Basic Cryptography Solutions

Computer Algebra for Cryptography (B-KUL-H0E74A)

Exercise 1.

- ```
  function RSAKeyGen(b)
    bound := Round(2^(b/2));
    p := NextPrime(Random(bound div 2, bound) : Proof := false);
    q := NextPrime(Random(bound div 2, bound) : Proof := false);
    N := p*q;
    EulerPhi := (p-1)*(q-1);
    repeat e := Random(EulerPhi); until GCD(EulerPhi, e) eq 1;
    _, d, _ := XGCD(e, EulerPhi);
    return N, e, d mod EulerPhi;
  end function;
  ```
- ```
  function RSAEncrypt(m, N, e)
    return Integers() ! ((Integers(N) ! m)^e);
  end function;
  ```
- ```
  function RSADecrypt(c, N, d)
    return Integers() ! ((Integers(N) ! c)^d);
  end function;
  ```
- ```
  N, e, d := RSAKeyGen(2048);
  m := Random(N);
  m eq RSADecrypt(RSAEncrypt(m, N, e), N, d);
  ```
- Remark that using the Chinese Remainder Theorem requires knowing the factorization of $N = pq$, hence can not be used to encrypt a message since in that case only the public key is known. To obtain p and q from N, e and d one could use the following function:

  ```
  function GetFactors(N, e, d)
    k := e*d-1;
    val2 := Valuation(k,2); // compute max power of 2 that divides k
    k := k div 2^val2;
    ZN := Integers(N);
    repeat
  ```

```
      x := Integers() ! ((Random(ZN))^k);
      p := GCD(x-1, N);
    until (p ne 1) and (p ne N);
    return p, N div p;
  end function;
```

Alternatively, one could just alternate the key generation to also return p and q since from a security point of view, knowing either d or p, q are equivalent anyway.

```
  function RSADecryptCRT(c, N, d, e)
    p, q := GetFactors(N, e, d);
    mq := Integers() ! ((Integers(q) ! c)^(d mod (q-1)));
    mp := Integers() ! ((Integers(p) ! c)^(d mod (p-1)));
    return CRT([mq,mp],[q,p]);
  end function;
```

Exercise 2.

```
function SquareMult(a,n)

  if a eq 0 then
    if n lt 0 then error "Negative power of non-invertible element"; end if;
    return Parent(a) ! 0;   // make sure output has same type
  end if;
  if n eq 0 then return Parent(a) ! 1;  end if;
  if n lt 0 then return SquareMult(1/a,-n); end if;

  powa := a;
  res := Parent(a) ! 1;
  e := n;

  while e gt 0 do
    if (e mod 2) eq 1 then res := res*powa; end if;
    powa := powa^2;
    e := e div 2;
  end while;

  return res;
end function;
```

To check the speed of this algorithm compared to the built-in function, we can use the command `Cputime()`.

```
F := GF(NextPrime(2^1000 : Proof := false));
```

```
t := Cputime();
for i := 1 to 1000 do
  A := Random(F)^Random(2^1000);
end for;
Cputime(t);
t := Cputime();
for i := 1 to 1000 do
  A := SquareMult(Random(F),Random(2^1000));
end for;
Cputime(t);
```

Exercise 3.

- ```
  function StringToHill(s)
    if not Regexp("^[A-Z]+$",s) then
      error "The input string should consist of only capital letters.";
    else
      return [Integers(26) ! StringToCode(Eltseq(s)[i])-65 : i in [1..#s]];
    end if;
  end function;
  ```

- ```
  function HillToString(a)
    return &cat[CodeToString(Integers() ! a[i]+65) : i in [1..#a]];
  end function;
  ```

  Remark that coercion in Magma has higher priority when it comes to order of operations than addition.

- ```
  function HillKeyGen(s)
    isSqr, sqrt := IsSquare(#s);
    if isSqr then
      return Matrix(Integers(26), sqrt, StringToHill(s));
    else
      error "Wrong input length.";
    end if;
  end function;
  ```

- ```
  function HillEncrypt(s,A)
    k := #Rows(A);
    s := StringToHill(s);
    while #s mod k ne 0 do Append(~s, 0); end while;
    vecs := [];
    for i := 1 to (#s div k) do
      Append(~vecs, Matrix(Integers(26), k, 1, [s[j] : j in [k*(i-1)+1..k*i]]));
    end for;
    code := [];
    for v in vecs do
  ```

```
    code cat:= Eltseq(A*v);
  end for;
  return HillToString(code);
end function;
```

- ```
  function HillDecrypt(s,A)
      return HillEncrypt(s,A^-1);
  end function;
  ```

- The thing that can go wrong (or does go wrong with the given private key of length 25), is that the matrix A may not be invertible. This happens if the determinant of A is not coprime with 26, i.e. the determinant is a multiple of 2 or 13.

  A better choice would be to append the alphabet with characters; e.g. allowing both uppercase and lowercase letters and a white space would result in 53 possible characters. Given that this is a prime number, only matrices with determinant zero would not be invertible.
  ```