Applied Security

Attacks on RSA implementations

Notes

General Overview

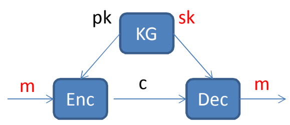
- RSA (implementations)
- Focus on fault analysis (and countermeasures)
- Focus on timing analysis
- Focus on differential attacks
- Focus on countermeasures

Beware: in order to follow the lectures you NEED to be familiar with various cryptographic algorithms and implementation techniques!

Notes

R(ivest)S(hamir)A(dleman)

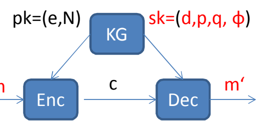
- Public key cryptosystem
  - i.e. users have key pair (public=e, private/secret=d), keys are mathematically linked via trapdoor one-way functions
    - Easy to compute but hard to invert unless you know some secret trapdoor information (aka the secret key)
- Encryption/Decryption/Key generation:



Notes

RSA-Key Generation

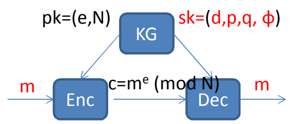
- Key generation:
  - Generate two large primes p and q, then compute N=p*q, and ϕ(N)=(p-1)*(q-1)
  - Select a random integer 1<e<ϕ(N), such that gcd(e, ϕ(N))=1, and derive d=e^{-1} (mod ϕ(N))



Notes

RSA: Encryption/Decryption

- Encryption/Decryption:
  - Encryption: obtain receiver's public key (N,e), represent message as a number 0<m<N, and compute $c=m^e \pmod N$
  - Decryption: recover m by computing $m=c^d \pmod N$



Notes

RSA-Example

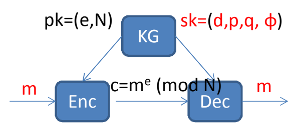
Key generation:

- Choose primes $p = 7$ and $q = 11$.
- Compute $N = 77$ and $\phi(N) = (p-1)(q-1) = 6 \times 10 = 60$.
- Choose $e = 37$, which is valid since $\gcd(37, 60) = 1$.
- Using the XGCD, compute $d = 13$ since $37 \times 13 \equiv 481 \equiv 1 \pmod{60}$.
- Public key $= (77, 37)$ and private key $= (13, 7, 11)$.

Encryption: suppose $m = 2$ then $c = m^e \mod N = 2^{37} \mod 77 = 51$.

Decryption: to recover m compute

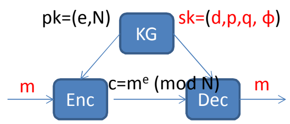
$$m = c^d \mod N = 51^{13} \mod N = 2 .$$



Notes

Security of Vanilla RSA

- Key generation
  - e and d are mathematically linked: $d=e^{-1} \pmod{\phi(N)}$
  - If we can factor N, we can compute $\phi(N)$ and hence derive d and so decrypt, so RSA cannot be more secure than factoring
    - RSAP $<_p$ FACTORING (in English: the RSAP, i.e. computing m given (c,e,N), is no harder than factoring N)
- But this does not imply that factoring is the only way to break RSA



Notes

Recall RSA is malleable

- Example: instruct payment of 10 pounds
  - m=10, $c = 10^e \pmod N$
  - Intercept ciphertext and pass on $2^e c$
  - Decryption: $(2^e c)^d \pmod N = 20$
- So by manipulating the ciphertext we get new valid encryptions to unknown related messages
  - We can also do this: $c_3 = c_2 * c_1$ because RSA is homomorphic
    - So we can create a valid encryption of the product of two messages without actually knowing the messages

Notes

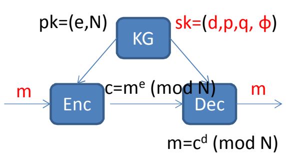
RSA in the Wild

- Vanilla RSA is not even Ind-CPA because it is deterministic
- Hence in real world protocols (Vanilla) RSA is ‚embedded' in ‚stuff' (we'll be more precise later)
- For now though we look at what ‚damage' we can do to (Vanilla) RSA when considering more resourceful adversaries

Notes

RSA implementations

- Key ingredients to make RSA fast:
  - Small public key
  - Fast exponentiation algorithm
  - Fast modular multiplication algorithm
- Remember that real life RSA key sizes mean working with very large numbers!



Notes

RSA implementations, cont.

- Focus is on decryption
  - $m=c^d$ (mod N)
- ‚Square and multiply‘ algorithm is a popular choice
  - Recall that there are other windowing methods out there too
- It processes the secret key bit by bit

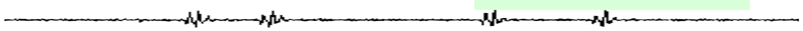
$d=\{d_w,d_{w-1},d_{w-2},\ldots,d_1,d_0\}_2$

```
m = 1;
For i = w-1 to 0
  m = m • m mod n
  if (d_i) == 1
    then m = m • c mod N
  (endif)
(endfor)
```

Algorithm (Binary-L2R-1Exp)

Input: A group element $x \in \mathbb{G}$ of order n, an integer $0 \le y < n$ represented in base-2
Output: The group element $r = [y]x \in \mathbb{G}$

```
t ← 0_G
for i = |y| − 1 downto 0 step −1 do
  t ← [2]t
  if y_i = 1 then
    t ← t + x
  end
end
return t
```



Notes

RSA implementations, cont.

- Focus is on decryption
  - $m=c^d$ (mod N)
- Montgomery multiplication

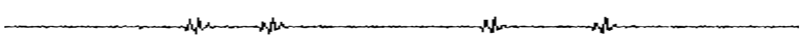
Algorithm ($\mathbb{Z}_N$-MontExp)

Input: A base-b, unsigned integer $0 \le x < N$, and a base-2, unsigned integer $0 \le y < N$
Output: A base-b, unsigned integer $r = x^y$ (mod N)

```
t̂ ← Z-MontMul(1, ρ^2), x̂ ← Z-MontMul(x, ρ^2)
for i = |y| − 1 downto 0 step −1 do
  t̂ ← Z-MontMul(t̂, t̂)
  if y_i = 1 then
    t̂ ← Z-MontMul(t̂, x̂)
  end
end
return Z-MontMul(t̂, 1)
```

$d=\{d_w,d_{w-1},d_{w-2},\ldots,d_1,d_0\}_2$

```
m = 1;
For i = w-1 to 0
  m = m • m mod n
  if (d_i) == 1
    then m = m • c mod N
  (endif)
(endfor)
```



Notes

RSA implementations, cont.

- Focus is on decryption
  - $m=c^d \pmod N$
- Montgomery multiplication

$d=\{d_w,d_{w-1},d_{w-2},\ldots,d_1,d_0\}_2$

```
m = 1;
For i = w-1 to 0
  m = m • m mod n
  if (d_i) == 1
    then m = m • c mod N
  (endif)
(endfor)
```

Algorithm ($\mathbb{Z}_N$-MontExp)

Input: A base-b, unsigned integer $0 \le x < N$, and a base-2, unsigned integer $0 \le y < N$

Output: A base-b, unsigned integer $r = x^y \pmod N$

```
t̂ ← Z-MontMul(1, ρ²), x̂ ← Z-MontMul(x, ρ²)
for i = |y| − 1 downto 0 step −1 do
  t̂ ← Z-MontMul(t̂, t̂)
  if y_i = 1 then
    t̂ ← Z-MontMul(t̂, x̂)
  end
end
return Z-MontMul(t̂, 1)
```

Algorithm ($\mathbb{Z}_N$-MontMul)

Input: Two base-b, unsigned integers, $0 \le x, y < N$

Output: A base-b, unsigned integer $r = x \cdot y \cdot \rho^{-1} \pmod N$

```
r ← 0
for i = 0 upto l_N − 1 step +1 do
  u ← (r_0 + y_i · x_0) · ω (mod b)
  r ← (r + y_i · x + u · N)/b
end
if r ≥ N then
  r ← r − N
end
return r
```

Notes

RSA implementations, cont.

Let $m_1, \ldots, m_r$ be pairwise relatively prime and let $a_1, \ldots, a_r$ be integers.

We want to find x modulo $M = m_1 m_2 \cdots m_r$ such that

$$x \equiv a_i \pmod{m_i} \quad \text{for all } i.$$

The CRT guarantees a unique solution given by

$$x = \sum_{i=1}^{r} a_i \cdot M_i \cdot y_i \pmod M$$

with

$$M_i = M/m_i \quad \text{and} \quad y_i = M_i^{-1} \pmod{m_i}.$$

Note that $M_i \equiv 0 \pmod{m_j}$ for $j \neq i$ and that $M_i \cdot y_i \equiv 1 \pmod{m_i}$.

Suppose we want to compute

$$y = x^d \pmod N$$

where $N = p \cdot q$.

We know by Lagranges Theorem

$$x^{p-1} = 1 \pmod p$$

So we first compute $y \pmod p$ and $y \pmod q$ via

$$y_p = y \pmod p = x^d \pmod p = x^{d \pmod{p-1}} \pmod p,$$
$$y_q = y \pmod q = x^d \pmod q = x^{d \pmod{q-1}} \pmod q.$$

We then solve for y by applying the CRT to the equations

$$y \equiv y_p \pmod p$$
$$y \equiv y_q \pmod q.$$

- Last trick we mention: CRT (Chinese Remainder Theorem)
  - Means we can perform computations modulo p and modulo q rather than modulo N

Notes

Security of RSA implementations

Algorithm (Binary-L2R-1Exp)

```
Input: A group element x ∈ G of order n, an
       integer 0 ≤ y < n represented in base-2
Output: The group element r = [y]x ∈ G

t ← 0_G
for i = |y| − 1 downto 0 step −1 do
    t ← [2]t
    if y_i = 1 then
        t ← t + x
    end
end
return t
```

Algorithm ($\mathbb{Z}_N$-MontMul)

```
Input: Two base-b, unsigned integers,
       0 ≤ x, y < N
Output: A base-b, unsigned integer r = x · y · ρ^−1
        (mod N)

r ← 0
for i = 0 upto l_N − 1 step +1 do
    u ← (r_0 + y_i · x_0) · ω (mod b)
    r ← (r + y_i · x + u · N)/b
end
if r ≥ N then
    r ← r − N
end
return r
```

- Two conditional clauses, that make flow of data (and operations dependent on secret)
  - This means that observable behaviour in the form of timing characteristics, power consumption, EM emanation, etc. will depend on secret

Notes

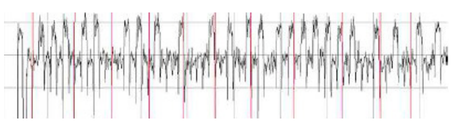
Security of RSA implementations: Fault Analysis

- We can also exploit the dependency on the secret via active attacks
  - Assume we can manipulate the smart card such that we can produce a ‚fault' whilst it performs the exponentiation

Algorithm (Binary-L2R-1Exp)

```
Input: A group element x ∈ G of order n, an
       integer 0 ≤ y < n represented in base-2
Output: The group element r = [y]x ∈ G

t ← 0_G
for i = |y| − 1 downto 0 step −1 do
    t ← [2]t
    if y_i = 1 then
        t ← t + x
    end
end
return t
```



We can use power traces to detect the square and multiply patterns.

Notes

RSA fault analysis, cont.

- We configure our setup such that in step i of our attack, the i-th bit is set to zero: $y_i'=0$, $y_j'=y_j$
  - All other bits of the secret remain unchanged

```
Algorithm (Binary-L2R-1Exp)
  Input: A group element x ∈ G of order n, an
         integer 0 ≤ y < n represented in base-2
  Output: The group element r = [y] x ∈ G

t ← 0_G
for i = |y| − 1 downto 0 step −1 do
  | t ← [2] t
  | if y_i = 1 then
  |   | t ← t + x
  | end
end
return t
```

- We decrypt a random text c as reference
- For each index i we force the key bit to 0
  - Iff $y_i=0$ then no change in the device/key occurs and the decryption returns c again
  - Iff $y_i=1$ then the key has been changed and c' is returned
- We can recover the entire key with n queries!

Summary

- RSA implementations are complex and many options exist
  - Square and multiply exponentiation
  - Montgomery multiplication
  - Chinese remainder theorem
- We are interested in the security implications that these choices bring up and how the mathematical properties of RSA help (or hinder) us exploiting them.

Notes