A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

This is a summary of the paper "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" by Ronald L. Rivest. (1978)

Abstract

This encryption method has two important consequences.

1. The encryption key can be publicly revealed. Only the person who has the decryption key can decipher the message.

2. A message can be signed by a private decryption key. Anyone can verify this signature using the public encryption key.

The algorithm

M : a number representing the message

p, q : secret large prime numbers

n : product of p, q

When $ed \equiv 1 \pmod{(p-1)(q-1)}$, use e to encrypt and d to decrypt.

1 Introduction

Two properties should be preserved.

messages are private

messages can be signed

This method is an implementation of a "public-key cryptosystem".

2 Public-Key Cryptosystems

In a public key cryptosystem, there is an encryption procedure E and a decryption procedure D.

There are 4 properties.

1. $D\big(E(M)\big) = M$
2. E and D are easy to compute
3. revealing E does not reveal an easy way to compute D
4. $E\big(D(M)\big) = M$

Satisfying property 1~3 is a "trap-door one-way function".

If it also satisfies property 4, it is a "trap-door one-way permutation". Every message is the ciphertext for some other message and every ciphertext is a permissible message. Property 4 is needed only to implement signatures.

3 Privacy

Classical encryption methods had the "key distribution problem". A private transaction was needed to distribute the key before a private communication. However, A public-key crypto system's key can be distributed over insecure communication channels.

For example, Bob wants to send a private message *M* to Alice. First, Bob gets E_A from the public file. Then he sends the enciphered message $E_A(M)$. Alice deciphers the message by computing $D_A\big(E_A(M)\big) = M$. Alice can send a private response with E_B, which is also available in the public file.

No private transaction is needed between Alice and Bob. The only setup required is that each user should place their enciphering algorithm in the public file.

4 Signatures

The recipient of a signed message should have proof that the message originated from the sender. The recipient should be able to prove that the message was not forged by himself.

An electronic signature must be message-dependent and signer-dependent.

For example, Bob wants to send a signed message to Alice. First, Bob computes his signature *S* for the message using D_B, $S = D_B(M)$. Then he sends encrypted $E_A(S)$ to Alice (for privacy). He doesn't need to send *M* because it can be computed from *S*. Alice gets *S* by decrypting the ciphertext using D_A. Then she can extract the message by using E_B which is available on the public file, $M = E_B(S)$.

The message-signature pair (*M*, *S*) proves that the message is from Bob. Also, Alice cannot modify *M* into a different version, because she can't create the corresponding signature using D_B.

The problem is proving the integrity of the public file. This can be solved if the public file signs each message. Alice can check the signature with the public file's encryption algorithm E_{PF}. The problem of looking up E_{PF} itself is avoided by giving E_{PF} once when Bob first showed up in person. Finally, the ciphertext that Alice receives will be $D_{PF}(E_A(D_B(M)))$. Only a single secure meeting is needed between each user.

5 Our Encryption and Decryption Methods

1. Represent the message *M* as an integer between *0* and *n*-1. (Break the messages into a series of integer blocks.)
2. To encrypt, ciphertext $C \equiv E(M) \equiv M^e \ (mod\ n)$
3. To decrypt, $M \equiv D(C) \equiv C^d \ (mod\ n)$

Encryption doesn't increase the size of a message. Both the message and the ciphertext are integers in the range 0 to *n*-1.

The encryption key is a pair of positive integers (*e*, *n*), and the decryption key is a pair of positive integers (*d*, *n*). Each user makes the encryption key public, and keeps the decryption key private.

How to choose keys

Generate two large random primes *p* and *q*.

Compute *n* as the product of two primes *p* and *q*. (Although *n* is public, *p* and *q* should be hidden. Because it is difficult to factor *p* and *q* from *n*.)

$$n = pq$$

Pick an integer *d* to be a large random integer which is relatively prime to (*p*-1)(*q*-1).

$$\gcd\big(d, (p-1)(q-1)\big) = 1$$

Integer *e* is computed from p, q, and d to be the "multiplicative inverse" of d, modulo (*p*-1)(*q*-1).

$$ed \equiv 1 \ (mod \ (p-1)(q-1))$$

E and *D* are inverse permutations.

6 The Underlying Mathematics

The correctness of this algorithm is demonstrated using an identity due to Euler and Fermat.

By Fermat's little theorem, for any integer relatively prime to *n*,

$$M^{\varphi(n)} \equiv 1 \ (mod \ n)$$

Here $\varphi(n)$ is the Euler totient function giving the number of positive integers less than n which are relatively prime to *n*.

By the elementary properties of the Euler totient function,

$$\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1) = n - (p+1) + 1$$

Since *d* is relatively prime to $\varphi(n)$, it has a multiplicative inverse *e* in the ring of integers modulo $\varphi(n)$,

$$ed \equiv 1 \ (mod \ \varphi(n))$$

To prove $D\big(E(M)\big) = M$ and $E\big(D(M)\big) = M$,

$$D\big(E(M)\big) \equiv \big(E(M)\big)^d \equiv (M^e)^d \ (mod \ n) = M^{ed} \ (mod \ n)$$

$$E\big(D(M)\big) \equiv (D(M))^e \equiv (M^d)^e \ (mod \ n) = M^{ed} \ (mod \ n)$$

Using $ed \equiv 1 \ (mod \ \varphi(n))$, for some integer k,

$$ed = k\varphi(n) + 1$$

So,

$$M^{ed} \equiv M^{k\varphi(n)+1} \ (mod \ n)$$

Using $M^{\varphi(n)} \equiv 1 \ (mod \ n)$,

$$M^{p-1} \equiv 1 \ (mod \ p)$$

Since (p-1) divides $\varphi(n)$,

$$M^{k\varphi(n)+1} \equiv M \ (mod\ p)$$

This is trivially true when $M \equiv 0 \ (mod\ p)$. So, this holds for all *M*. Similarly, for q,

$$M^{k\varphi(n)+1} \equiv M \ (mod\ p)$$

Using there last two equations,

$$M^{ed} \equiv M^{k\varphi(n)+1} \equiv M \ (mod\ n)$$

This implies for all $M \ (0 \leq M < n)$. Therefore, *E* and *D* are inverse permutations.

7 Algorithms

Below are descriptions of an efficient algorithm to show that is method is practical.

How to Encrypt and Decrypt Efficiently

Computing $M^e \ (mod)$ requires at most $2log_2(e)$ multiplications and $2log_2(e)$ divisions using the procedure below. (decryption is similar)

1. The binary representation of *e* is $e_k e_{k-1} \cdots e_1 e_0$.
2. Set variable $C = 1$.
3. Repeat this step for $i = k, k-1, \cdots, 1, 0$. $C = C^2 \ (mod\ n)$. Also, if $e_i = 1$, $C = CM \ (mod\ n)$.
4. *C* is the encrypted form of *M*.

This procedure is the "exponentiation by repeated squaring and multiplication". This is half as good as the best.

The encryption time per block increases no faster than the cube of the numbers of digits in *n*.

How to Find Large Prime Numbers

Each user must privately choose two large random prime numbers *p* and *q*. These numbers should be large enough so that someone can't factor $n = pq$ easily using public *n*.

To find a large prime number, generate larger random numbers until a prime number is found.

To test a larger number for primality, use the "probabilistic" algorithm. To test a number *b*, pick a random number *a* from $\{1, 2, \cdots, b-1\}$. Then test whether

$$\gcd(a, b) = 1 \ and \ J(a, b) = a^{(b-1)/2} \ (mod\ b)$$

Where $J(a, b)$ is the Jacobi symbol.

If *b* is prime, this is always true. If *b* is composite, this will be false with probability at least 1/2. If this holds for many values of *a*, then *b* is almost certainly prime. However, here is a small chance that *b* is composite. (the receiver would detect that the decryption didn't work correctly)

If b is odd, $a \leq b$, and $\gcd(a, b) = 1$, $J(a, b)$ has a value in $\{-1, 1\}$ and can be efficiently computed.

Also, to gain protection from factoring algorithms, p and q should differ in length by a few digits, (p-1) and (q-1) should contain large prime factors, and $\gcd((p-1),(q-1))$ should be small. These conditions are easily checked.

To find a prime number p such that (p-1) has a larger prime factor, generate a larger prime number u, then let p be the first prime in the sequence $iu+1$ ($i=2,4,6,\cdots$). Additionally, it is more secure if (u-1) has a prime factor.

How to Choose *d*

It is easy to choose a number d relatively prime to $\varphi(n)$. Any prime number greater than $\max(p,q)$ will do. (d should be large enough for security.)

How to Compute *e* from *d* and $\varphi(n)$

To compute e, use a variation of Euclid's algorithm for computing $\gcd(\varphi(d),d)$.

Calculate $\gcd(\varphi(d),d)$ by computing a series $x_0, x_1, x_2, \cdots$. Where $x_0=\varphi(n), x_1=d, x_{i+1}\equiv x_{i-1}\ (mod\ x_i)$. Compute until a x_k equal to 0 is found. Then $\gcd(x_0,x_1)=x_{k-1}$.

For numbers a_i and b_i which satisfies $x_i=a_ix_0+b_ix_1$, if $x_{k-1}=1$, then b_{k-1} is the multiplicative inverse of $x_1\ (mod\ x_0)$. (Because k is less than $2log_2(n)$, this computation will be fast.)

If e is less than $log_2(n)$, start over by choosing another d. This guarantees that every encrypted message goes through reduction modulo n.

8 A Small Example

$p=47$

$q=59$

$n=pq=47\cdot 59=2773$

$d=157$

$\varphi(2773)=46\cdot 58=2668$

Then, e can be computed as follows.

$x_0=2668, a_0=1, b_0=0$

$x_1=157, a_1=0, b_1=1$

$x_2=156, a_2=1, b_2=-16\ (since\ 2668=157\cdot 16+156)$

$x_3=1, a_3=-1, b_3=17\ (since\ 157=1\cdot 156+1)$

Therefore, $e=17$. (multiplication inverse $(mod\ 2668)$ of d)

With $n=2773$, message can be encoded two letters per block, substituting a two-digit number for each letter. ($blank=0, A=01, B=02, \cdots, Z=26$)

The message “HELLO WORLD” is encoded into

$$0805\ 1212\ 1500\ 2315\ 1812\ 0400$$

Since $e = 10001_{(2)}$, the first block (0805) is enciphered,

$$M^{17} \equiv ((((1^2 \cdot M)^2)^2)^2)^2 \cdot M \equiv 1601\ (mod\ 2773)$$

Repeat for other blocks.

To decipher this *M*,

$$1601^{157} \equiv 805\ (mod\ 2773)$$

9 Security of the Method: Cryptanalytic Approaches

No techniques exist to prove the security of an encryption scheme. The only way is to test whether anyone can think of a way to break it.

Breaking our system is at least as difficult as factoring *n*. There are factoring algorithms such as Fermat’s and Legendre’s. But there is no algorithm that can factor a large number in a reasonable amount of time.

There are ways to determine the private decryption key from the public encryption key in the following sections.

Factoring *n*

The factors of *n* enable to compute $\varphi(n)$ and *d*.

The fastest factoring algorithm known can factor *n* in approximately

$$exp\sqrt{\ln(n) \cdot \ln(\ln(n))} = n^{\sqrt{lnln(n)/\ln(n)}} = (\ln(n))^{\sqrt{\ln(n)/\ln(\ln(n))}}$$

If *n* is 200 digits, about 1.2×10^{23} operations are needed.

Computing $\varphi(n)$ Without Factoring *n*

If $\varphi(n)$ can be computed, *d* can be computed as the multiplicative inverse of *e* modulo $\varphi(n)$.

Computing $\varphi(n)$ is no easier than factoring *n*, because *n* can be easily factored using $\varphi(n)$. However, this is not reasonable.

To factor *n* using $\varphi(n)$,

$(p+q)$ is obtained from *n* and $\varphi(n) = n - (p+q) + 1$.

$(p-q)$ is the square root of $(p+q)^2 - 4n$

q is half the difference of $(p+q)$ and $(p-q)$

This is why n must be composite. $\varphi(n)$ is too easy to compute if n is prime.

Determining *d* Without Factoring *n* or Computing $\varphi(n)$

d should be chosen from a large enough set so that a direct search for it is unfeasible.

Computing d is no easier than factoring n, because n can be easily factored using d. However, this is not reasonable.

To factor n using d,

$e \cdot (d-1)$ is a multiple of $\varphi(n)$

n can be factored using any multiple of $\varphi(n)$ (shown by Miller)

Also, a cryptanalyst might try to find d' which is equivalent to d secretly held by a user of the public-key cryptosystem. If d' was common, it could be found by a brute-force search. However, all d' differ by $lcm((p-1),\ (q-1))$.

Computing *D* in Some Other Way

Any general way of breaking this scheme must be as difficult as a factoring algorithm.

However, this conjecture is not proved.

10 Avoiding "Reblocking" When Encrypting A Signed Message

A signed message has to be "reblocked" for encryption when the signature n is larger than the encryption n. This can be avoided using a threshold value h. A threshold value h is chosen for the public-key crypto system.

Every user has two public key pairs (e, n). One for enciphering and the other for signature-verification. If every signature n is less than h, and every encryption n is greater than h, reblocking is not needed. The message is blocked according to the transmitter's signature n.

Using another solution, every use has a single public key pair. n is between h and $2h$. A message is encoded in numbers less than h. However, if the ciphertext is greater than h, it is repeatedly enciphered until it is less than h. (decryption is similar)

11 Conclusions

This is an implementation of a public-key cryptosystem which is a "trap-door one-way permutation" whose security rests in the difficulty of factoring large numbers. It permits secure communications without couriers to carry keys and permits to sign documents.

The security of this system needs to be examined more. Especially about factoring large numbers.