem - Example

- $\forall \vec{a}, \vec{n}((\forall i, j(i \neq j \rightarrow \gcd(n_i, n_j) = 1)) \rightarrow \exists x \forall i(x \equiv a_i \pmod{n_i}))$
- $x \equiv 3 \pmod{5} \land x \equiv 4 \pmod{7}$
- $\mathrm{eea}(5, 7)$ gives us $(3, -2)$ as the Bezout coefficients
- This tells us that $3 * 5 + (-2) * 7 = 1$
- CRT gives us that $x = 3 * (-2) * 7 + 5 * 3 * 5$ solves the equation

Saltstack 2013 e=1 bug

Change key generation seq

0.15

thatch45 authored and **basepi** committed on May 8, 2013

1 parent 43d8c16 commit 5dd304276ba5745ec21fc1e6686a0b28da29e6fc

Browse files

Showing 1 changed file with **1 addition** and **1 deletion**.

Unified Split

2 salt/crypt.py

```diff
@@ -47,7 +47,7 @@ def gen_keys(keydir, keyname, keysize, user=None):
         priv = '{0}.pem'.format(base)
         pub = '{0}.pub'.format(base)
 
-        gen = RSA.gen_key(keysize, 1, callback=lambda x, y, z: None)
+        gen = RSA.gen_key(keysize, 65537, callback=lambda x, y, z: None)
         cumask = os.umask(191)
         gen.save_key(priv, None)
         os.umask(cumask)
```

Resources

- https://en.wikipedia.org/wiki/RSA_(cryptosystem)
- https://en.wikipedia.org/wiki/Chinese_remainder_theorem
- https://en.wikipedia.org/wiki/Modular_arithmetic
- https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf
- https://cryptopals.com/, Sets 5 and 6
- https://docs.saltstack.com/en/latest/topics/releases/0.15.1.html#rsa-key-generation-fault