Lecture 7

Cryptography 3: RSA

Agenda

1. Modular arithmetic
2. Diffie Hellman
3. RSA definition and hardness
4. RSA properties
5. RSA attacks
6. Tools

1. Modular arithmetic

Modular arithmetic

- Ring of integers modulo 256 - so called **uint8_t**.
- Congruence relation:
  - 0 ≡ 256 ≡ 512 ≡ 768 ≡ ...
  - 1 ≡ 257 ≡ 513 ≡ 769 ≡ ...
  - 0 ≡ 256*x
  - 255 ≡ -1
- "Equality"
- Formally: [0] = [256] = [512] = [768] = ...

Addition & Subtraction

- Addition:
  - $1 + 1 \equiv 2$
  - $128 + 128 \equiv 0$
  - $128 + 129 \equiv 0$
- Subtraction:
  - $3 - 2 \equiv 1$
  - $2 - 2 \equiv 0$
  - $2 - 3 \equiv -1 \equiv 255$

Multiplication

- Multiplication:
  - $2 * 2 \equiv 4$
  - $254 * 255 \equiv -2 * -1 \equiv -2 \equiv 254$
- So far so good

Division

- Inverse: $2 * {}^{1}/_{2} = 1$; $5 * {}^{1}/_{5} = 1$
- Multiplication in finite rings:
  - $\mathbf{x}^{\mathbf{m}} \equiv 1 \pmod{\mathbf{n}}$ *(for every x, n exists m such that...)*
  - $\mathbf{x}^{\mathbf{p}-1} \equiv 1 \pmod{\mathbf{p}}$ *(fermat's little theorem)*
- Extended GCD(**x**, **y**):
  - $i\mathbf{x} + j\mathbf{y} = \gcd(\mathbf{x}, \mathbf{y})$ $\gcd(\mathbf{x}, \mathbf{p}) \equiv 1 \pmod{\mathbf{p}}$
- Modular inverse:
  - $i\mathbf{x} + j\mathbf{n} = \gcd(\mathbf{x}, \mathbf{n}) \equiv 1 \pmod{\mathbf{n}}$
  - $\mathbf{i} * \mathbf{x} \equiv 1 \pmod{\mathbf{n}}$

2. Diffie Hellman

Key exchange

Alice, Bob: want to establish common secret

Eve: listening to Alice and Bob and want to intercept secret

Discrete logarithm

Easy: $a \equiv x^b \pmod{p}$

Hard: b = discrete_log(a, x, p)

Diffie Hellman

Public elements: x, p

Alice secret: a, Alice send to Bob: $x^a \bmod p$

Bob secret: b, Bob sent to Alice: $x^b \bmod p$

Alice calculates: $(x^b)^a \bmod p = x^{ab} \bmod p$

Bob calculates: $(x^a)^b \bmod p = x^{ab} \bmod p$

Man in the middle attack

Eve can modify messages from Bob and Alice

1. Eve exchange secret with Alice
2. Eve exchange secret with Bob
3. Eve reencrypt messages from one secret to another

Alice <-> Eve <-> Bob

Meet in the middle

Baby-step giant-step

Discrete logarithm: $a = x^b \bmod p$

$m = \text{sqrt}(p)$, $b = i*m + j$

sets of values: x^j, $a(x^{-m})^i = x^{i*m+j - m*i} = x^j$

Exercise

P = 1000000007

X = 2

A = 646497125

$A = X^B \bmod P$

B = ?

P-1 smooth

$P-1 = p_1 p_2 p_3 p_4 ... p_n$

Pohlig Hellman algorithm

Solution: P = 2*Q+1, Q prime

3. RSA definition

RSA

- RSA (Rivest, Shamir, Adleman)
- One of the first (practical) asymmetric encryption algorithms
- Two different keys: "public key" and "private key"

RSA: key generation

- Choose two distinct prime numbers: **p**, **q**
- **n** = **p** * **q**
- Compute φ(**n**)
- Choose **e** s.t. 1 < **e** < φ(**n**)
- Compute **d** ≡ $\mathbf{e}^{-1}$ *(mod φ(**n**))*
- Private key = (**p**, **q**, **e**, **d**, **n**). Public key = (**e**, **n**)
- Obviously, **d*****e** ≡ 1 *(mod φ(**n**))*

RSA: encryption & decryption

- Private key = (**p**, **q**, **e**, **d**, **n**). Message = **m**
- $\mathbf{c} \equiv \mathbf{m}^{\mathbf{e}} \pmod{\mathbf{n}}$
- $\mathbf{m} \equiv \mathbf{c}^{\mathbf{d}} \pmod{\mathbf{n}}$

Chinese Remainder Theorem

$x \equiv a_1 \pmod{n_1}$

$x \equiv a_2 \pmod{n_2}$

$x \equiv a_3 \pmod{n_3}$

...

$x \equiv \text{crt}(a_1, a_2, a_3, ...) \pmod{n_1 n_2 n_3 ...}$

RSA: encryption & decryption

- Private key = (**p**, **q**, **e**, **d**, **n**). Message = **m**
- $\mathbf{m} \equiv \mathbf{m}^{\mathbf{ed}} \pmod{\boldsymbol{n}}$ $\quad\quad$ $\mathbf{ed} \equiv 1 \pmod{\varphi(\mathbf{n})}$
- $\mathbf{m} \equiv \mathbf{m}^{\mathbf{k}\varphi(\mathbf{n})+1} \pmod{\boldsymbol{n}}$
- $\mathbf{m} \equiv \mathbf{m}*\mathbf{m}^{\mathbf{k}(\mathbf{p}-1)(\mathbf{q}-1)} \pmod{\boldsymbol{n}}$
- $\mathbf{m} \equiv \mathbf{m}*(\mathbf{m}^{(\mathbf{p}-1)})^{(\mathbf{q}-1)} \pmod{\boldsymbol{n}}$ $\quad$ $\mathbf{m} \equiv \mathbf{m}*(\mathbf{m}^{(\mathbf{q}-1)})^{(\mathbf{p}-1)} \pmod{\boldsymbol{n}}$
- Fermat's Little Theorem: $\quad$ $\mathbf{a}^{\mathbf{p}-1} \equiv 1 \pmod{\mathbf{p}}$
- $\mathbf{m} \equiv \mathbf{m}^{\mathbf{ed}} \pmod{\boldsymbol{p}}$ $\quad\quad$ $\mathbf{m} \equiv \mathbf{m}^{\mathbf{ed}} \pmod{\boldsymbol{q}}$
- $\mathbf{m} \equiv \mathbf{m}^{\mathbf{ed}} \pmod{\mathbf{n} = \mathbf{p}*\mathbf{q}}$

RSA CRT

$\mathbf{s}_1 \equiv \mathbf{m}^{\mathbf{dp}} \pmod{\mathbf{p}}$

$\mathbf{s}_2 \equiv \mathbf{m}^{\mathbf{dq}} \pmod{\mathbf{q}}$

$\mathbf{s} \equiv \mathrm{crt}((\mathbf{s}_1, \mathbf{p}), (\mathbf{s}_2, \mathbf{q}))$

RSA hardness

1. Factoring problem:

Given **n** = **p***q**, get **p**, **q**

No fast factoring algorithm for classical computer known

Fun fact: solved for quantum computer!

2. RSA problem:

Given **n**, **e**, **c** ≡ $\mathbf{m}^{\mathbf{e}}$ (mod **n**), get **m**

RSA & Python

```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)


def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m
```

http://store.tailcall.net/rsa.py

https://var.tailcall.net/rsa/privkey

https://var.tailcall.net/rsa/pubkey

4. RSA properties

Not really a blackbox: Jacobi symbol

- Jacobi symbol:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \cdots \left(\frac{a}{p_k}\right)^{\alpha_k},$$

- Preserved by RSA encryption!

Not really a blackbox: Homomorphism

- Homomorphism
- f(**a**) * f(**b**) = f(**a** * **b**)
- $\text{encrypt}(m_1) * \text{encrypt}(m_2) \equiv \text{encrypt}(m_1 * m_2) \ (mod\ \boldsymbol{n})$

https://var.tailcall.net/rsa/hmorph

Not really a blackbox: Solution

- **Padding** scheme
- **Random** padding scheme
- PKCS#1 and OAEP

PKCS#1 Padding

- For signing:

`msg = 0x00 | 0x01 | PS | 0x00 | D`

(where **PS** = 0xFF | 0xFF | ... | 0xFF)

- For encryption:

`msg = 0x00 | 0x02 | PS | 0x00 | D`

(where **PS** = random bytes)

Bleichenbacher's Attack

`msg = 0x00 | 0x02 |` **`PS`** `| 0x00 |` **`D`**

Attacker evasedropping connection, TLS key exchange

Client -> **c** = encrypt(**PreMasterSecret**) -> Server

decrypt(encrypt(**PreMasterSecret**)) = **PreMasterSecret**

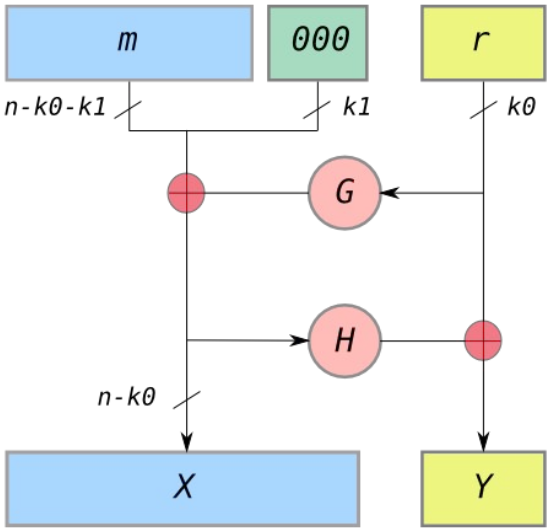
Attacker -> $\mathbf{c}*\mathbf{s}^{e}$ -> Server

decrypt($\mathbf{c}*\mathbf{s}^{e}$) -> decrypt($\mathbf{m}^{e}*\mathbf{s}^{e}$) -> **m*s**

Oracle: does **m*s** start with 0x00 | 0x02?

OAEP

- Better padding scheme
- Optimal Asymmetric Encryption Padding
- Form of a Feistel network, uses pair of random oracles G,H
- **Provably** secure
- "Plaintext aware"



Plaintext-Aware Encryption

- Cryptosystem is plaintexy aware if it's difficult to create a valid ciphertext without knowing corresponding plaintext
- Defined in OAEP
- Any cryptosystem that is Plaintext-Aware and Semantically Secure is secure against a chosen-ciphertext attack

Semantic Security

Cryptosystem is Semantically Secure <=> (Computationaly) impossible to determine any partial information about plaintext from ciphertext

5. RSA attacks

5.1. Elementary attacks

Factorizing n

- Trivial attack ("brute force")

- Solution: use bigger N

Cubic root

- **ct** ≡ **pt**$^{\mathbf{e}}$ (mod **n**), **e** = 3
- **ct** = 27
- **pt** = ?...

- Solution: padding

Common modulus, many users

- Generating big prime numbers (**p**, **q**) is rather slow.
- Server generates common modulus **n**.
- Alice gets $\mathbf{e}_1$, $\mathbf{d}_1$
- Bob gets $\mathbf{e}_2$, $\mathbf{d}_2$

- Problem: (**e**, **d**) is enough to factorize **n**...
- Solution: don't do this

Common modulus, gcd(e_1, e_2) = 1

- Generating big prime numbers (p, q) is rather slow.
- Server generates common modulus N.
- $\mathbf{c_1}$ encrypted with $\mathbf{e_1}$, $\mathbf{d_1}$
- $\mathbf{c_2}$ encrypted with $\mathbf{e_2}$, $\mathbf{d_2}$
- Problem: For example $\mathbf{e_1}$ = 3, $\mathbf{e_2}$ = 5. $c_1 = \mathbf{m}^3$, $c_2 = \mathbf{m}^5$. We can compute $\mathbf{m}^8$ as $\mathbf{c_1}*\mathbf{c_2}=\mathbf{m}^3*\mathbf{m}^5$. After this, $\mathbf{m}^{11}=\mathbf{m}^8*\mathbf{m}^3$. After this, $\mathbf{m}^6 = \mathbf{m}^{11}*\mathbf{m}^{-5}$. After this, $\mathbf{m} = \mathbf{m}^1 = \mathbf{m}^6*\mathbf{m}^{-5}$
- Solution: don't do this, really

Low private exponent

- A.K.A. **Wiener's Attack**
- Usable when e is big <=> d is small
- $d < 1/3 * N^{1/4}$
- Uses continued fractions to recover private key from public key

5.2. Coppersmith's method

Coppersmith's Method

- Method to find small integer zeroes of polynomials modulo a given integer.

- $F(x) = \mathbf{a_0} + \mathbf{x a_1} + \mathbf{x^2 a_2} + \mathbf{x^3 a_3} + \ldots$
- find **y** s.t.: $F(\mathbf{x_0}) \equiv 0 \pmod{\mathbf{n}}$

- Uses **LLL algorithm**

Interlude: LLL

- Lenstra–Lenstra–Lovász (LLL) lattice basis reduction algorithm
- "Find short vectors in lattice"

$$\begin{matrix} 1 & 0 & 0 \\ 12 & 1 & 0 \\ 33 & 82 & 1 \end{matrix} \quad \Rightarrow \quad \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

Interlude: LLL

Application:

- r=1.618034 is a solution to some quadratic equation

$$\begin{bmatrix} 1 & 0 & 0 & 100000r^2 \\ 0 & 1 & 0 & 100000r \\ 0 & 0 & 1 & 100000 \end{bmatrix}$$

- Small solution: [**a**, **b**, **c**, 100000(**a**r^2 + **b**r + **c**)]
  [1, -1, -1, 0.0000...]

Coppersmith Attack

- Attack on RSA when public exponent **e** is big
- Coppersmith method can effectively find all roots of f(x) modulo N, smaller than $X = \mathbf{N}^{1/d}$

- Other uses: high bits of key known, recover the rest
- Other uses: low bits of key known, recover the rest
- Other uses: (...)

Hastad's Broadcast Attack

- The same message m sent to e=3 people
- $(3, \mathbf{n}_1)$, $(3, \mathbf{n}_2)$, $(3, \mathbf{n}_3)$ - three public keys with e=3
- $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$ - intercepted messages
- $\mathbf{m}^3 \pmod{\mathbf{n}_1}$, $\mathbf{m}^3 \pmod{\mathbf{n}_2}$, $\mathbf{m}^3 \pmod{\mathbf{n}_3}$
- $\mathbf{m}^3 \equiv \text{crt}((\mathbf{m}^3, \mathbf{n}_1), (\mathbf{m}^3, \mathbf{n}_2), (\mathbf{m}^3, \mathbf{n}_3)) \pmod{\mathbf{n}_1\mathbf{n}_2\mathbf{n}_3}$
- $\mathbf{m}^3 < \mathbf{n}_1\mathbf{n}_2\mathbf{n}_3$
- $\mathbf{m}^3 = \text{crt}((\mathbf{m}^3, \mathbf{n}_1), (\mathbf{m}^3, \mathbf{n}_2), (\mathbf{m}^3, \mathbf{n}_3))$

Franklin-Reiter Attack

- Related-message attack on RSA
- $f(x) = a\mathbf{x} + \mathbf{b} \pmod{\mathbf{n}}$
- $\mathbf{m_2} \equiv f(\mathbf{m_1}) \pmod{\mathbf{n}}$
- $\mathbf{m_1}$ is root of $\mathbf{g_1}(\mathbf{x}) = \mathbf{f}(\mathbf{x})^e - c_1$
- $\mathbf{m_2}$ is root of $\mathbf{g_2}(\mathbf{x}) = \mathbf{x}^e - c_2$
- $\mathbf{x} - \mathbf{m_2}$ divides both.
- Idea: use $\gcd(\mathbf{g_1}, \mathbf{g_2})$ to find $\mathbf{x} - \mathbf{m_2}$
- $\mathbf{m_2}$ recovered.

5.3. Implementation attacks

Timing attacks

```
def square_and_multiply(c, d):
    x = c
    for j in range(n):
        x = x*x % N
        if d[j] == 1:
            x = x * C % N
    return x
```

Fault attacks on RSA-CRT

$s_1 \equiv m^{dp} \pmod{p}$; $s_2 \equiv m^{dq} \pmod{q}$; $s \equiv \mathrm{crt}((s_1, p), (s_2, q))$

$s_1 \equiv m^{dp} \pmod{p}$; $s_2 \equiv m^{dq} \pmod{q}$; $s' \equiv \mathrm{crt}((s_1, p), (s_2, q))$

$s'^{e} \equiv m \pmod{p}$; $s'^{e} \not\equiv m \pmod{q}$

$s'^{e} - m \equiv 0 \pmod{p}$; $s'^{e} - m \not\equiv 0 \pmod{q}$

$p \mid s'^{e}$; $q \nmid s'^{e}$; $p = \gcd(s'^{e} - m, n)$

6. Tools

SageMath

http://www.sagemath.org/

- discrete_log: baby-step giant-step, Pohlig hellman
- Coppersmith: small_roots
- LLL
- factor

Factorization

- http://factordb.com/
- yafu: https://github.com/DarkenCode/yafu
- https://github.com/Ganapati/RsaCtfTool
- https://pypi.python.org/pypi/primefac

LLL attacks

https://github.com/mimoo/RSA-and-LLL-attacks (Sage Scripts)

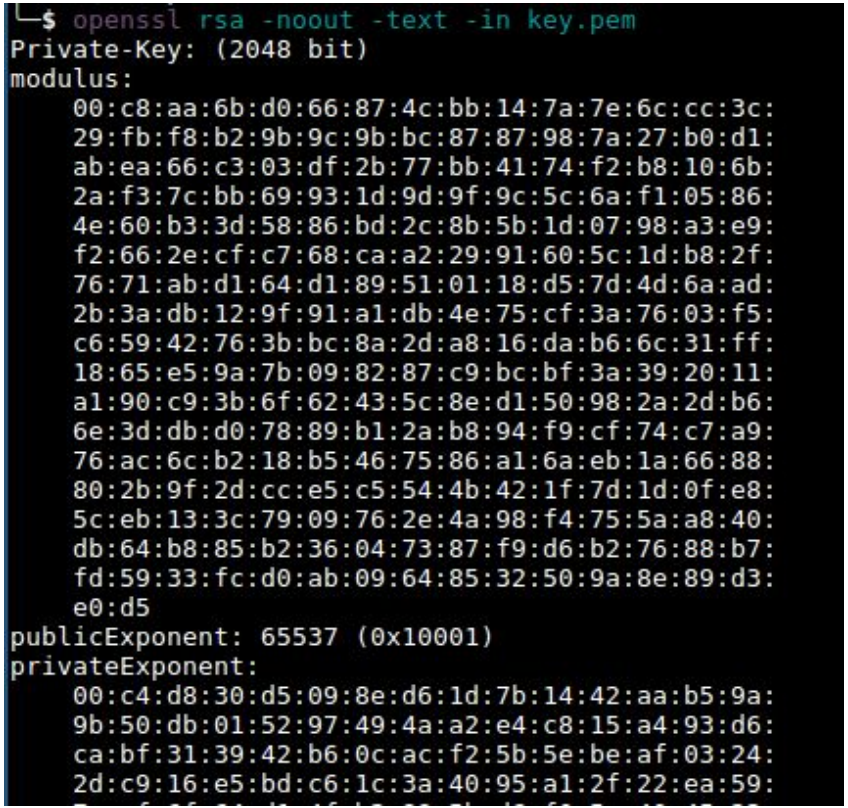
- Coppersmith: Stereotyped messages
- Coppersmith: Factoring with high bits known
- Boneh Durfee

OpenSSL

- Swiss army knife for RSA

Read private key:

```
openssl rsa -noout -text -in key.pem
```



OpenSSL

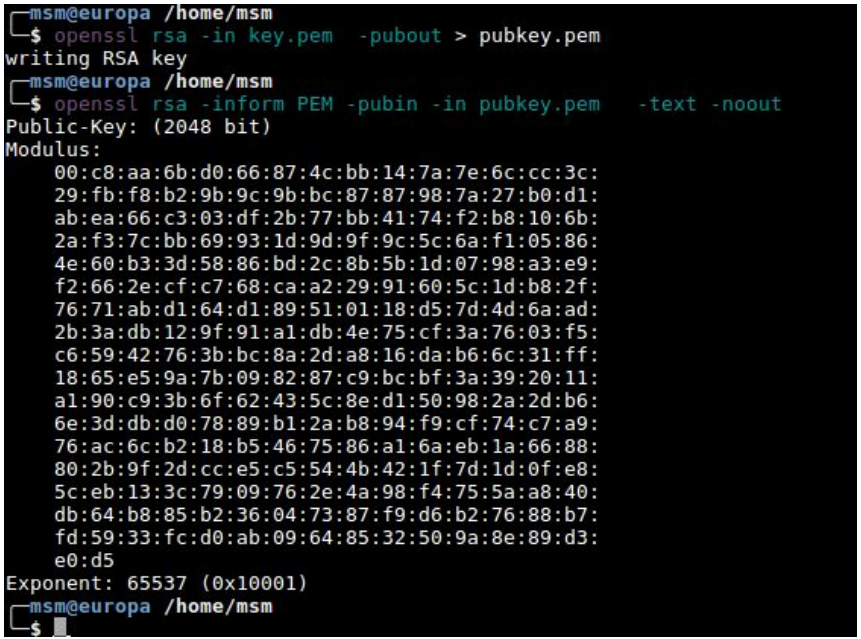
- Swiss army knife for RSA

Convert privkey to pubkey:

```
openssl rsa -in key.pem  -pubout > pubkey.pem
```

Read public key:

```
openssl rsa -inform PEM -pubin -in pubkey.pem   -text -noout
```



Bibliography