6.857 Computer and Network Security October 3, 2002

Lecture Notes 9 : DSA/DSS, RSA, chosen-ciphertext attack

Lecturer: Ron Rivest *Scribe: Schmidt/Magdsick/Mazza/Baekkelund*

Outline

- Digital Signature Algorithm (DSA)
- "RSA" (modulo a prime)
- RSA (modulo a composite)

[These notes come from Fall 2001. Check with students' notes for new topics brought up in 2002.]

1 Digital Signature Algorithm (DSA)

DSA is a public key signature algorithm and is specified by the NIST's Digital Signature Standard[1] (DSS). DSA is used to create a small[2], publicly verifiable signature σ for a given message M.

The DSA has three components, key generation, signature creation, and signature verification.

[1] `http://www.nist.gov/public_affairs/releases/digsigst.htm`
[2] A small signature is good.

Key Generation: $q =$ some random 160-bit[3] prime
$p = qt + 1 \quad$ (1024 bits)
g with order q, g not a generator (order q, not $p-1$)
$x \in_R \{0, 1, ..., q-1\}$
$y = g^x (\mathrm{mod}\ p)$

public key : (p, q, g, y)
secret key : x

Signature Creation: for message M calculate $m = h(M)$
$k \in_R \{1, 2, ..., q-1\}$
$r = (g^k \bmod p) \bmod q$
$s = (m + rx)/k \ (\mathrm{mod}\ q)$

$\sigma = (r, s)$, where r and s are each 160 bits

Signature Verification: given M, p, y, r, s, q, g, h
check if $0 < r < q$ and $0 < s < q$
compute $w = s^{-1} (\mathrm{mod}\ q)$

check if $r = g^{wm} y^{rw} \ (\mathrm{mod}\ p) \bmod q$

Question : *Doesn't DSA specify SHA-1 as the hash function h?*
Answer : Yes, h is SHA-1 in DSA, so $length(m)$ is 160 bits.

Question : *I've heard that DSA has a subliminal channel...*
Answer : Yes, it is true that a malicious manufacturer could use a guess and check algorithm to have some control over the signature output by manipulating k. In this way, a DSA implementation could leak some bits of the secret key in each signature. The malicious manufacturer would only need to know the special encoding format to extract leaked bits of the key.[4]

Question : *What about the tightness or provability of the security of DSA?*
Answer : These schemes used in practice are not provably tight or anything like the theory. In essence, one can only reduce such claims of provability or tightness to "hard" problems like factoring or discrete logarithms.

2 "RSA" (modulo a prime)

This Mental Poker cryptosystem idea works perfectly! But can we actually find an encryption function that works this way...? Let's look at a simple version of RSA. (This scheme is actually due

[3] 160-bit numbers are chosen because square root attacks, like the solution to Problem 1-1, would still require brute-forcing through 2^{80} possibilities.

[4] Note that any implementation that allowed a malicious manufacturer to guess k (for instance using a poor pseudorandom number generator) would allow the manufacturer to extract the secret key x from just one signature. Also of note is Karl's clarification of this question and his answer to his own question, emailed out to the class list by the staff.

to Pohlig.)

Alice and Bob agree on a large, public prime p.

Consider $E(m)$ to be the function $m^e \pmod p$ for some secret key e and message m.

Since $(m^e)^{e'} = (m^{e'})^e$ for secret keys e and e', one can accomplish $D_A(D_B(E_A(E_B(m)))) = m$ like we wanted above.

Question : *When is this one-to-one (i.e. invertible)? ... and when it is, how do we decrypt it?*
Answer : Let's play with logarithms.

"RSA" (modulo a prime): let g be a generator (of order p)
If $m = g^k \pmod p$, then $m^e = g^{ke} \pmod p$

exponentiation by e is equivalent to multiplying the logarithm by e
$m \to m^e \pmod p$
$k \to ke \pmod{p-1}$

Question : *Well ... how do we decrypt it?*
Answer : To undo multiplication, multiply by the multiplicative inverse of e (call it d), modulo $(p-1)$. This is only possible if $gcd(e, p-1) = 1$ (verifiable by Extended Euclid Algorithm).

"RSA" (modulo a prime): $e \cdot d \equiv 1 \mod (p-1)$

Example:[5]
$6^{-1} \pmod{16}$ doesn't exist
$3^{-1} \pmod{16} = 11$

Theorem : If p is prime, $gcd(e, p-1) = 1$, $e \cdot d \equiv 1 \pmod{p-1}$
then $(m^e)^d \equiv m \pmod p$ holds $\forall m$

In this encryption algorithm, d and e are both private. Hence, it is not a public key cryptosystem. It is an interesting secret key encryption scheme.

Question : *So ... does this solve the poker problem?*
Answer : Yes! Back to poker ...

3 RSA (modulo a composite)

RSA was the first public key digital signature proposed. The space of elements for the message we want to encrypt and for the ciphertext are both the same: $\{0, 1, ..., n-1\}$ where $n = p \cdot q$ is the product of two randomly chosen large prime numbers p and q.

The Chinese Remainder Theorem tells us that everything happens in the field of $\pmod n$ happens simultaneously in the field $\pmod p$ and in the field $\pmod q$. In particular, if the following two

[5] The actual example given in class used 15 as the "$p-1$". 16 has many factors. This is a modified example, using $p = 17$, $p - 1 = 16$. Note also that both e and d must be odd in RSA or "RSA" (modulo a prime).

properties are verified:

$m^{ed} \equiv m \pmod p$
$m^{ed} \equiv m \pmod q$

Then

$m^{ed} \equiv m \pmod n$

(More generally, if $x \equiv y \pmod p$ and $x \equiv y \pmod q$ then $x \equiv y \ (mod\ (p \cdot q))$.)

The three steps of RSA are:

Key Generation:	Generate two random primes numbers p, q $s.t.$ $p \neq q$ Compute $n = p \cdot q$ Pick random e $s.t.$ $1 < e < (p-1) \cdot (q-1)$ and $gcd(e, (p-1), (q-1)) = 1$ Compute d $s.t.$ $1 < d < (p-1) \cdot (q-1)$ and $e \cdot d \equiv 1 \pmod{(p-1) \cdot (q-1)}$ The public key is (n, e) and the private key is d.
Signature Generation:	Compute $\sigma = m^d \pmod n$ The signature for m is σ
Signature Verification:	Given the public key (n, e) Verify that $\sigma^e \pmod n = m$

The security of this algorithm is completely dependent upon the difficulty of the mathematical problem of factorization[6]. If an adversary were able to factor n to $(p-1) \cdot (q-1)$, then using the Extended Euclidean Algorithm, the adversary could easily calculate d from $(p-1) \cdot (q-1)$ and e. Even today, twenty years after RSA's conception, the best factoring algorithm only has the following running time $e^{(1.93 \cdot (ln\ n)^{1/3} \cdot (ln\ (ln\ n))^{2/3})}$ and thus cracking RSA remains computationally infeasible for large values of n.

One last note is that the multiplicative property of this signature scheme is a double-edged sword. While it can help us solve problems like that of Mental Poker, at times the multiplicative property can cause trouble when an exact multiple of a message would still make sense in the message domain.

[6]However, no one has shown that breaking RSA implies factoring is easy.