Inverse Problems, Cryptography and Security

Richard P. Brent
MSI and RSCS
ANU

13 April 2012

Cryptanalysis as an Inverse Problem

According to Wikipedia, *cryptanalysis* is the art of defeating cryptographic security systems, and gaining access to the contents of encrypted messages, without being given the cryptographic key.

We can think of encryption as a mapping $f : X \to Y$, where X is the space of *plain-texts*, and Y is the space of *cipher-texts*. For example, X and Y might both be the set of strings over some alphabet, or (after a simple encoding) the set of non-negative integers.

f should be an injection, so a unique inverse function $g : Y \to X$ exists. The recipient of an encrypted message $f(x)$ applies g to obtain the original message $x = g(f(x))$.

The cast

When discussing cryptography, the usual cast of characters includes:

- *Alice*, who is sending a message to Bob (or receiving a message from Bob).
- *Bob*, who is receiving a message from Alice (or sending a message to Alice).
- *Eve*, who is eavesdropping on the communications between Alice and Bob.

We'll assume that Eve can read the ciphertext that Alice sends to Bob, but can not change it. She is a "passive" eavesdropper.

More dangerous would be an "active" eavesdropper who could perhaps impersonate Bob to Alice and Alice to Bob (a "man in the middle" attack).

The key

Assume that Eve knows the encryption function f and the ciphertext y, and wants the plaintext x. One method is to try all $x' \in X$ until we find x' such that $y = f(x')$; then $x = x'$ is the plaintext. However, the search space X is usually so large that this method is impractical.

The cryptographic *key* is some information K which makes it easy to compute the inverse function g. For someone who does not know K (say Eve), computing g should be difficult. In other words, f is a “one-way” function – it is easy to compute f, but difficult to compute its inverse.

For someone who does know K (say Bob), computing g should be relatively easy.

Secret key cryptography

In *symmetric* or *secret key* cryptography, Alice and Bob share a secret key K. For example, if K is long enough, the encryption function might be just $f(x) = x \oplus K$ (bit-wise exclusive or). Then the decryption function $g = f$, since

$$(x \oplus K) \oplus K = x.$$

This is fine, except that Alice and Bob have to know K in advance, and K needs to be as long as the message x.

K is called a *one-time pad*. It is dangerous to reuse a one-time pad, since

$$(x_1 \oplus K) \oplus (x_2 \oplus K) = x_1 \oplus x_2,$$

and there is enough redundancy in English text that knowing $x_1 \oplus x_2$ is usually sufficient to give most of x_1 and x_2.

Public key cryptography

In *asymmetric* or *public key* cryptography, Bob “publishes” his *public key* K (so both Alice and Eve know K). Bob also has a *secret key* S, which he should keep secret (so neither Alice nor Eve should know S).

When Alice sends a message to Bob, she encrypts it using Bob’s public key, i.e. the encrypted message is

$$y = f_K(x).$$

Bob can decrypt Alice’s message using his secret key:

$$x = g_S(y).$$

Thus g_S is the inverse function of f_K.

Eve knows f_K and y, but it is difficult for her to compute x without knowing the secret key S.

Finding suitable functions f_K and g_S

It is not so easy to find functions f_K and g_S that meet our requirements. In fact, it has never been *proved* that such functions exist!

Nevertheless, a few pairs of functions are *believed* to satisfy our requirements. The simplest is used in the *RSA* cryptosystem, named after its inventors[1] Rivest, Shamir and Adleman (*not* Robert Scott Anderssen).

Another relatively simple algorithm is used in the *El Gamal* cryptosystem[2]. It is closely related to the Diffie-Hellman key exchange protocol, and can be generalised to give the basis of *elliptic curve cryptography*.

I will briefly outline the RSA and El Gamal cryptosystems.

[1] R, S & A were not have been the first to discover the algorithm, but they were the first to make it public – shades of Newton and Leibniz!

[2] Named after its inventor Taher El Gamal.

Euler's totient function

If n is a positive integer, then Euler's *totient* or *phi* function $\phi(n)$ is defined by

$$\phi(n) = |\{k : 1 \le k \le n,\ (k,n) = 1\}|,$$

where (k, n) is the greatest common divisor of k and n.

$\phi(n)$ is a multiplicative function, and

$$\phi(p^\alpha) = p^\alpha \left(1 - \frac{1}{p}\right)$$

for prime p, so it is easy to compute

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

if we know the prime factors of n. (The product is taken over all primes p dividing n.)

A theorem of Euler

If $(a, n) = 1$, then

$$a^{\phi(n)} = 1 \bmod n.$$

This follows from the fact that $\phi(n)$ is the order of the multiplicative group of integers modulo n.

In the case that n is a prime, it is just *Fermat's little theorem*

$$a^{n-1} = 1 \bmod n.$$

In the application to RSA cryptography, $n = pq$ is the product of two distinct primes, and $\phi(n) = (p-1)(q-1)$.

The RSA cryptosystem – Bob's setup

Bob chooses two distinct, large primes p and q, computes a "modulus" $n = pq$, and chooses an "encryption exponent" $e > 1$ such that $(e, \phi(n)) = 1$. He publishes his public key (n, e).

Bob uses the extended Euclidean algorithm to compute his "decryption exponent" d, an integer such that $de = 1 \bmod \phi(n)$. He keeps (n, d) as his private key.

At this point Bob can forget p and q (although he might keep them for reasons of efficiency). In any case, d, p, q and $\phi(n)$ must be kept secret.

RSA encryption

In order to send a message to Bob, Alice first encodes it as an integer x in the range $0 < x < n$ (splitting the message into blocks if necessary, and padding to ensure that $x^3 > n$).

Using Bob's encryption exponent e (which is public knowledge), Alice computes the ciphertext

$$y = x^e \bmod n.$$

Note that modular exponentiation can be done quickly using the binary representation of the exponent e.

RSA decryption

Bob receives the ciphertext y from Alice. Using his secret key d, he computes

$$x = y^d \bmod n.$$

He can then decode the integer x to obtain the original message.

Why does this work? Because

$$y^d = x^{de} \bmod n,$$

but $de = 1 \bmod \phi(n)$, so by Euler's theorem

$$x^{de} = x \bmod n.$$

Security of RSA – How hard is the inverse problem?

If Eve can factor *n* to find *p* and *q*, then she can compute $\phi(n)$ and the decryption exponent *d* in the same way that Bob did.

Hence, *p* and *q* should be chosen large enough that their product $n = pq$ is "impossible" to factor using existing algorithms and computer hardware. Nowadays, one should probably choose *p* and *q* to be primes of at least 512 bits, so *n* has at least 1024 bits. (A number of 768 bits, or 232 decimal digits, was factored by Kleinjung et al in 2010; the prime factors each have 116 decimal digits.)

There is no *proof* that factoring is hard – if $P = NP$, then factoring can be done in polynomial time. The best known algorithms are not polynomial (in $\lambda = \log n$), but they are not "fully" exponential. The number field sieve algorithm has running time

$$\exp(\lambda^{1/3+\varepsilon}).$$

Pitfalls

There are many pitfalls related to the practical use of RSA. Here are three of them:

- If the message x is too small, say $x < n^{1/e}$, then the ciphertext is just $y = x^e$, so Eve can find x by taking an e-th root.
- If Bob does not keep $\phi(n)$ secret, then Eve knows both $p + q = n + 1 - \phi(n)$ and $pq = n$, so she can find p and q by solving a quadratic equation.
- The ciphertext is a single-valued function of the plain-text. Thus, if Alice is sending simple yes/no messages like “buy” or “sell”, Eve will soon figure out which is which. A solution is to include some random padding in each message.

What if?

What if someone found a fast algorithm for integer factorization? (And did not keep it secret!)

Would that be the end of public key cryptography?

Not necessarily, because there are public key cryptosystems that are not based on the assumed difficulty of integer factorization.

One such is the El Gamal cryptosystem, which is based on the assumed difficult of the discrete logarithm problem (which I will define soon).

Maybe integer factorization is easy, but finding discrete logarithms is hard (or vice versa – we don't know).

The discrete logarithm problem

Suppose that p is a fixed prime, and g a primitive root mod p, i.e. a generator of the multiplicative group of integers mod p.

Consider x, y such that $y = g^x$ mod p. Since $g^{p-1} = 1$ mod p, we assume that $0 \le x < p - 1$ and $0 < y < p$.

If x is given, it is easy to compute y (this is just modular exponentiation, as used in the RSA algorithm, although for a prime modulus).

On the other hand, if y is given, it is not obvious how to compute x efficiently.

This is the *discrete logarithm problem* – the name comes from the analogy with the real case ($x = \log_g y$).

Diffie-Hellman key exchange

Before describing the El Gamal cryptosystem I will briefly mention the related *Diffie-Hellman key exchange protocol*.

Alice and Bob choose a prime modulus p and a primitive root g. They are not secret, so we can assume that Eve knows them.

Alice generates a random integer x, computes $a = g^x \bmod p$, and sends it to Bob.

Bob generates a random integer y, computes $b = g^y \bmod p$, and sends it to Alice.

Alice computes $K = b^x \bmod p$ and Bob computes $K = a^y \bmod p$. These two integers are equal because

$$b^x = g^{yx} = g^{xy} = a^y \bmod p.$$

Alice and Bob could now communicate using symmetric cryptography with K as a shared key.

What about Eve?

Eve knows $g, p, g^x \mod p$, and $g^y \mod p$. She needs to know $K = g^{xy} \mod p$. One way to do this is to solve a discrete logarithm problem to find x (or y).

There may be a way to find K without first finding x or y (this is called the *Diffie-Hellman problem*). However, it is plausible that finding K is no easier than solving a discrete logarithm problem.

The situation is analogous with RSA – it might be possible for Eve to crack RSA without factoring the modulus, but it is not obvious how she can do so.

The El Gamal cryptosystem

El Gamal is similar to Diffie-Hellman. Bob chooses a prime p and a primitive root g. He also chooses a random integer y, and computes $b = g^y \bmod p$. His public key is (g, p, b) and his secret key is (g, p, y).

For Alice to send a message to Bob, she chooses a random integer x, computes $a = g^x \bmod p$ and $c = mb^x \bmod p$ (we assume that she encoded the message as an integer $m \in (0, p)$, using some straightforward scheme).
The ciphertext is (a, c).

Bob receives (a, c), computes $a^y \bmod p$, and uses the fact that $a^y = b^x = g^{xy} \bmod p$ (as in Diffie-Hellman). Thus, he can compute $m = (a^y)^{-1} c \bmod p$.

Eve is stuck, unless she can solve the Diffie-Hellman problem or guess the random numbers used by Alice or Bob.

Elliptic curve cryptography

Elliptic curve cryptography (ECC) is based on the fact that the discrete logarithm problem can be defined in any group. El Gamal uses the multiplicative group of a finite field $\mathbb{Z}/p\mathbb{Z}$. However, we could also use the group defined in the usual way on an elliptic curve over a finite field.

Why do this? The main motivation is that there is an *index calculus* algorithm for solving the discrete logarithm problem over a finite field in sub-exponential time (though not polynomial time) – analogous to the number field sieve algorithm for integer factorization. The index calculus algorithm does not seem to generalise to the elliptic curve case.

Thus, if the parameters are chosen to give the same level of security, we can get by with shorter keys (and faster algorithms) using ECC. Of course, there are pitfalls $\cdots$

References

1. W. Diffie and M. E. Hellman, New directions in cryptography, *IEEE Trans. Inf. Theory* IT-22 (1976), 644–654.

2. T. El Gamal, A public-key cryptsystem and a signature scheme based on discrete logarithms, *IEEE Trans. Inf. Theory* 31 (1985), 469–472.

3. N. Koblitz, Elliptic curve cryptosystems, *Math. Comp.* 48 (1987), 203–209.

4. A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

5. V. Miller, Uses of elliptic curves in cryptography, *Proc. Crypto 85*, 417–426.

6. R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Comm. ACM* 21 (1978), 120–126.