RSA Security

Cryptanalysis

2016.10

RSA Security

1. Reductions of One-way Functions
2. RSA Security
   1. RSA Encryption and the RSA Problem
   2. Knowledge of the Private Exponent and Factoring
   3. Knowledge of $\phi(N)$ and Factoring
   4. Use of a Shared Modulus
   5. Use of a Small Public Exponent
3. Physical Attacks

One-way Functions

- One-way Functions
  - A function $f: X \to Y$ with $y = f(x)$ is a **one-way function** if
    - For all $x \in X$ it is very easy or efficient to compute $f(x)$
    - For almost all $y \in Y$, finding an $x \in X$ with $f(x) = y$ is computationally infeasible
  - A **trapdoor one-way function** is a one-way function $f: X \to Y$, but given some extra information, called the **trapdoor information**, it is easy to invert f
    - i.e. given $y \in Y$, it is easy to find $x \in X$ such that $f(x) = y$
  - Remark There is no proof that such functions actually exist

One-way Functions

- Candidate one-way functions
  - Multiplication
    - Given primes p and q, computing $N = p \cdot q$ is easy
    - The inverse problem is called **factoring**
      - Given N, find p and q
  - Modular exponentiation
    - Given N and $a \in \mathbf{Z}_N$, computing $b \equiv a^m \pmod N$ is efficient using *square and multiply*
    - The inverse is called the **discrete logarithm problem**
      - Given N, a, $b \in \mathbf{Z}_N$, find m such that $b \equiv a^m \pmod N$

Reductions

- Reductions
  - We will reduce one hard problem to another, which will allows us to compare the relative difficulty, i.e. we can say “Problem *A* is no harder than Problem *B*”
  - Let *A* and *B* be two computational problems, then *A* is said to *polytime reduce* to *B*, written $A \leq_P B$ if
    - There is an algorithm which solves *A* using an algorithm which solves *B*
    - This algorithm runs in polynomial time if the algorithm for *B* does
  - Assume we have an oracle (or efficient algorithm) to solve problem *B*, then we use this oracle to give an efficient algorithm for problem *A*

Hard Problems

- List of Hard Problems – Part I
  - Given N but not p, q such that $N = p \bullet q$
    - FACTORING : Find p and q
    - RSAP : Given $c \in \mathbf{Z}_N$ and e with $\gcd(e, \varphi(N)) = 1$, find m such that $m^e \equiv c \pmod N$
    - QUADRES : Given a, determine whether $a \equiv x^2 \pmod N$
    - SQROOT : Given a such that $a \equiv x^2 \pmod N$, find x
  - Later we will prove that
    - QUADRES $\leq_P$ SQROOT $\equiv_P$ FACTORING
    - RSAP $\leq_P$ FACTORING

Discrete Logarithm Problem

- Discrete Logarithm Problem
  - Let $(G, \times)$ be an abelian group
  - Given $g, h \in G$, find x (if it exists) such that $g^x = h$
- The difficulty of this problem depends on the group *G*:
  - Very easy: polynomial time algorithm, e.g. $(\mathbf{Z}_N, +)$
  - Rather hard: sub-exponential time algorithm, e.g. $(\mathbf{F}_p, \times)$
  - Very hard: exponential time algorithm, e.g. Elliptic Curve groups

Hard Problems

- List of Hard Problems – Part II
  - Given an abelian group $(G, \times)$ and $g \in G$
    - DLP : Given $h \in G$ such that $h = g^x$, find x
    - DHP : Given $a = g^x$ and $b = g^y$, find $c = g^{xy}$
    - DDH : Given $a = g^x$, $b = g^y$, $c = g^z$, determine if $z = xy$
  - Later we will prove that:
    - If we can solve DLP then we can solve DHP
    - If we can solve DHP then we can solve DDH
    - That is, DDH $\leq_P$ DHP $\leq_P$ DLP

DHP $\le_P$ DLP

- Reductions – DHP $\le_P$ DLP
  - Here we show how to reduce DHP to DLP
    - i.e. we give an efficient algorithm for solving the DHP given an oracle for the DLP
    - Given g^x and g^y, we wish to find g^{xy}
    - First compute y = DLP(g^y) using the oracle
    - Then compute $(g^x)^y = g^{xy}$
    - So DHP is no harder than DLP, i.e. DHP $\le_P$ DLP
  - <u>Remark</u> In some groups, we can show that DHP is equivalent to DLP

DDH $\le_P$ DHP

- Reductions – DDH $\le_P$ DHP
  - Here we show how to reduce DDH to DHP
    - i.e. we give an efficient algorithm for solving the DDH given an oracle for the DHP
    - Given elements g^x, g^y and g^z, determine if $z = x\,y$
    - Using the oracle to solve DHP, compute g^{xy} = DHP(g^x, g^y)
    - Then check whether $g^{xy} = g^z$
    - So DDH is no harder than DHP, i.e. DDH $\le_P$ DHP
  - <u>Remark</u> In some groups, we can show that DDH is probably easier than DHP

SQROOT $\le_P$ FACTORING

- Reductions – SQROOT $\le_P$ FACTORING
  - Here we show how to reduce SQROOT to FACTORING
    - i.e. we give an efficient algorithm for solving SQROOT given an oracle for FACTORING
    - Given $z = x^2 \pmod N$ we wish to compute x
      - Using the oracle for FACTORING, find the prime factors p_i of N
      - Compute $\sqrt{z} \pmod{p_i}$ (can be done in polynomial time)
      - Recover $\sqrt{z} \pmod N$ using CRT
  - So computing square roots modulo N is no harder than factoring, i.e., SQROOT $\le_P$ FACTORING

FACTORING $\le_P$ SQROOT

- Reductions – FACTORING $\le_P$ SQROOT
  - Here we show how to reduce FACTORING to SQROOT
    - i.e. we give an efficient algorithm for FACTORING given an oracle for SQROOT
    - Given $N = p\,q$, we wish to compute p and q
      - Compute $z = x^2$ for a random $x \in \mathbf{Z}_N^*$
      - Compute $y = \sqrt{z} \pmod N$ using the oracle for SQROOT
        - There are four possible square roots because of two factors
        - With 50% probability we have $y \ne \pm x \pmod N$
      - Factor N by computing $\gcd(x - y, N)$
  - So factoring is no harder than computing square roots modulo N, i.e., FACTORING $\le_P$ SQROOT

FACTORING $\equiv_P$ SQROOT

- Reductions – FACTORING $\equiv_P$ SQROOT
  - Summarizing the result of the previous two slides

    FACTORING $\leq_P$ SQROOT

    SQROOT $\leq_P$ FACTORING
  - So FACTORING and SQROOT are computationally equivalent

    SQROOT $\equiv_P$ FACTORING

RSAP $\leq_P$ FACTORING

- Reductions – RSAP $\leq_P$ FACTORING
  - Here we show how to reduce RSAP to FACTORING
    - i.e. we give an efficient algorithm for solving RSAP given an oracle for FACTORING
    - Given $c = m^e \pmod N$ and the integer e, find m
      - Find the factorization of $N = p\,q$ using the oracle
      - Compute $\phi(N) = (p-1)(q-1)$
      - Use XGCD to compute $d = e^{-1} \pmod{\phi(N)}$
      - Finally, recover $m = c^d \pmod N$
  - So the RSA problem is no harder than factoring
    - i.e. RSAP $\leq_P$ FACTORING
  - There is some slight evidence that it might be easier, but nobody has a proof yet

RSA Security

1. Reductions of One-way Functions
2. **RSA Security**
   1. **RSA Encryption and the RSA Problem**
   2. Knowledge of the Private Exponent and Factoring
   3. Knowledge of $\phi(N)$ and Factoring
   4. Use of a Shared Modulus
   5. Use of a Small Public Exponent
3. Physical Attacks

Security

- Security of RSA
  - Relies on the difficulty of finding d from N and e
  - We have already proved RSAP $\leq_P$ FACTORING
    - Since if we can find p and q, then we can compute d
    - If factoring is easy, then we can break RSA
  - Currently the largest factored RSA number has 768 bits
    - http://eprint.iacr.org/2010/006 (Dec 12, 2009)
  - For medium security, better choose $\geq$ 1024 bits
  - NESSIE recommended N with $\geq$ 1536 bits and $e \geq 65537$
    - **N**ew **E**uropean **S**chemes for **S**ignatures, **I**ntegrity, and **E**ncryption
    - (a project from 2000 to 2002) http://www.cryptonessie.org

RSA-Problem

- It can be shown that RSA, as we have described it, is not secure against **chosen ciphertext attacks**
- But, RSA is secure against **chosen plaintext attack** assuming the RSA-Problem is hard
  - We give an algorithm which solves the RSA-Problem
  - Using an algorithm to break RSA as an oracle
- Recall the RSA-Problem
  - Given $N = p\,q$, $y \in \mathbf{Z}_N$, and e with $\gcd(e, \phi(N)) = 1$
  - Find x such that $x^e \equiv y \pmod{N}$

Chosen Plaintext Attack

- Chosen Plaintext Attack
  - Assume we are given the integer $y \in \mathbf{Z}_N$ and want to find an integer x such that $x^e \equiv y \pmod{N}$
  - Furthermore, suppose we have an oracle which breaks the RSA cryptosystem, i.e. which can decrypt ciphertexts
  - Let $c = y$ and using the oracle, "decrypt" the ciphertext to obtain the plaintext m
  - Then we can take $x = m$ since $m^e \equiv c \equiv y \pmod{N}$
  - So if we can break RSA then we can solve the RSA-Problem
  - Conclusion: RSA is secure under chosen plaintext attack

RSA Security

1. Reductions of One-way Functions
2. RSA Security
   1. RSA Encryption and the RSA Problem
   2. **Knowledge of the Private Exponent and Factoring**
   3. Knowledge of $\phi(N)$ and Factoring
   4. Use of a Shared Modulus
   5. Use of a Small Public Exponent
3. Physical Attacks

Private Exponent & Factoring

- **Lemma** Knowing d is equivalent to factorization of $N = p\,q$
- Proof ($\Leftarrow$) $d \equiv e^{-1} \bmod (p-1)(q-1)$

($\Rightarrow$) $ed - 1 = k(p-1)(q-1)$ for some $k \in \mathbf{Z}$, so $x^{ed-1} \equiv 1 \pmod{N}$ for $x \neq 0$.

We want to put $y_1 = \sqrt{x^{ed-1}} \equiv x^{(ed-1)/2}$ and then use $y_1^2 - 1 \equiv 0 \pmod{N}$ to factor N by $\gcd(y_1 - 1, N)$, which works only if $y_1 \neq 1 \pmod{N}$.

Suppose $y_1 \equiv 1 \pmod{N}$, then we take a square root of y_1:

$y_2 = \sqrt{y_1} \equiv x^{(ed-1)/4}$ and we know $y_2^2 \equiv y_1 \equiv 1 \pmod{N}$.

Hence we compute $\gcd(y_2 - 1, N)$ and see if this factors N.

Repeat until we factor N, or $(ed-1)/2^s$ is no longer divisible by 2.

We will factor N with probabilit y 1/2.

Private Exponent & Factoring

- **Example** Factor $N = 1441499$ with $e = 17$, $d = 507905$
- Solution

Put $t_1 = (ed-1)/2 = 4317192$ and $y_0 = 2^{ed-1} \equiv 1 \pmod{N}$.

Compute $y_1 = \sqrt{y_0} \equiv 2^{(ed-1)/2} \equiv 2^{t_1} \equiv 1 \pmod{N}$.

So we need $t_2 = t_1/2 = (ed-1)/4 = 2158596$

and compute $y_2 = \sqrt{y_1} = y_0^{1/4} \equiv 2^{(ed-1)/4} \equiv 2^{t_2} \equiv 1 \pmod{N}$.

Repeat again $t_3 = t_2/2 = (ed-1)/8 = 1079298$

and compute $y_3 = \sqrt{y_2} = y_0^{1/8} \equiv 2^{(ed-1)/8} \equiv 2^{t_3} \equiv 119533 \pmod{N}$.

So $y_3^2 - 1 = (y_3+1)(y_3-1) \equiv 0 \pmod{N}$ and $\gcd(y_3-1, N) = 1423$.

Therefore $1441499 = 1423 \times 1013$.

RSA Security

1. Reductions of One-way Functions
2. RSA Security
   1. RSA Encryption and the RSA Problem
   2. Knowledge of the Private Exponent and Factoring
   3. **Knowledge of $\phi(N)$ and Factoring**
   4. Use of a Shared Modulus
   5. Use of a Small Public Exponent
3. Physical Attacks

$\phi(N)$ and Factoring

- **Lemma** Knowing $\phi(N)$ is equivalent to factorization of $N = p\,q$

- Proof ($\Leftarrow$) $\phi(N) = (p-1)(q-1)$

($\Rightarrow$) $\phi(N) = (p-1)(q-1) = N - (p+q) + 1$

$\Rightarrow a = p + q = N - \phi(N) + 1$

$(p+q)^2 = p^2 - 2pq + q^2 + 4pq = (p-q)^2 + 4N$

$\Rightarrow b = p - q = \sqrt{(p+q)^2 - 4N}$

Therefore $p = (a+b)/2$ and $q = (a-b)/2$

$\phi(N)$ and Factoring

- **Example** Factor $N = 18923$ with $\phi(N) = 18648$
- Solution

$p + q = N + 1 - \phi(N) = 276$

$(p-q)^2 = (p+q)^2 - 4\,p\,q = 276^2 - 4\,N = 484$

$p - q = 22$

$p = (276 + 22)/2 = 149$

$q = (276 - 22)/2 = 127$

We have $N = 149 \times 127$

RSA Security

1. Reductions of One-way Functions
2. RSA Security
   1. RSA Encryption and the RSA Problem
   2. Knowledge of the Private Exponent and Factoring
   3. Knowledge of $\phi(N)$ and Factoring
   4. **Use of a Shared Modulus**
   5. Use of a Small Public Exponent
3. Physical Attacks

Shared Modulus

- Attack from a malicious insider
  - Assume for efficiency that each user has
    - The same modulus N
    - Different public/private exponents (e_i, d_i)
  - Suppose M is the user number one, and M wants to find d_2 of user number two
    - M computes p and q since d_1 is known
    - M computes $\phi(N) = (p-1)(q-1)$
    - M computes $d_2 = e_2^{-1} \pmod{\phi(N)}$
  - So each user can find every other user's key

Shared Modulus

- Attack from an external eavesdropper
  - Now suppose the attacker is not one of the people who share a modulus
  - Suppose Alice sends the message m to two people with public keys (N, e_1) and (N, e_2), i.e. $N_1 = N_2 = N$
  - Eve can see the messages $c_1 = m^{e_1}$ and $c_2 = m^{e_2}$
  - Eve can now compute

$$t_1 = e_1^{-1} \pmod{e_2} \text{ and } t_2 = (t_1 e_1 - 1)/e_2$$

  - Eve can then compute the message from

$$c_1^{t_1} c_2^{-t_2} = m^{e_1 t_1} m^{-e_2 t_2} = m^{1+e_2 t_2} m^{-e_2 t_2} = m^{1+e_2 t_2 - e_2 t_2} = m$$

Shared Modulus

- Example
  - Take the public keys as
    - $N = N_1 = N_2 = 18923$
    - $e_1 = 11,\ e_2 = 5$
  - Take the ciphertexts as
    - $c_1 = 1514,\ c_2 = 8189$
    - The associated plaintext is $m = 100$
  - Then $t_1 = 1$ and $t_2 = 2$
  - We can now compute the message from

$$c_1^{t_1} c_2^{-t_2} = 100 \pmod{N}$$

RSA Security

1. Reductions of One-way Functions
2. RSA Security
   1. RSA Encryption and the RSA Problem
   2. Knowledge of the Private Exponent and Factoring
   3. Knowledge of $\phi(N)$ and Factoring
   4. Use of a Shared Modulus
   5. **Use of a Small Public Exponent**
3. Physical Attacks

Small Public Exponent

- Suppose we have three users
  - With public moduli N_1, N_2, and N_3
  - All with public exponent $e = 3$
- Suppose someone sends them the same message m
- The attacker sees the messages
  - $c_1 = m^3 \pmod{N_1}$
  - $c_2 = m^3 \pmod{N_2}$
  - $c_3 = m^3 \pmod{N_3}$

Small Public Exponent

- Now the attacker, using the CRT, computes the solution to $X = c_i \pmod{N_i}$ to obtain $X \pmod{N_1N_2N_3}$
- But since $m^3 < N_1N_2N_3$ we must have

  $X = m^3$ over the integers
- Hence $m = X^{1/3}$
- This attack is interesting since we find the message without factoring the modulus
- This is an evidence that breaking RSA is easier than factoring

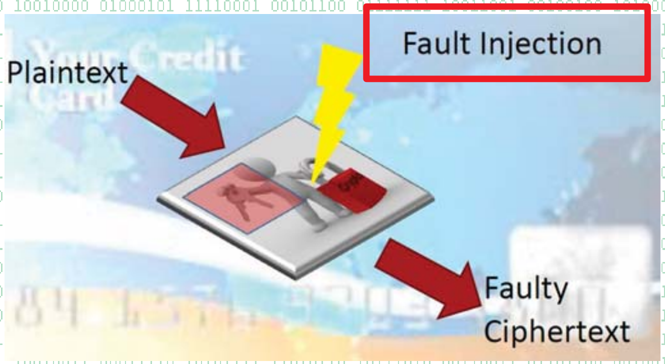
Small Public Exponent

- <u>Example</u>
  - Take $N_1 = 323$, $N_2 = 299$, $N_3 = 341$
  - The attacker sees $c_1 = 50$, $c_2 = 268$, $c_3 = 1$ and wants to determine the value of m
  - An attacker computes $X = 300763 \pmod{N_1N_2N_3}$ via CRT
  - The attacker computes $m = X^{1/3} = 67$ over $\mathbf{Z}$
  - Lesson
    - A given plaintext should be randomly padded
      - So no ciphertext is sent to two people
    - Very small exponents should be avoided for encryption

RSA Security

1. Reductions of One-way Functions
2. RSA Security
   1. RSA Encryption and the RSA Problem
   2. Knowledge of the Private Exponent and Factoring
   3. Knowledge of $\phi(N)$ and Factoring
   4. Use of a Shared Modulus
   5. Use of a Small Public Exponent
3. **Physical Attacks**

Fault Analysis



- How to reveal the secret?
- How to inject a fault?

RSA Decryption / Signature

- Efficient implementation splits exponentiation by Chinese Remainder Theorem (CRT)
  - $d_p = d \bmod (p-1)$
  - $d_q = d \bmod (q-1)$
  - $k = p^{-1} \bmod q$
  - $m_p = c^{d_p} \bmod p$
  - $m_q = c^{d_q} \bmod q$
  - $m = c^d \bmod n = p\,(k\,(m_q - m_p) \bmod q\,) + m_p$

RSA Decryption / Signature

- Inject a fault during CRT that corrupts m_q
  - m_q' is a corrupted result of m_q computation
  - $m' = p\,(k\,(m_q' - m_p) \bmod q\,) + m_p$
- Subtract m' from m
  - $m - m' = p\,(k(m_q - m_p) \bmod q) - p\,(k(m_q' - m_p) \bmod q)$
    $= p\,(x_1 - x_2)$
- Compute GCD
  - $\gcd(\,m - m', n\,) = \gcd(\,p(x_1 - x_2), pq\,) = p$
- Countermeasure: Check $c \equiv m^e$ before output it