C&RSA&DHM algorithms – the shortests course. . .

Przemysław Śliwiński

Part I

RSA & Cocks

Algorithm 1 (Rivest, Shamir, Adleman, 1977 & Cocks 1973) *Let p and q be a pair of primes. Let $n = pq$. We will need the value of the* Euler's totient function *$\varphi(n)$, that yields a number of coprime numbers less than n. It happens that for two primes, p, q, the formula is particularly simple,* i.e. *we have that*

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

To create the first pair of encryption keys we need to find any *number $e : e \perp \varphi(n)$, that is, that e and $\varphi(n)$ are relatively prime,* i.e. *e is coprime with $\varphi(n)$ (their* greatest common divisor *equals one;* viz. $\gcd(e, \varphi(n)) = 1$*). The pair (e, n) is the first (either* public *or* private*) key. To get the other key we need to find a number d, which is a* multiplicative inverse *of e modulo $\varphi(n)$,* i.e. *we are looking for d such that*[1]

$$(d \cdot e) \bmod \varphi(n) = 1.$$

The pair (d, n) is the other (either private *or* public*) key. Now, to encode (encrypt) the message m, it suffices to perform a single operation employing one key (e, n)*

$$c = m^e \bmod n.$$

To decipher (decrypt) it, one just needs the corresponding pair (d, n) since

$$m = c^d \bmod n.$$

[1]Operation $a \bmod p$ is just a rest of integer division of a by p, that is $a \bmod p = a - \left\lfloor \frac{a}{p} \right\rfloor p$.

Proof. In order to verify the RSA algorithm, we merely need to verify the last equality, that is, the fact that

$$\left(c^d \bmod n\right)^e \bmod n = m^{ed} \bmod n = m.$$

To this end, we need to recall the following properties of the RNS arithmetic:

1. The *Fermat Little Theorem* stating that if $a \perp p$ then $a^{p-1} \bmod p = 1$.

2. The *Chinese Remainder Theorem* which states that for $N = n_1 \cdot \dots \cdot n_r$, where $n_i \perp n_j$ for $i \neq j$, and for any $k, l < N$, we have the following equivalences

   $$k \equiv l \bmod N \Leftrightarrow k \equiv l \bmod n_i \text{ and for each } i = 1, \dots, r.$$

   Clearly, in our case, that is for $r = 2$, it reduces to a simple logical conjuction

   $$k \equiv l \bmod pq \Leftrightarrow k \equiv l \bmod p \wedge k \equiv l \bmod q.$$

3. The basic fact that $a^p \bmod q = (a \bmod q)^p \bmod q = (\prod_{i=1}^{p} a \bmod q) \bmod q = \underbrace{(a \bmod q \cdots a \bmod q)}_{p \text{ times}} \bmod q$.

4. The observation that $ed - 1 = h\varphi(pq)$ for some (unknown) $h \in \mathcal{N}$, which holds because $ed \bmod \varphi(pq) = 1$, and thus $(ed \bmod \varphi(pq) - 1) \bmod \varphi(pq) = 0 = h\varphi(pq))$

5. Eventually:

   $$\begin{aligned} m^{ed} \bmod q &= \left(m^{ed-1}\right) m \bmod q \\ &= \left(m^{h\varphi(pq)}\right) m \bmod q \\ &= \left(m^{h(p-1)(q-1)}\right) m \bmod q \\ &= \left(m^{(q-1)}\right)^{h(p-1)} m \bmod q \\ &= (1)^{h(p-1)} m \bmod q = m \bmod q. \end{aligned}$$

6. Verification that $m^{ed} \bmod p = m \bmod p$ is left to the reader...

■

Algorithm 2 (Digital signature *aka* fingerprint) *Compute a* hash value *of* m *and encrypt it using a private key* (e, n) *or* (d, n).

Example 3 *Let $p = 11$ and $q = 7$. Then $n = pq = 77$ and the corresponding totient function of n, is*

$$\varphi(n) = (p-1)(q-1) = 60.$$

Let now $e = 17$. Indeed, $\gcd(17, 60) = 1$, that is, both numbers are relatively prime. Clearly, $d = 53$. So the keys are $(17, 77)$ and $(53, 77)$. Let $m = 61$, then $c = 61^{17} \bmod 77 = 52$ and $52^{53} \bmod 77 = 61 = m$.

To compute both *gcd* and *multiplicative inverse* one can use the *compilation-time recursive template programming* tricks:

```
template <int k, int l>
struct gcd
{
   enum { value = gcd<l, k % l>::value };
};
template <int k>
struct gcd<k, 0>
{
   enum { value = k };
};


template <long  w, long  M, long  k = w, long = 0>
struct mul_inv
{
   enum
   {
      res = (k - 1) * M % w
   };
   enum
   {
      mi = mul_inv<w, M, k - 1, res>::mi
   };
};
template <long w, long M, long k>
struct mul_inv<w, M, k, 1>
{
   enum {  mi = k  };
};
template <long  M, long k, long res>
struct mul_inv<1, M, k, res>
{
   enum {  mi = 1  };
};
```

Example 4 (Multiplication in a cloud) *Let again* $p = 11$ *and* $q = 7$. *Then* $n = pq = 77$ *and the corresponding totient function of* n, *is*

$$\varphi(n) = (p-1)(q-1) = 60.$$

Let now $e = 17$. *Indeed,* $\gcd(17, 60) = 1$, *that is, both numbers are relatively prime. Clearly, the multiplicative inverse,* $d = 53$. *So the public key is* $(17, 77)$ *and the private one is* $(53, 77)$.

1. Let $m_1 = 11$ and $m_2 = 3$, then their **encoded** values (using a public key) are

   $$c_1 = 11^{17} \bmod 77 = 44 \text{ and } c_2 = 3^{17} \bmod 77 = 75,$$

   respectively.

2. Now c_1 and c_2 are sent to the cloud and there they are multiplied there

   $$c_{12} = 44 * 75 = 3300.$$

3. The product $c_{12} = 3300$ (or $3300 \bmod 77 = 66$) is sent back and we can **decode** it (using a private key) as if it is a message, so $m = 3300^{53} \bmod 77 = 33$ (or, equivalently, $66^{53} \bmod 77 = 33$).

Part II

DHM & Cocks

Definition 5 *A number g is a* primitive root modulo p *if every number n, coprime to p, is congruent to a power of g modulo p. If p is prime, then powers of g* generate *all numbers $1, \ldots, p-1$ (albeit in a 'random' order).*

Algorithm 6 (Diffie–Hellman–Merkle, 1976 & Cocks, 1969) *Let Alice and Bob publicly select p and g, and* (in pectore) *the private numbers a and b. Then, they compute the public messages*

$$A = g^a \bmod p \text{ and } B = g^b \bmod p,$$

send them to each other, and compute their common secret key s

$$s = B^a \bmod p \text{ and } s = A^b \bmod p.$$

Example 7 *Let $p = 23$ and $g = 5$ (verify that g is indeed a primitive root modulo p). Alice chooses $a = 11$ and Bob $b = 8$. Hence, Alice sends*

$$A = 5^{11} \bmod 23 = 22$$

and Bob sends

$$B = 5^8 \bmod 23 = 16.$$

Then Alice and Bob compute

$$s = 16^{11} \bmod 23 = 1 \text{ and } s = 22^8 \bmod 23 = 1,$$

and both have the same (random) secret number $s = 1$, which they can use as a key in e.g. AES.

Proof. Observe that

$$\begin{aligned} s &= A^b \bmod p \\ &= (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p \\ &= \left(g^b \bmod p\right)^a = B^a \bmod p. \end{aligned}$$

■

Part III

Hamming codes (no Cooks!)

TBF. . .