Cryptography I (2WC12)

Course Notes

Jos Wetzels

a.l.g.m.wetzels@student.utwente.nl

1 General

1.1 Extended Euclidian Algorithm

gcd(a, b): Greatest Common Divisor (GCD), ie. largest number which divides a and b without leaving remainder.

egcd(a, b): Gives gcd and Bezout's identity coefficients x and y, ie.: $aX + bY = gcd(a, b)$

The pseudocode is as follows:

```
extended_gcd(a, b)
    s := 0;    old_s := 1
    t := 1;    old_t := 0
    r := b;    old_r := a
    while r ≠ 0
        quotient := old_r div r
        (old_r, r) := (r, old_r - quotient * r)
        (old_s, s) := (s, old_s - quotient * s)
        (old_t, t) := (t, old_t - quotient * t)
    Print "Bézout coefficients:", (old_s, old_t)
    Print "greatest common divisor:", old_r
    Print "quotients by the gcd:", (t, s)
```

1.2 Euler's Phi

φ(n): Number of integers in $\mathbb{Z}_n$ relatively prime to n.

A naïve and rather inefficient method for computing Euler's Phi in pseudocode is as follows:

```
PHI(n):
  amount = 0
  For(K=1, K<=N, K++):
    If(GCD(N,K)==1):
      amount++
  Return amount
```

1.3 Euler's Theorem

Iff a and n are coprime (ie. *gcd(a, n)* = 1) positive integers then $a^{\varphi(n)}$ = 1 mod n. The theory can be used to easily reduce large powers modulo *n*. For example, consider $7^{222}\ mod\ 10$. Note that 7 and 10 are coprime and $\varphi(10) = 4$. By Euler's theorem we know $7^4 \equiv 1\ mod\ 10$ and we get $7^{222} \equiv 7^{4*55+2} \equiv 7^{4^{55}} * 7^2 \equiv 1^{55} * 7^2 \equiv 49 \equiv 9\ mod\ 10$.

1.4 Fermat's little theorem

Fermat's Little Theorem states that if p is a prime number then for any integer a the number $a^p - a$ is an integer multiple of p. This can be expressed as $a^p \equiv a\ (mod\ p)$.

If a is not divisible by p then the theorem is equivalent to the statement that $a^{p-1} - 1$ is an integer multiple of p, ie.: $a^{p-1} \equiv 1\ (mod\ p)$.

1.5 Chinese Remainder Theorem

The CRT is a result about congruences in number theory which states that given positive integers that are pairwise coprime $n_1, n_2, .., n_k$ then for any given sequence $a_1, a_2, .., a_k$ there exists an integer x solving the following system of linear congruences:

$$\begin{cases} x \equiv a_1\ mod\ n_1 \\ \dots \\ x \equiv a_k\ mod\ n_k \end{cases}$$

Furthermore all solutions x of this system are congruent modulo the product $N = n_1 n_2 .. n_k$ hence:

$$x \equiv y\ mod\ n_i, 1 \leq i \leq k \leftrightarrow x \equiv y\ mod\ N$$

Example: Consider the modular congruences (the lcm of whose moduli is 60):

$$\begin{cases} x \equiv 2 \; mod \; 3 \\ x \equiv 3 \; mod \; 4 \\ x \equiv 1 \; mod \; 5 \end{cases}$$

We can translate these to equations:

$$\begin{cases} x \equiv 2 + 3t \\ x \equiv 3 + 4s \\ x \equiv 1 + 5u \end{cases}$$

Start by substituting *x* from the first equation into the second congruence:

$$2 + 3t \equiv 3 \; mod \; 4$$
$$3t \equiv 1 \; mod \; 4$$
$$t \equiv 3^{-1} * 1 \equiv 3 \; mod \; 4$$

Which means that $t = 3 + 4s$ for some integer *s*. Substitute *t* into the first equation:

$$x = 2 + 3t = 2 + 3(3 + 4s) = 11 + 12s$$

Substitute this *x* into the third congruence:

$$11 + 12s \equiv 1 \; mod \; 5$$
$$1 + 2s \equiv 1 \; mod \; 5$$
$$2s \equiv 0 \; mod \; 5$$

Meaning that $s = 0 + 5u$ for some *u*. Finally we have:

$$x = 11 + 12s = 11 + 12(5u) = 11 + 60u$$

Hence:

$$x \equiv 11 \; mod \; 60$$

2 Modular Arithmetic

Modulo: a mod n, or a % n, is the remainder of dividing a by n.

Congruency: a and b are congruent modulo n, written as $a \equiv b \; mod \; n$, iff n divides their difference, ie. $n | abs(a - b)$. That is, the difference is a multiple of n.

Congruency class: The set of all *x* congruent to a.

2.1.1 Integer Rings

An integer ring $\mathbb{Z}_n$ consists of:

1. The set of integers $\mathbb{Z}_n = \{0,1,..,n-1\}$
2. The operations "+" and "*" under which the ring is *closed*, ie.:
 (a) $\forall a,b,c \in \mathbb{Z}_n: a+b \equiv c \bmod n$
 (b) $\forall a,b,d \in \mathbb{Z}_n: a*b \equiv d \bmod n$

The following holds for all integer rings:

- *Operation properties*: Distributive, associative and commutative, eg. $(a+b) \bmod n = (a \bmod n) + (b \bmod n)$. $(ab) \bmod n = (a \bmod n)(b \bmod n)$.
- *Special elements*: Neutral element 0 with respect to addition. Neutral element 1 with respect to multiplication.
- *Inverses*: Every element has an additive inverse but not every element has a multiplicative inverse.

2.1.2 Division

$\frac{a}{b} \bmod n = ((a \bmod n)(b^{-1} \bmod n)) \bmod n$. This is only defined if the multiplicative inverse of *b* exists.

2.1.3 Exponentiation

Given $c \equiv a^b \bmod n$ there are multiple approaches. For small numbers we can calculate *c* directly. For large values we can use the fact that: $(a*a*a) \bmod n = a * ((a*a) \bmod n) \bmod n$.

```
c = 1
For i=0 to b
  c = a*c mod n
```

2.1.3.1 Square-and-Multiply

Square-and-Multiply is a method for fast computation of large positive integer powers of a number. It works by iterating down from the exponent to 0 and applying multiplication whenever the current exponent iteration is even and squaring whenever it is odd.

```
SquareAndMultiply(B,P,M)://gives B^P mod M
  X=1
  While (P>0)
    If (P ≠ 0 mod 2)
      X=B*X
      X=X mod M
      P=P-1
    P=P/2
    B=B*B
    B=B mod M
  Return X
```

2.1.4 Square roots

The modular square root of an integer a modulo n is an integer r such that $r^2 \equiv a \bmod n$. An integer a is a *quadratic residue* modulo n if it is congruent to a *perfect square* modulo n, ie. if $\exists r: r \in \mathbb{Z} : r^2 \equiv a \bmod n$.

a) Prime modulus: When m is prime (or a prime power) we distinguish the following cases:

3. a is congruent to 0: $r = 0$
4. If a is not a *quadratic residue* modulo n: There are no square roots r for a.
5. If a is a *quadratic residue* modulo n: There are two square roots r for a. Both square roots are eachother's additive inverse (ie. their sum is 0 modulo n).

First we compute the *legendre symbol* $(a|n) = \begin{cases} -1: if\ (a^{\frac{n-1}{2}} \bmod n = (n-1)) \\ a^{(n-1)/2} \bmod n: Otherwise \end{cases}$

If $(a|n) = -1$ (ie. it is a *quadratic non-residue*) there is no solution, if it is 0 r = 0 (since a is congruent to 0). Assuming it is 1 (ie. it is a *quadratic residue*) we have the following cases:

- $n = 2$: The square root is congruent to r.
- $n \equiv 3 \bmod 4$: $r \equiv \pm a^{\frac{n+1}{4}} \bmod n$
- $n \equiv 5 \bmod 8$: $r \equiv \pm av(i-1) \bmod n$ where $v = (2a)^{(n-5)/8} \bmod n$ and $i = 2av^2 \bmod n$.
- $Other\ cases$: In other cases we will have to employ a method like the *Tonelli-Shanks algorithm* which is outside the scope of these notes.

b) Composite modulus: If we have a composite modulus we can factor it into prime powers and generate a solution using the *Chinese Remainder Theorem*.

Naïve approach: A naïve approach to finding the square roots of some integer A modulo n would be to simply use brute-force:

```
FindSQRT(A, N):
  For(I=0; I<N; I++)
    B = (I*I) mod N
    If(B=A):
      Return I
  Return Failure
```

2.1.5 Additive inverse

Given a, its modular additive inverse is b such that $a + b = 0\ mod\ n$ and can be calculated by $b = \big(n - (abs(a) mod\ n)\big) mod\ n$.

2.1.6 Multiplicative inverse

Given a, its modular multiplicative inverse a^{-1} is defined as $a^{-1} * a = 1\ mod\ n$ and can be calculated by $egcd(n, a) = xn + ya$ where y is the modular multiplicative inverse of a. If y is negative we simply set $y = y + n$.

Note: a^{-1} only exists *iff a* and *n* are coprime, ie. $gcd(a, n) = 1$.

3 Integer Factorization

Integer factorization, or prime factorization, is the decomposition of a composite number (a non-prime, ie. a number that has at least 1 positive divisor other than the number itself) into smaller *non-trivial* divisors which when multiplied together form the original integer.

Every positive integer has a unique prime factorization (or *prime decomposition*). Hence one can factor any integer down to its constituent *prime factors* by repeated application of a *general* algorithm. This is not always the case for *special-purpose* algorithms though since the algorithm might not apply (or apply only inefficiently) to the smaller factors.

3.1 Some Terminology

Smooth integer: An integer *n* is said to be *smooth* if it factors completely into small prime numbers. Specifically, it is B-*smooth* if none of its prime factors is bigger than B.

Powersmooth integer: An integer *n* is said to be *powersmooth* if it factors completely into small prime powers. Specifically, it is B-*powersmooth* if all prime powers p^x dividing *n* satisfy $p^x < B$.

3.2 Special-Purpose algorithms

A *special-purpose* algorithm's running time depends on the properties of the number to be factored or one of its unknown factors: size, form, etc. Some algorithms, called *first category* algorithms, such as trial division are applied before others to remove small factors.

Some special-purpose algorithms require a so-called *factor base* which is a small set of prime numbers which constitute the only prime factors of a generated set of numbers used in the algorithms.

```
TrialDivision(n):
    if n == 1:
      return [1]
    primes = prime_sieve(sqrt(n) + 1)
    prime_factors = []

    for p in primes:
        if p^2 > n:
          break
        while n % p == 0:
            prime_factors.append(p)
            n = (n / p)
    if n > 1:
      prime_factors.append(n)
    return prime_factors
```

3.2.1 Trial Division

Trial division is the most laborious but simplest integer factorization algorithm. Trial division tests if an integer *n* can be divided by each number in turn that is less than *n*. For example, for *n* = 12 the only numbers that divide it are 1,2,3,4,6 and 12 and selecting the largest prime-powers in the list gives us 12 = 3*4. Trial division can be optimized by only selecting for prime numbers as candidate factors (since we don't have to test 4 if *n* wasn't divisible by 2) and going no further than $\sqrt{n}$ (since if *n* is divisible by p then $n = p * q$ and if $q < p$ would hold q or a prime factor thereof would have been selected earlier. This yields the following pseudo-code:

3.2.2 Fermat Factorization

Fermat Factorization only works on odd integers and is based on the representation of an odd integer *n* as the difference of two squares $n = a^2 + b^2$ which is factorable as (a+b)(a-b). If neither factor is equals 1 it is a proper factorization of *n*. Each odd number has such a factorization, if $n = cd$ is a factorization then $n = (\frac{c+d}{2})^2 - (\frac{c-d}{2})^2$. Since *n* is odd so are *c* and *d* and the halves are integers. This yields the following pseudo-code algorithm:

```
FermatFactor(N):
  A = ceil(sqrt(N))
  B = (A^2) - N
  While(B is not a square):
    A++
    B = (A^2) - N
  Return ((A-sqrt(B)), (A+sqrt(B)))
```

Example: Given n = 5959:

Try:	1	2	3
A	78	79	80
$\mathbf{B^2}$	125	282	441
B	11.18	16.79	21

The third try produces the perfect square 441 so $a = 80, b = 21$ and the factors of n are $(a - b) = 59, (a + b) = 101$.

3.2.3 Dixon's Factorization Method

Dixon's method, also known as the *random squares method*, is based on finding a congruence of squares modulo the integer n we seek to factor. Dixon's method replaces Fermat's condition *"is the square of an integer"* with the weaker condition *"has only small prime factors"*.

Given a number n which we seek to factor we choose a bound B and identify the *factor base* P (the set of all primes less than or equal to B). Next we search for positive integers z such that $z^2 \ mod \ n$ is B-smooth. We can therefor write, for suitable exponents a_k:

$$z^2 \equiv \prod_{p_i \in P} {p_i}^{a_i} \ mod \ n$$

When we have generated enough relations (generally a few more than the size of P) we can use methods like Gaussian elimination to multiply together the relations in such a way that the exponents of the primes on the right-hand side are all even:

$${z_1}^2 {z_2}^2 .. {z_k}^2 \equiv \prod_{p_i \in P} {p_i}^{a_{i,1}+a_{i,2}+..+a_{i,k}} \ mod \ n$$

Where $a_{i,1} + a_{i,2} + .. + a_{i,k} \equiv 0 \ mod \ 2$. This gives us a congruence of squares of the form $a^2 \equiv b^2 \ mod \ n$ which we can turn in a factorization of $n = \gcd(a + b, n) * (\frac{n}{\gcd(a+b,n)})$. If this factorization turns out to be trivial (ie. $n = n * 1$) then we have to try again with a different combination of relations.

Example: Given $n = 84923$, $B = 7$ and $P = \{2,3,5,7\}$ we will factor n by searching for random integers $ceil(\sqrt{n}) \leq a \leq n$ whose squares are *B*-smooth. Suppose that two of the numbers we find are 513 and 537:

$$513^2 \ mod \ 84923 = 8400 = 2^4 * 3 * 5^2 * 7$$
$$537^2 \ mod \ 84923 = 33600 = 2^6 * 3 * 5^2 * 7$$

So:

$$(513 * 537)^2 \ mod \ 84923 = 2^{10} * 3^2 * 5^4 * 7^2$$

Then:

$$(513 * 537)^2 \ mod \ 84923 = 275481^2 \ mod \ 84923$$
$$= (84923 * 3 + 20712)^2 \ mod \ 84923$$
$$= (84923 * 3)^2 + 2 * (84923 * 3 * 20712) + 20712^2 \ mod \ 84923$$

$$= 0 + 0 + 20712^2 \; mod \; 84923$$

That is:

$$20712^2 \; mod \; 84923 = (2^5 * 3 * 5^2 * 7)^2 \; mod \; 84923 = 16800^2 \; mod \; 84923$$

The resulting factorization is:

$$84923 = \gcd(20712 - 16800, 84923) = 163 * 521$$

3.2.4 Pollard's ρ for integer factorization

Pollard's ρ algorithm factors an integer *n* and is based on Floyd's cycle-finding algorithm and the observation that (as in the birthday problem) t random numbers $\{x_1, .., x_t\}$ in the range [1,*n*] will contain a repetition with probability P > 0.5 if $t > 1.77n^{\frac{1}{2}}$. The ρ algorithm uses *g*(x), a polynomial modulo *n*, as a pseudorandom sequence generator.

The algorithm takes as input the integer *n* to be factored and the polynomial *g(x)*. The polynomial $g(x) = x^2 - 1 \; mod \; n$ was specified in the original algorithm but nowadays $g(x) = x^2 + 1 \; mod \; n$ is more common. The output is either a non-trivial (but not necessarily the smallest) factor of *n* or failure. The algorithm can fail even if *n* is composite in which case it can be rerun with a starting value other than 2 or a different *g(x)*.

```
RhoFactor(N, G):
  X = 2, Y = 2, D = 1
  While (D=1):
    X = G(X)
    Y = G(G(Y))
    D = GCD(|X-Y|, N)
  If (D=N):
    Return Failure
  Else:
    Return D
```

3.2.5 Pollard's p-1

Pollard's p-1 algorithm is a special-purpose integer factorization algorithm that finds factors of an integer *n* for which the preceding number (*p-1*) is *powersmooth*. Given prime factor *p* of our integer *n* we know that for all integers *a* coprime to *p* and for all positive integers *k*: $a^{k(p-1)} \equiv 1 \; mod \; p$. If a number *x* is congruent to 1 modulo a factor of *n* then $gcd(x - 1, n)$ will be divisible by that factor. We make the exponent $k(p - 1)$ a large multiple of $(p - 1)$ by making it a number with many prime factors, generally the product of all prime powers less than some limit B. Starting with a ran-

dom *x* we repeatedly replace it with $x^w \ mod \ n$ as *w* runs through those prime powers. Check at each stage or once at the end whether $gcd(x-1, n)$ is not equal to 1. Below are two reference pseudocodes:

```
P-1Factor(N):
  select a smoothness bound B
 label2:
  M = ∏_{primes q≤B} g^{floor(log_q B)}  // exponent M
  a = A coprime to n // can be random or fixed
  g = gcd(a^M − 1, n) // exponentiation can be done mod n
  if (1 < g < n):
    return g
  elseif(g = 1):
    select larger B and goto label2 or failure
  elseif(g = n):
    select smaller B and goto label2 or failure
```

Note that *M* is the *Least Common Multiple* of all numbers $\leq B$. Below follows a non-standard but efficient version of *p-1*:

```
P-1Factor(N, B, A):
  For(J=2, J ≤ B, J++):
    A = A^J mod N
  G = gcd(A − 1, N)
  If (1 < G < N):
    Return G
  Else:
    Return Failure
```

How to choose B: Assume p-1, where p is the smallest prime factor of *n*, can be modelled as a random number $\leq \sqrt{n}$. By *Dixon's theorem* we know that the probability that the largest factor of such a number is $\leq (p-1)^{\varepsilon}$ is roughly $\varepsilon^{-\varepsilon}$ so there is a probability of $3^{-3} = \frac{1}{27}$ that $B = n^{\frac{1}{2*3}} = n^{\frac{1}{6}}$ will yield factorization and a probability of $4^{-4} = \frac{1}{256}$ that $B = n^{\frac{1}{2*4}} = n^{\frac{1}{8}}$ will yield factorization.

3.3 General-Purpose Algorithms

General-purpose algorithms such as the quadratic and numberfield sieves, Williams' p+1 algorithm and the ECM algorithm are outside the scope of these notes.

4 Primality Testing

A *primality test* is an algorithm for checking if an input number is prime. Some tests prove that a number is prime while others prove it is composite, hence the latter might be more correctly referred to as *compositeness tests*.

4.1 Fermat Primality Test

The *Fermat primality test* is a probabilistic compositeness test to determine if a number is *probable pime*, that is, if it satisfies a condition satisfied by all prime numbers but not by most composite numbers.

Fermat's little theorem states that if p is prime and $1 \leq a < p$ then $a^{p-1} \equiv 1 \; mod \; p$. If we want to test if p is prime we can pick random a's in the interval and see if the equality holds. If it doesn't for some value of a then p is composite, if it does hold for all chosen a's then p is probable prime. Below is the pseudocode (where n is the integer to be tested and k the accuracy):

```
FermatTest(N, K):
  For(I=1, I ≤ K, I++):
    A = rand(1, N-1)
    If (A^(N-1) ≢ 1 mod N):
      Return Composite
  Return ProbablePrime
```

4.2 Miller-Rabin Primality Test

The *Miller-Rabin primality test* is a probabilistic compositeness test to determine if a number is *probable prime* in similar fashion to the Fermat test.

Square roots of unity (that is, those numbers that yield 1 when squared) in the finite field $\mathbb{Z}/p\mathbb{Z}$ where p is prime and $p > 2$ are always *trivial* (ie. 1 and -1, which always yield 1 when squared modulo p), there are no *non-trivial* square roots of 1 modulo p. This can be shown as follows, suppose x is a square root of 1 modulo p then:

$$x^2 \equiv 1 \; mod \; p$$
$$(x-1)(x+1) \equiv 0 \; mod \; p$$

Hence x is congruent to either 1 or -1 modulo p. Now given a prime n where $n > 2$ it follows that n-1 is even and can be written as $2^s * d$ where s and d are positive integers and d is odd.For each a in the field $\mathbb{Z}/p\mathbb{Z}$ either:

$$a^d \equiv 1 \; mod \; n$$
or
$$a^{2r*d} \equiv -1 \; mod \; n$$

For some $0 \leq r \leq s-1$. By *Fermat's little theorem* we know that either of the above holds since $a^{n-1} \equiv 1 \; mod \; n$. Hence if we keep taking square roots of a^{n-1} (where *n* is the integer to test) we will either get 1 or -1. If we get -1 the second equality holds and we are done. If we never get -1 then we have eliminated all powers of 2 and are left with the first equality. Thus the *Miller-Rabin primality test* shows that if we can find an *a* such that:

$$a^d \not\equiv 1 \; mod \; n$$
and
$$a^{2r*d} \not\equiv -1 \; mod \; n$$

For all $0 \leq r \leq s-1$ then *n* is composite. Below is the pseudocode (where *n* is the integer to be tested and *k* the accuracy):

```
MillerRabinTest(N, K): //N>3 and N is odd
  //write N-1 as 2^s * d
  S = 0, D = N-1
  While(D%2=0):
    D = D div 2
    S++

  //Witness testing
  For(I=1, I ≤ K, I++): //WitnessLoop
    A = rand(2, N-2)
    X = A^D mod N
    If (X = 1) or (X = N - 1):
      Next(WitnessLoop)

    For(J=1, J ≤ S - 1, J++):
      X = X^2 mod N
      If (X = 1):
        Return Composite
      If (X = N - 1):
        Next(WitnessLoop)
    Return Composite
  Return ProbablePrime
```

5 Groups & Fields

5.1 Groups

A *group* is a set of elements G with an operation $\circ$ combining two elements of G. A group has the following properties:

- $\circ$ is closed, ie. $\forall a, b \in G: a \circ b = c \in G$
- $\circ$ is associative, ie. $\forall a, b, c \in G: a \circ (b \circ c) = (a \circ b) \circ c$
- Neutral/Identity element: $\exists 1: 1 \in G: [\forall a \in G: a \circ 1 = 1 \circ a = a]$
- Inverse element: $\forall a \in G: [\exists a^{-1}: a \circ a^{-1} = a^{-1} \circ a = 1]$

A special type of group is the *abelian* group:

- *Abelian (commutative)* iff: $\forall a, b \in G: a \circ b = b \circ a$

A Group supports an operation and its inverse operation (eg. if $\circ$ is addition the inverse is subtraction).

Element order: The order *ord(a)* of an element *a* of a group $(G,\circ)$ is the smallest positive integer k such that $a^k = a \circ a \circ .. k\ times .. \circ a = 1$ where 1 is the identity element of G.

5.1.1 Subgroups

Given a group (G, $\circ$), H $\subseteq$ G is a subgroup of G if H also forms a group under $\circ$.

5.1.2 Cyclic groups

A *cyclic group* G is generated by a single *generator* element g. That is, it consists of a set of elements such that $\exists g \in G: G = \{g^n | \ n\ is\ an\ integer\}$. Since any group generated by an element in a group is a subgroup of that group showing that the only subgroup of a group G that contains g is G itself is sufficient to show that it is cyclic. Every cyclic group is abelian.

An element g is a generator of the group $\mathbb{Z}_n$if g and n are coprime, thus there are $\varphi(n)$ different generators for $\mathbb{Z}_n$. Furthermore, a group G which contains an element a with maximum order $ord(a) = |G|$ is said to be cyclic, with elements with maximum order being called *primitive elements* or *generators*.

Example: If $G = \{g^0, g^1, .., g^5\}$ then $g^0 = g^6$ and G is cyclic.

Cyclic subgroup theorem: Given a cyclic group $(G,\circ)$ every element $a \in G$ with $ord(a) = s$ is the primitive element of a cyclic subgroup with *s* elements.

Number of subgroups: Given a finite cyclic group G of order *n* with *a* as a generator of G, then for every integer *k* that divides *n* there exists exactly one cyclic subgroup H of G with order *k*. This subgroup is generated by $a^{\frac{n}{k}}$. H consists exactly of the elements $a \in G$ which satisfy the condition $a^k = 1$. There are no other subgroups.

Finding cyclic subgroups: Finding cyclic subgroups of a finite cyclic group G can be done by considering each element of G as a generator and generating the corresponding subgroup. For example in $\mathbb{Z}_{12}^{+}$ we would take each element and generate the group by applying repeated addition modulo 12. So for *g=3* we would start at 0 and add 3 modulo 12 to obtain the subgroup $\{0,3,6,9\}$.

5.1.3 Finite groups

A *finite group* G is a group with a finite number, called the group's *order*, of elements.

Lagrange's theorem: For any finite group G, the order of every subgroup H divides the order of G.

Generalization of Fermat's Little Theorem: For any finite group G and $a \in G$ it holds that $a^{|G|} = 1$.

Multiplicative groups of prime fields: For every prime *p*, $(\mathbb{Z}_p^*, \times)$ is an abelian finite cyclic group.

On finite cyclic groups: Given a finite cyclic group G it holds that:
- The number of primitive elements of G is $\varphi(|G|)$
- If $|G|$ is prime then all elements $a \neq 1 \in G$ are primitive

5.1.4 Semigroups

A semigroup is a set S together with an associative binary operation *. A semigroup with identity is called a *monoid*.

5.2 Rings

A *ring* is a set R with binary operators + and * satisfying the following properties:

- R is an *abelian group* under addition, meaning:
  - The + operator is associative and commutative
  - R is closed under addition
  - There is an additive identity element 0
  - There is an additive inverse for every element

- R is a *monoid* under multiplication, meaning:
  - The * operator is associative
  - There is a multiplicative identity element 1

- Multiplication distributes over addition

5.2.1 Subrings

A subring S is a subset of a ring R that is itself a ring when the operators + and * are restricted to the subset and which contains the multiplicative identity of R. More formally, a subring $(S, +, *, 0, 1)$ of ring $(R, +, *, 0, 1)$ has $S \subseteq R$ while preserving structure and is both a subgroup of (R,+,0) and a submonoid of (R,*,1).

5.2.2 Semirings

A semiring is a set R with binary operations + and * such that (R,+) is a commutative monoid with identity element 0, (R,*) is a monoid with identity element 1, multiplication right- and left-distributes over addition and 0 is the absorption element for multiplication.

5.3 Fields

A *field* is a set F with an additive and a multiplicative group, thus supporting all four basic arithmetic operations (ie. addition, subtraction, multiplication and division). A *field* has the following properties:

- All elements of F form an additive, abelian group with group operation "+" and neutral element 0.
- All elements of F except 0 form a multiplicative, abelian group with group operation "*" and neutral element 1.
- Mixing group operations holds under distributivity, ie. $\forall a, b, c \in F: a(b + c) = (ab) + (ac)$

In other words F is a *commutative ring with unity* in which each nonzero element is invertible.

5.3.1 Subfields

If (K, +, *) and (L, +, *) are fields and $K \subseteq L$ then K is a *subfield* of L and:
- We can add elements of L to and multiply them with elements of K
- L is a vectorspace over K (other properties work because of the distributive laws)

5.3.2 Extension degree

Given fields K and L where $K \subseteq L$ the extension degree [L: K] is defined as $dim_K L$, the dimension of L as a K-vectorspace.

5.3.3 Finite/Galois Fields

A *finite field* or *Galois field* is a field with a finite number of elements. The number of elements in the field is called its *order* or *cardinality* denoted as |F|.

Cardinality/Order: The *order* or *cardinality* |F| of a finite field F is its number of elements. A field with order |F| only exists if |F| is a prime power, ie. $|F| = p^n$ for some positive integer n and prime integer p, where p is called the *characteristic* of the finite field.

For example: $\left|\mathbb{F}_q\right| = q,\ \left|\mathbb{F}_q^*\right| = \left|\mathbb{F}_q - \{0\}\right| = q - 1$

Characteristic: The characteristic char(F) of a field F is the smallest positive integer p such that $e_* + e_* + .. + e_* = e_+$, where e_* is the neutral multiplication element and $e_+$the neutral addition element (eg. 1+1+..+1=0). If no such p exists char(F) = 0. The characteristic of a field is either 0 or prime.

5.3.4 Prime Fields

Given a field K the smallest subfield contained in K is called the *prime field* of K. A *prime field* is a finite field of prime order, ie. n = 1. Elements of prime field GF(p) can be represented by integers $\{0,1,..,p1\}$ with modular integer addition and multiplication as its two operations.

Integer rings as finite fields: The integer ring $\mathbb{Z}_p$, where p is prime, is denoted as prime field GF(p) with a prime number of elements. All nonzero elements of GF(p) have an inverse and arithmetic is done modulo p.

GF(2): Smallest finite field that exists where addition is equivalent to XOR and multiplication equivalent to AND. See page 94 in Paar/Pelzl for truth tables.

5.3.5 Semifields

A semifield is a semiring in which all elements have a multiplicative inverse.

6 Discrete Logarithm Problem

The *Discrete Logarithm Problem* is the problem of determining the integer k given the equation: $b^k = g$ where b and g are elements of a finite group. Here the integer k is the *discrete logarithm* (the finite-group-theoretic analogue of ordinary logarithms) of g to the base b, ie.: $k = \log_b g$. Depending on b and g there can be no or even multiple discrete logarithms.

The general difficulty of solving this problem underlies several public-key cryptosystems.

6.1 DLP in Prime Fields

Given is the finite cyclic group $\mathbb{Z}_p^*$ of order *p-1*, where p is a prime, and a primitive element $a \in \mathbb{Z}_p^*$ and another element $b \in \mathbb{Z}_p^*$. The DLP is the problem of determining the integer $1 \leq k \leq p-1$ such that:

$$a^x \equiv b \bmod p$$

Such an integer must exist since a is a primitive element and each group element in a cyclic group can be expressed as a power of any primitive element.

It is often desirable to have a DLP in groups with prime cardinality to prevent the Pohlig-Hellman attack. Since $\mathbb{Z}_p^*$ has an order that is obviously not prime one often uses DLPs in subgroups of $\mathbb{Z}_p^*$ with prime order rather than in $\mathbb{Z}_p^*$ itself.

6.2 Generalized DLP

The DLP isn't restricted to $\mathbb{Z}_p^*$ but can be defined over any cyclic group. This is called the generalized DLP and can be stated as follows: Given a finite cyclic group G with group operation $\circ$ and cardinality n, we consider a primitive element $a \in G$ and another element $b \in G$. The DLP is finding the integer $1 \leq x \leq n$ such that:

$$b = a \circ a \circ .. x\ times .. \circ a = a^x$$

Such an integer must exist since a is a primitive element and hence each element of the cyclic group G can be generated by repeated application of the group operation on a.

There are cyclic groups in which the DLP is *not* difficult and hence are unsuitable for cryptographic purposes.

6.3 Generic attacks on the DLP

Generic DL algorithms are methods which only use the group operation and no other algebraic structure of the considered group. Since they do not exploit any special group properties they work in any cyclic group. There are two classes of generic attacks on the DLP:

- Running time depends on the size of the cyclic group: *Brute-Force*, *Baby-step Giant-Step*, *Pollard's ρ for DLP*.

- Running time depends on the size of the prime factors of the group order: *Pohlig-Hellman*

6.3.1 Attacks on weak cyclic groups

The DLP is not difficult in *all* cyclic groups and as such some are unsuitable for cryptographic purposes.

Example: Consider the additive group of integers modulo a prime, eg.: the finite cyclic group $G = (\mathbb{Z}_{11}, +)$ with generator $a = 2$. We try to solve the DLP for $b = 3$, ie. we have to find the integer $1 \leq x \leq 11$ such that:

$$x * 2 = 2 + 2 + .. x\ times .. + 2 \equiv 3\ mod\ 11$$

It is important to notice here that the DLP is defined in terms of the group operation, in this case addition. An 'attack' against this DLP works by expressing the additive relationship between *a*, *b* and *x* in terms of multiplication:

$$x * 2 \equiv 3\ mod\ 11$$

Hence, solving for *x* is simply a matter of inverting the generator *a*:

$$x \equiv 2^{-1} * 3\ mod\ 11$$

Using the EGCD to compute $2^{-1} = 6\ mod\ 11$ this gives us $x \equiv 6 * 3 \equiv 7\ mod\ 11$.
The core problem here is that we have mathematical operations (multiplication and inversion) which are not in the additive group making the generalized DLP over *G* easy.

6.3.2 Brute-Force attack

A brute force attack is the most naïve and computationally costly way for computing the DL $\log_a b$. We simply compute powers of the generator *a* successively until the result equals *b*, ie.:

$$a^1 \not\equiv b?$$
$$a^2 \not\equiv b?$$
$$\ldots$$
$$a^x \not\equiv b?$$

For a random logarithm *x* we expect to find the correct solution after checking half of all possible *x* giving a complexity of $\mathcal{O}(|G|)$ steps.

Prevention: To avoid brute-force attacks the cardinality |G| of the underlying group must be sufficiently large. For instance, in $\mathbb{Z}_p^*$ (the basis for DHKE) $\frac{p-1}{2}$ tests are required on average to compute a DL. Hence |G| = *p-1* should be at least in the order of 2^{80} to make a brute-force search infeasible using today's computer technology.

6.3.3 Shanks' Baby-Step Giant-Step attack

Shanks' algorithm is a time-memory tradeoff method which reduces the time of a brute-force attack at the cost of extra storage. The idea is based on rewriting the DL $x = \log_a b$ in a two-digit representation:

$$x = x_g m + x_b \text{ for } 0 \leq x_g, x_b < m$$

Where *m* is chosen to be of the size of the square root of the group order, ie. $m = ceil\left(\sqrt{|G|}\right)$. We can now rewrite the DL as $b = a^x = a^{x_g m + x_b}$ which gives us:

$$b * (a^{-m})^{x_g} = a^{x_b}$$

BSGS seeks to find a solution (x_g, x_b) to the above equation from which the DL follows according to the first equation. We can find this solution by searching for x_g and x_b separately, ie. by using a divide-and-conquer approach. First we compute all a^{x_b} where $0 \leq x_b < m$ (the Baby-step phase that takes $m \approx \sqrt{|G|}$ steps and needs to store equally many elements). Then the algorithm checks (in the Giant-step phase) for all x_g with $0 \leq x_g < m$ whether the following holds:

$$b * (a^{-m})^{x_g} =? a^{x_b}$$

For some precomputed a^{x_b}. If so the DL is given by $x = x_g m + x_b$. The *BSGS* algorithm takes $\mathcal{O}(\sqrt{|G|})$ steps in total and an equal amount of memory. This means that in a group of order 2^{80} and attacker would only need 2^{40} operations and memory. Hence groups must have order at least $|G| \geq 2^{160}$. For example, in $\mathbb{Z}_p^*$ the prime *p* should have length at least 160 bit to prevent effective *BSGS* attacks.

Notes: It is not necessary to know |G| in advance, using an upper bound on |G| is sufficient. In addition, *BSGS* is usually used for groups of prime order. For groups of composite order *Pohlig-Hellman* is usually more efficient.

```
BSGS(G, A, B): //cyc.group G, gen. A, target B
  N = |G|
  M = ceil(√N)
  For(J=0, J<M, J++):
    Store (J, A^J mod P) in lookup table

  L = A^-M mod P
  Y = B
  For (I=0, I<M, I++):
    If(Y is A^J in any lookup table entry):
      Return I*M + J
    Else:
      Y = Y*L mod P
```

6.3.4 Pollard's ρ for DLP

Pollard's ρ for DLP is an algorithm for solving the DLP analogous to *Pollard's ρ for integer factorization*. The basic idea is to pseudorandomly generate group elements of the form $a^i * b^j$. For every element we keep track of the values *i* and *j*. We continue until we obtain a collision of two elements, ie. until we have:

$$a^{i_1} * b^{j_1} = a^{i_2} * b^{j_2}$$

If we substitute $b = a^x$ and compare exponents on both sides of the equation this yields the relation:

$$i_1 + x * j_1 \equiv i_2 + x * j_2 \; mod \; |G|$$

(Since we are in a cyclic group of order |G| we have to take the exponent modulo |G|).
From here we can compute the DL as follows:

$$x \equiv \frac{i_2 - i_1}{j_1 - j_2} \; mod \; |G|$$

In order to find the collision described above we will use *Floyd's Cycle-finding Algorithm* (as is the case with *ρ integer factorization*) to find a cycle in the sequence $x_i = a^{A_i} * b^{B_i}$ where the step function (eg. $f: x_i \rightarrow x_{i+1}$) are pseudorandom.

One way to define such functions *f, g, h* (where *f* is the step function for x, g for A_i and h for B_i) is to use the following rules: Divide G into three disjoint sets of approximately equal size s_0, s_1 *and* s_2 (where *p* is the group order and N is *p+1*):

$$f(x) = \begin{cases} bx \; mod \; N & if \; x \in s_0 \\ x^2 \; mod \; N & if \; x \in s_1 \\ ax \; mod \; N & if \; x \in s_2 \end{cases}$$

$$g(x,n) = \begin{cases} n & if \; x \in s_0 \\ 2n \; mod \; p & if \; x \in s_1 \\ n+1 \; mod \; p & if \; x \in s_2 \end{cases}$$

$$h(x,n) = \begin{cases} n+1 \; mod \; p & if \; x \in s_0 \\ 2n \; mod \; p & if \; x \in s_1 \\ n & if \; x \in s_2 \end{cases}$$

This gives us the following pseudo-code:

```
PollardRhoDLP(a, b, N)://a=gen.,b∈ G,N is order
  A_0 = 0, B_0 = 0, X_0 = 1 ∈ G, i = 1
  For(i=1, i<N, i++)
    X_i = f(X_{i-1}), A_i = g(X_{i-1}, A_{i-1}), B_i = h(X_{i-1}, B_{i-1})
    X_2i = f(f(X_{2i-2})), A_2i = g(f(X_{2i-2}), g(X_{2i-2}, A_{2i-2}))
    B_2i = h(f(X_{2i-2}), h(X_{2i-2}, B_{2i-2}))
    If(X_i = X_2i):
      R=B_i − B_2i
      If(R=0):
        Return Failure
      X=R^{-1}*(A_2i − A_i)mod p
      Return X
```

Pollard's ρ for DLP takes $\mathcal{O}(\sqrt{|G|})$ steps like *BSGS* but uses only negligible space and is the currently best known algorithm for computing the DL in elliptic curve groups. As with *BSGS* we need at least $|G| \geq 2^{160}$.

6.3.5 Pohlig-Hellman attack

The *Pohlig-Hellman attack* is based on the *Chinese Remainder Theorem* and exploits a possible factorization of the order of a group. It is usually not used by itself but in conjunction with other DLP attack algorithms. The *Pohlig-Hellman attack* targets groups whose order is a *smooth integer*.

Let $|G| = p_1^{e_1} * p_2^{e_2} * \dots p_L^{e_L}$ be the primepower factorization of the group order |G|. Again we seek to compute $\mathrm{z} = \log_a b$ using a divide-and-conquer approach. The basic idea is that rather than dealing with the large group G we compute discrete logarithms $x_i \equiv \mathrm{z} \bmod p_i^{e_i}$ in the subgroups of order $p_i^{e_i}$. The desired DL can then be computed from all x_i $(i = 1, \dots, L)$ by using the CRT. Each individual x_i can be computed using *Pollard's ρ for DLP* or the *BSGS* attack.

In general, though, we can compute $x_i \equiv \mathrm{z} \bmod p_i^{e_i}$ for some i as follows (let $p_i = q$ and $e_i = c$ for simplification). First we express *x* as:

$$x_i = \sum_{j=0}^{c-1} z_j * q^j$$

Where $0 \leq z_j \leq q - 1$. From this follows: $z = z_0 + z_1 * q + .. + z_{c-1} * q^{c-1} + s * q^c$ for some integer *s*. We compute z_0 from:

$$b^{\frac{p-1}{q}} \equiv a^{\frac{z_0*(p-1)}{q}} \; mod \; p.$$

Next we can compute $z_1, z_2, .., z_{c-1}$ from the generalization:

$$b_j = b_a{}^{-(z_0+z_1*q+..+z_{j-1}*q^{j-1})} \; mod \; p$$

We can make a generalization of the first equation here:

$$b_j^{\frac{p-1}{q^{j+1}}} \equiv a^{\frac{z_j*(p-1)}{q}} \; mod \; p$$

Hence given b_j we can compute z_j easily. We thus have the following recurrence relation:

$$b_{j+1} = b_j * a^{-a_j*q^j} mod \; p$$

This allows us to compute *z* using the earlier equation by computing the variables using the above recurrence and generalized equations alternatingly.

The runtime clearly depends on the prime factors of the group order and as such to prevent the attack the group order must have its largest prime factor in the range of 2^{160}. Of course, an

attacker needs to know the prime factorization of the group order for the attack to be successful and especially in the case of elliptic curve cryptosystems computing the order of the cyclic group is not always easy.

Example: Consider $\mathbb{Z}_p^*$, $p = 8101$. A generator of $\mathbb{Z}_p^*$ is $a = 6$. We have to find *x* so that $a^x = 7531 \; mod \; 8101$.

$\left|\mathbb{Z}_p^*\right| = p - 1 = 8100$ is a *smooth* integer that factors into $\{(2^2)(3^4)(5^2)\}$. Hence we shall determine the numbers:

$$x_2 \equiv \text{x mod } 2^2$$
$$x_3 \equiv \text{x mod } 3^4$$
$$x_5 \equiv \text{x mod } 5^2$$

Determination of x_2:
Since x_2 is a number mod 2^2 we have $x_2 = c_0 + 2c_1$ with c_i being either 0 or 1.

First we precompute the numbers (for 0) $a^{\frac{0*(p-1)}{2}} = 6^{\frac{0*(8101-1)}{2}} = 6^0 = 1$ and (for 1) $a^{\frac{1*(p-1)}{2}} = 6^{\frac{1*(8101-1)}{2}} = 6^{4050} \; mod \; 8101 = 8100$.

Compute c_0:
$\beta = 7531$. $\beta^{\frac{(p-1)}{2}} = 7531^{4050} \; mod \; 8101 = 8100$. Hence, $c_0 = 1$.

Compute c_1:
$\beta' = \frac{\beta}{a^{c_0}} = \frac{7531}{6} = 7531 * 6^{-1} = 7531 * 6751 \; mod \; 8101 = 8006$.
$\beta'^{\frac{(p-1)}{4}} = 8006^{2025} \; mod \; 8101 = 1$. Hence, $c_1 = 0$.

Combining into x_2:
This gives us $x_2 = c_0 + 2c_1 = 1 + 2 * 0 = 1$

Determination of x_3:
Since x_3 is a number mod 3^4 we have $x_3 = c_0 + 3c_1 + 9c_2 + 27c_3$ with c_i being either 0, 1 or 2.

First we precompute the numbers (for 0) $a^{\frac{0*(p-1)}{3}} = 6^{\frac{0*(8101-1)}{3}} = 6^0 = 1$ and (for 1) $a^{\frac{1*(p-1)}{3}} = 6^{\frac{1*(8101-1)}{3}} = 6^{2700} \; mod \; 8101 = 5883$ and (for 2) $a^{\frac{2*(p-1)}{3}} = 6^{\frac{2*(8101-1)}{3}} = 6^{5400} \; mod \; 8101 = 2217$.

Compute c_0:
$\beta = 7531$. $\beta^{\frac{(p-1)}{3}} = 7531^{2700} \; mod \; 8101 = 2217$. Hence, $c_0 = 2$.

Compute c_1:
$\beta' = \frac{\beta}{a^{c_0}} = \frac{7531}{6^2} = 7531 * (6^{-1})^2 = 7531 * 7876 \; mod \; 8101 = 6735$.
$\beta'^{\frac{(p-1)}{3*3}} = 6735^{900} \; mod \; 8101 = 1$. Hence, $c_1 = 0$.

Compute c_2:
$\beta'' = \frac{\beta'}{a^{3*c_1}} = \frac{6735}{6^0} = 6735 * (1^{-1}) = 6735.$
$\beta''^{\frac{(p-1)}{3*3*3}} = 6735^{300}\ mod\ 8101 = 2217$. Hence, $c_1 = 2$.
Compute c_3:
$\beta''' = \frac{\beta''}{a^{9*c_2}} = \frac{6735}{6^{18}} = 6735 * (6^{-1})^{18} = 6992.$
$\beta'''^{\frac{(p-1)}{3*3*3*3}} = 6992^{100}\ mod\ 8101 = 5883$. Hence, $c_3 = 1$.

Combining into x_3:
This gives us $x_3 = c_0 + 3c_1 + 9c_2 + 27c_3 = 2 + 3 * 0 + 9 * 2 + 27 * 1 = 47$

Determination of x_5:
Since x_5 is a number mod 5^2 we have $x_5 = c_0 + 5c_1$ with c_i being either 0, 1, 2, 3 or 4.

First we precompute the numbers $a^{\frac{i*(p-1)}{5}}$ for $0 \leq i \leq 4$ which gives us (in-order): $\{1, 3547, 356, 7077, 5221\}$.

Compute c_0:
$\beta = 7531.\ \beta^{\frac{(p-1)}{5}} = 7531^{1620}\ mod\ 8101 = 5221$. Hence, $c_0 = 4$.

Compute c_1:
$\beta' = \frac{\beta}{a^{c_0}} = \frac{7531}{6^4} = 7531 * (6^{-1})^4 = 7531 * 2019\ mod\ 8101 = 7613.$
$\beta'^{\frac{(p-1)}{5*5}} = 7613^{324}\ mod\ 8101 = 356$. Hence, $c_1 = 2$.

Combining into x_5:
This gives us $x_5 = c_0 + 5c_1 = 4 + 5 * 2 = 14$

Determination of X:
We determine *x* by combining x_2, x_3 and x_5 using the *Chinese Remainder Theorem* as follows:

$$x \equiv 1\ mod\ 2^2$$
$$x \equiv 47\ mod\ 3^4$$
$$x \equiv 14\ mod\ 5^2$$

Which can be rewritten as:

$$x = 1 + 4a$$
$$x = 47 + 81b$$
$$x = 14 + 25c$$

We first substitute the first into the second equation:

$$1 + 4a \equiv 47\ mod\ 81$$
$$4a \equiv 46\ mod\ 81$$
$$a \equiv 46 * 4^{-1}\ mod\ 81$$
$$a \equiv 52\ mod\ 81$$
$$a \equiv 52 + 81b$$

Next we rewrite the first equation and substitute it into the third:

$$x = 1 + 4 * (52 + 81b) = 209 + 324b$$

$$209 + 324b \equiv 14 \; mod \; 25$$
{200 is a multiple of 25}
$$9 + 324b \equiv 14 \; mod \; 25$$
$$324b \equiv 5 \; mod \; 25$$
$$b \equiv 5 * 324^{-1} \; mod \; 25$$
$$b \equiv 20 \; mod \; 25$$
$$b \equiv 20 + \; 25c$$

Finally we get:

$$209 + 324b = 209 + 324(20 + 25c) = 209 + 6480 + 8100c = 6689 \; mod \; 8101$$

Which is our solution for $a^x = 7531 \; mod \; 8101$.

6.4 Non-generic attacks on the DLP

Non-generic attacks on the DLP exploit special properties of certain groups (and hence are not generally applicable). The most important non-generic attack on the DLP is the so-called *Index Calculus* attack.

6.4.1 Index Calculus Attack

The index calculus attack is a very efficient algorithm for computing DLs in the cyclic groups $\mathbb{Z}_p^*$ and $GF(2^m)^*$ which has sub exponential running time. The index calculus attack relies on the property of a significant fraction of elements of the targeted group G being expressible as products of elements of a small subset of G (called the *factor base*), eg. for $\mathbb{Z}_p^*$ this means many elements should be expressible as a product of small primes. Both $\mathbb{Z}_p^*$ and $GF(2^m)^*$ satisfy this requirement.

First, a factor base is chosen, usually the first *r* primes (starting with 2) together with -1 where *r* is chosen to yield the optimal factor base. The choice of *r* is outside the scope of these notes.

The algorithm, targeting the DLP $g^x \equiv h \; mod \; n$, consists of four stages:

- Find r linearly independent relations between the factor base and power of the generator g. Each relation contributes one equation to a system of linear equations in *r* unknowns, namely the DLs of the *r* primes in the factor base.
- We solve the system of linear equations to compute the DLs of the factor base.
- We search for a power *s* of generator g which, when multiplied with argument h, may be factored in terms of the factor base $g^s h = (-1)^{f_0} 2^{f_1} 3^{f_2} .. p_r{}^{f_r}$.
- We rearrange the results of the 2nd and 3rd stages by algebraic manipulation to work out the DL $x = f_0 \log_g -1 + f_1 \log_g 2 + f_2 \log_g 3 + .. + f_r \log_g p_r - s$

Example: Given $p = 347, a = 37$ we want to find $L_a(91) = x$ (ie. the x where $a^x \equiv 91 \ mod \ p$.). Given is also the factor base $B = \{2,3,5,7\}$.

Initial step:
Compute $a^k \ mod \ p$ for values of k looking for numbers that can be expressed entirely within primes from our factor base.

$k = 1, a^k = 37$ which fails.
$k = 2, a^k = 2^3 * 41$ which fails.
$k = 3$ also fails.
$k = 4, a^k = 14 = 2 * 7$ which gives the relation $4 = L_a(2) + L_a(7)$.

We continue this process and eliminates duplicates and clear multiples to give us the relations:

$$4 = 1L_a(2) + 1L_a(7) \ mod \ p - 1$$
$$12 = 2L_a(3) + 1L_a(5) + 1L_a(7) \ mod \ p - 1$$
$$22 = 1L_a(3) + 2L_a(7) \ mod \ p - 1$$
$$29 = 3L_a(2) + 3L_a(3) \ mod \ p - 1$$

Intermediary step:
We then have the matrix equation to solve for the unknowns $L_a(2), L_a(3), L_a(5), L_a(7)$:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 2 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 3 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} L_a(2) \\ L_a(3) \\ L_a(5) \\ L_a(7) \end{bmatrix} = \begin{bmatrix} 4 \\ 12 \\ 22 \\ 29 \end{bmatrix} \ mod \ (p-1)$$

We can row reduce this in the usual way (without division):

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 9 \end{bmatrix} \begin{bmatrix} L_a(2) \\ L_a(3) \\ L_a(5) \\ L_a(7) \end{bmatrix} = \begin{bmatrix} 4 \\ 22 \\ 34 \\ 49 \end{bmatrix} \ mod \ (p-1)$$

Since gcd(9, 346) = 1 we can perform division using the multiplicative inverse:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} L_a(2) \\ L_a(3) \\ L_a(5) \\ L_a(7) \end{bmatrix} = \begin{bmatrix} 4 \\ 22 \\ 34 \\ 313 \end{bmatrix} \ mod \ (p-1)$$

Now simple backsubstitution to transform the first matrix to the identity matrix yields in order the values:

$$L_a(7) \equiv 313, L_a(5) \equiv 215, L_a(3) \equiv 88, L_a(2) \equiv 37 \text{ (all mod p-1)}$$

We have now computed the DLs of each prime in the factor base B.

Final step:
In order to compute the actual DL $L_a(91)$ we compute ba^k for $k = 1,2,..$ until we find a decomposition entirely in terms of primes in our factor base (or we get a definite failure and we have to add additional elements to B).
For $k = 1$ we get $91 * 37 \equiv 224 \equiv 2^2 * 61 \; mod \; 347$ which isn't fully in our factor base.
For $k = 2$ we get $91 * 37^3 \equiv 6 \equiv 2 * 3 \; mod \; 347$ which is in our factor base thus giving us the equation $L_a(91) = -k + L_a(p_1) + L_a(p_2) + .. + L_a(p_i)$ which evaluates to $L_{37}(91) = -2 + L_{37}(2) + L_{37}(3) = 123$.

7 Elliptic Curves

Elliptic Curve Cryptography (ECC) is based on the generalization of the DLP. Certain types of *curves* can be formed from polynomial equations (with the curve being the set of points (*x*,*y*) which are solutions to the equation). *Elliptic Curves* are special types of polynomial equations which for cryptographic purposes we will consider over a finite field, with prime fields GF(*p*) being the most popular choice.

The elliptic curve over $\mathbb{Z}_p, p > 3$ is the set of all pairs $(x, y) \in \mathbb{Z}_p$ which fulfill (in Weierstrass form):

$$y^2 \equiv x^3 + a * x + b \; mod \; p$$

Together with an imaginary point of infinity $\mathcal{O}$, where $a, b \in \mathbb{Z}_p$ and the following holds (ie. the discriminant is not 0): $4 * a^3 + 27 * b^2 \neq 0 \; mod \; p$.

Nonsingularity: Elliptic curves must be *nonsingular*. That is, they have no self-intersections or *cusps*, which is achieved if the discriminant of the curve $-16(4a^3 + 27b^2)$ is nonzero.

Characteristic: When we talk about the characteristic we talk about the characteristic char(K) of the field K over which the elliptic curve is defined. In the special cases of fields with characteristic 2 and 3 we have the following Weierstrass equations:

For char(3): $y^2 = 4x^3 + ax^2 + bx + c$
For char(2): $y^2 + axy + by = x^3 + cx^2 + dx + e$

Order: The order of a point on an elliptic curve is defined as the number of times scalar multiplication (ie. repeated application of point addition) has to be applied to the point in order to reach the *identity* (or *neutral*) point. Ie.: if $iP = P + .. + P = N$ (where $N = (0,1)$ in the case of Edwards curves and $N = \mathcal{O}$ for Weierstrass curves) then the order of P on this curve is *i*.

7.1 Curve forms

There are various representational forms of elliptic curves, some of which we will list below.

7.1.1 Edwards curve

The equation of an Edwards curve over a field K which does not have characteristic 2 is:

$$x^2 + y^2 = 1 + d * x^2 y^2$$

For some scalar $d \in K \setminus \{0,1\}$. The following form is also an Edwards curve:

$$x^2 + y^2 = c^2(1 + d * x^2 y^2)$$

Where $c, d \in K$ with $cd(1 - c^4 * d) \neq 0$. Often, Edwards curves are defined as having $c = 1$. Every Edwards curve is birationally equivalent to an elliptic curve in Weierstrass form.

7.1.1.1 Edwards Addition

Given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ their sum, resulting in a third point $P_3 = (x_3, y_3)$, is given as follows:

$$x_3 = \frac{x_1 y_2 + y_1 x_2}{1 + d * x_1 x_2 y_1 y_2}$$

$$y_3 = \frac{y_1 y_2 - x_1 x_2}{1 - d * x_1 x_2 y_1 y_2}$$

The *neutral element* here is (0,1).

7.1.1.2 Edwards Doubling

Given a point $P_1 = (x_1, y_1)$ its double $P_3 = (x_3, y_3) = 2P_1$ is given as follows:

$$x_3 = \frac{2x_1 y_1}{1 + d{x_1}^2 {y_1}^2}$$

$$y_3 = \frac{{y_1}^2 - {x_1}^2}{1 - d * {x_1}^2 {y_1}^2}$$

7.1.1.3 Conversion

<u>*To Twisted Edwards Form*</u>: Given a field K with characteristic different from 2 and E_d a curve in Edwards form:

$$E_d: x^2 + y^2 = 1 + d * x^2 y^2$$

Then it is equivalent to a curve $E_{a,d}$ in Twisted Edwards Form:

$$E_{a,d}: a * x^2 + y^2 = 1 + d * x^2 y^2, a = 1$$

7.1.2 Twisted Edwards curve

Twisted Edwards Curves are a generalization of Edwards Curves where each Twisted Edwards Curve is a *twist* of an Edwards Curve. A Twisted Edwards Curve $E_{E,a,d}$ over a field K which does not have $d = 0$ is an affine plane curve defined by:

$$E_{E,a,d}: a * x^2 + y^2 = 1 + dx^2y^2$$

Where $a, d \in K \setminus \{0\}$ and $a \neq d$. Every Edwards Curve is a Twisted Edwards Curve with $a = 1$. Every Twisted Edwards Curve is birationally equivalent with an elliptic curve in Montgomery form and vice versa.

7.1.2.1 Twisted Edwards Addition

Given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ their sum, resulting in a third point $P_3 = (x_3, y_3)$, is given as follows:

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + d * x_1x_2y_1y_2}$$

$$y_3 = \frac{y_1y_2 - a * x_1x_2}{1 - d * x_1x_2y_1y_2}$$

7.1.2.2 Twisted Edwards Doubling

Given a point $P_1 = (x_1, y_1)$ its double $P_3 = (x_3, y_3) = 2P_1$ is given as follows:

$$x_3 = \frac{2x_1y_1}{a * {x_1}^2 + {y_1}^2}$$

$$y_3 = \frac{{y_1}^2 - a * {x_1}^2}{2 - a * {x_1}^2 - {y_1}^2}$$

7.1.2.3 Conversion

<u>*To Montgomery Form*</u>: Given a field K with characteristic different from 2 and $E_{a,d}$ an elliptic curve in Twisted Edwards form:

$$E_{a,d}: ax^2 + y^2 = 1 + dx^2y^2$$

with $a, d \in K \setminus \{0\}$ and $a \neq d$. And $M_{A,B}$ an elliptic curve in Montgomery form:

$$M_{A,B}: Bv^2 = u^3 + Au^2 + u$$

with $A \in K \setminus \{-2,2\}$ and $B \in K \setminus \{0\}$.

Then there exists a map $\psi: E_{a,d} \rightarrow M_{A,B}$ (with corresponding inverse documented in section 7.1.3.3) converting a curve in Twisted Edwards form to a curve in Montgomery form:

$$(x, y) \to (u, v) = \left(\frac{1+y}{1-y}, \frac{1+y}{(1-y)x}\right), A = \frac{2(a+d)}{a-d}, B = \frac{4}{a-d}$$

To Weierstrass Form: Given a field K with characteristic different from 2 and $E_{a,d}$ and elliptic curve in Twisted Edwards form:

$$E_{a,d}: ax^2 + y^2 = 1 + dx^2y^2$$

with $a, d \in K \setminus \{0\}$ and $a \neq d$. And $W_{A,B}$ an elliptic curve in Weierstrass form:

$$W_{A,B}: v^2 = u^3 + \frac{A}{B}u^2 + \frac{1}{B^2}u$$

Then there exists a map $\psi: E_{a,d} \to W_{A,B}$ (with corresponding inverse documented in section 7.1.4.2) converting a curve in Twisted Edwards form to Weierstrass form:

$$(x, y) \to (u, v) = \left(\frac{1+y}{(1-y)B}, \frac{1+y}{(1-y)xB}\right), A = \frac{2(a+d)}{(a-d)}, B = \frac{4}{(a-d)}$$

7.1.3 Montgomery curve

A Montgomery Curve $M_{A,B}$ over a field K is defined by:

$$M_{A,B}: By^2 = x^3 + Ax^2 + x$$

Where $A, B \in K$ and $B(A^2 - 4) \neq 0$. Generally the curve is considered over a finite field K with characteristic different from 2 and with $A \in K \setminus \{-2,2\}$ and $B \in K \setminus \{0\}$.

7.1.3.1 Montgomery Addition

Given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ their sum, resulting in a third point $P_3 = (x_3, y_3)$, is given as follows:

$$x_3 = \frac{B(x_2y_1 - x_1y_2)^2}{x_1x_2(x_2 - x_1)^2}$$

$$y_3 = \frac{(2x_1 + x_2 + A)(y_2 - y_1)}{x_2 - x_1} - \frac{B(y_2 - y_1)^3}{(x_2 - x_1)^3} - y_1$$

7.1.3.2 Montgomery Doubling

Given a point $P_1 = (x_1, y_1)$ its double $P_3 = (x_3, y_3) = 2P_1$ is given as follows:

$$x_3 = \frac{({x_1}^2 - 1)^2}{4x_1({x_1}^2 + Ax_1 + 1)}$$

$$y_3 = \frac{(2x_1 + x_1 + A)(3{x_1}^2 + 2Ax_1 + 1)}{2By_1} - \frac{B(3{x_1}^2 + 2Ax_1 + 1)^3}{(2By_1)^3} - y_1$$

7.1.3.3 Conversion

To Twisted Edwards Form: Given a field K with characteristic different from 2 and $M_{A,B}$ and elliptic curve in Montgomery form:

$$M_{A,B}: Bv^2 = u^3 + Au^2 + u$$

with $A \in K \setminus \{-2,2\}$ and $B \in K \setminus \{0\}$. And $E_{a,d}$ an elliptic curve in Twisted Edwards form:

$$E_{a,d}: ax^2 + y^2 = 1 + dx^2y^2$$

with $a, d \in K \setminus \{0\}$ and $a \neq d$.

Then there exists a map $\psi: M_{A,B} \to E_{a,d}$ (with corresponding inverse documented in section 7.1.2.3) converting a curve in Montgomery form to Twisted Edwards form:

$$(u, v) \to (x, y) = \left(\frac{u}{v}, \frac{u-1}{u+1}\right), a = \frac{A+2}{B}, d = \frac{A-2}{B}$$

Note that this map is not defined at the points $v = 0, u + 1 = 0$.

To Weierstrass Form: Given an elliptic curve in Montgomery form:

$$M_{A,B}: Bv^2 = u^3 + Au^2 + u$$

Then there exists a map $\psi: M_{A,B} \to W$ and we can convert it to Weierstrass form by dividing each term by B^3 and mapping:

$$(u, v) \to (x, y) = (\frac{u}{B}, \frac{v}{B})$$

To get the Weierstrass equation:

$$y^2 = x^3 + \frac{A}{B}x^2 + \frac{1}{B^2}x$$

To obtain short Weierstrass form we can replace *x* with $t - \frac{A}{3B}$ which gives us:

$$y^2 = t^3 + \frac{(3 - A^2)}{3B^2}t + \frac{2A^3 - 9A}{27B^3}x$$

7.1.4 Weierstrass curve

See the Weierstrass curve as defined in the introduction to section 7.

7.1.4.1 Weierstrass Addition and Doubling

Given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ their sum, resulting in a third point $P_3 = (x_3, y_3)$, can be expressed compactly as follows:

$$x_3 = s^2 - x_1 - x_2 \; mod \; p$$
$$y_3 = s(x_1 - x_3) - y_1 \; mod \; p$$

Where:

$$s = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \; mod \; p \quad if \; P \neq Q \; (addition) \\ \dfrac{3{x_1}^2 + a}{2y_1} \; mod \; p \; \; if \; P = Q \; (doubling) \end{cases}$$

Note that *s* is the slope of the line through and P and Q in the case of addition or the slope of the tangent through P in the case of doubling.
In order to obtain an identity element we define an abstract *point at infinity* $\mathcal{O}$ as the identity element $\mathcal{O}$ such that $P + \mathcal{O} = P$. Its inverse, -P, is defined as $-P = (x_p, -y_p)$.

7.1.4.2 Conversion

To Twisted Edwards Form: Given a field K with characteristic different from 2 and $W_{A,B}$ and elliptic curve in Weierstrass form:

$$W_{A,B}: v^2 = u^3 + \frac{A}{B}u^2 + \frac{1}{B^2}u$$

And $E_{a,d}$ an elliptic curve in Twisted Edwards form:

$$E_{a,d}: ax^2 + y^2 = 1 + dx^2y^2$$

with $a, d \in K \setminus \{0\}$ and $a \neq d$.
Then there exists a map $\psi: W_{A,B} \rightarrow E_{a,d}$ (with corresponding inverse documented in section 7.1.2.3) converting a curve in Weierstrass form to Twisted Edwards form:

$$(u, v) \rightarrow (x, y) = \left(\frac{u}{v}, \frac{Bu - 1}{Bu + 1}\right), a = \frac{A + 2}{B}, d = \frac{A - 2}{B}$$

7.2 Finding points on a curve and confirming a point is on a curve

The naïve approach to counting all elements on an elliptic curve involves running through all elements of the field over which the curve is defined and checking whether they satisfy the Weierstrass equation of the curve. In pseudo-code:

```
CountPoints(E,F):
  Points = []
  ForEach((x,y) ∈ F):
    If(y² = x³ + Ax + B):
      Points.add((x,y))
  Return Points
```

Where E is the curve and F the field over which it is defined.

Example: Given the curve $E: y^2 = x^3 + x + 1$ over $\mathbb{F}_5$. To count the points on E we make a list of values of *x*, the corresponding values of y^2 and the square roots *y*:

x	$x^3 + x + 1 \bmod 5$	y	Points
0	1	± 1	(0,1), (0,4)
1	3	-	-
2	1	± 1	(2,1), (2,4)
3	1	± 1	(3,1), (3,4)
4	4	± 2	(4,2), (4,3)

Note that eg. ± 1 translates to 1 and its additive inverse which, in $\mathbb{F}_5$, is 4.

Confirming a point is on the curve: If we want to confirm if a point $P = (a, b)$ is on the curve $E: y^2 = x^3 + Ax + B$ over $\mathbb{F}_p$ or not we simply evaluate E using P, eg.: $b^2 = a^3 + Aa + B$ and check whether *a* and $\sqrt{b^2}$ are in $\mathbb{F}_p$.

7.3 Hasse's Theorem

Hasse's theorem states that the number of points on a curve is roughly in the range of the prime *p*. More formally, given an elliptic curve E modulo *p* the number of points on the curve is denoted by #E and bounded by: $p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}$.

The implication here is that if we need an elliptic curve with 2^{160} elements we have to use a prime of length about 160 bit.

7.4 The ECDLP and Point Multiplication

The Elliptic Curved Discrete Logarithm Problem (ECDLP) is the DLP within an EC context, that is, it is concerned with finding the DL of a random elliptic curve element with respect to a publicly known base point. More formally, given an elliptic curve E we consider a primitive element P and another element T. The ECDLP is finding the integer d, where $1 \leq d \leq \#E$, such that: $P + P + .. \, d \; times .. + P = dP = T$. This operation is called *point multiplication* since we can write $T = dP$. It is not a direct multiplication however but the repeated application of the group operation. In the ECDLP *d* is the private key (an integer) while the public key *T* is a point on the curve with coordinates $T = (x_T, y_T)$.

Example: Given the curve $y^2 \equiv x^3 + 2x + 2 \; mod \; 17$ we want to compute $13P = P + P + .. + P$ where $P = (5,1)$. We can use the following pseudo-code algorithm for *point multiplication* by means of *Double-and-Add*:

```
DoubleAndAddPM(E,P,d):
  T=P
  For(I=T-1,I ≥ 0,I--):
    T=T+T mod n
    If(d_i = 1):
      T=T+P mod n
  Return T
```

Here E is an elliptic curve, P a point on the curve and d a scalar in binary representation $d = \sum_{i=0}^{t} d_i \, 2^i$ with $d_i \in 0{,}1$ and $d_t = 1$. The algorithm scans the bit representation of the scalar d from left to right, performing doubling in every iteration and only if the current bit has the value 1 does it perform an addition of p.

8 RSA

The basics of the RSA cryptosystem are presumed to be known and will not be covered by this section.

8.1 Speeding up RSA decryption using the CRT

The RSA decryption process can be sped up using the Chinese Remainder Theorem. The following values are precomputed and stored as part of the private key:

- p and q
- $d_P \equiv d \; mod \; (p-1)$ and $d_q \equiv d \; mod \; (q-1)$
- $q_{inv} \equiv q^{-1} \; mod \; p$

These values allow the recipient to compute the modular exponentiation $m \equiv c^d \; mod \; pq$ more efficiently as follows:

$$m_1 \equiv c^{d_p} \; mod \; p$$
$$m_2 \equiv c^{d_q} \; mod \; q$$
$$h \equiv q_{inv}(m_1 - m_2) \; mod \; p$$
$$m = m_2 + hq$$

If $m_1 < m_2$ then some methods compute $h \equiv q_{inv}\left(\left(m_1 + ceil\left(\frac{q}{p}\right)p\right) - m_2\right) mod \; p$

The method is more efficient because while we compute two modular exponentiations they both use smaller exponents and moduli.

8.2 Schoolbook RSA insecurities

So-called *'schoolbook RSA'* has several weaknesses, some of which we'll discuss below.

8.2.1 Unpadded RSA

Schoolbook RSA is unpadded which leads to several problems:

- *'Insufficient wraparound'*: When small exponents (eg. *e=3*) and plaintexts (eg. $m < n^{\frac{1}{e}}$) are used the ciphertext result *c* of the modular exponentiation operation of

RSA encryption is strictly less than the modulus *n*, thus causing no '*wraparound*' and making decryption as trivial as taking $\sqrt[e]{c}$.

In addition, for plaintext values of 0, 1 or -1 the corresponding ciphertext is always 0, 1 or -1.

- *Determinism*: Since RSA is deterministic identical plaintexts map to identical ciphertexts for a given key, allowing attacker derivation of statistical properties of plaintexts as well as potential *chosen plaintext* attacks (by encrypting likely plaintexts under the key using an encryption oracle and checking if the result matches the target ciphertext).

- *Malleability*: Schoolbook RSA is malleable in the sense that an attacker can calculate $x^e * c \; mod \; n = \big(x^e * (p^e \; mod \; n)\big) mod \; n = (x^e * p^e) \; mod \; n$ thus resulting in a ciphertext corresponding to the plaintext $x * p$, allowing an attacker to arbitrarily arithmetically manipulate the plaintext (such as doubling it, etc.). If the encrypted message contains, for example, an amount of money to be transferred a MITM attacker could double the amount without having to know the key or plaintext.

- *Håstad/Coppersmith attack*: If the same plaintext message is sent in encrypted form to *e* or more recipients and the receivers share the same exponent *e* but have different *p* and *q* (and therefore *n*) then it is easy to decrypt the message using the CRT.

  A simple version of the attack is as follows. Suppose Alice sends the same message M to k different receivers $P_1, P_2, .., P_k$ each using the same exponent e (say $e = 3$) and different moduli N_i. When $k \geq e$ an attacker Eve can intercept $C_1, C_2, .., C_k$ where $C_i \equiv M^e \; mod \; N_i$. Next Eve can assume $\gcd\big(N_i, N_j\big) = 1$ for all i and j (since otherwise one could compute a factor of one of the N_i by calculating this gcd). Then, by the CRT, Eve can compute $C \in \mathbb{Z}^*_{N_1 N_2 .. N_k}$ such that $C_i \equiv C \; mod \; N_i$. Then $C \equiv M^e \; mod \; N_1 N_2 .. N_k$. However, since $M < N_i$ for all *i* we have $M^e < N_1 N_2 .. N_k$ and thus $C = M^e$ holds over the integers and eve can compute the *e*-th root of *C* to obtain *M*.

  Example: Consider $e = 3, k = 3, n_1 = 87, n_2 = 115, n_3 = 187$ and $M = 10$, this gives:

$$C_1 \equiv 43 \equiv M^3 \; mod \; 87$$
$$C_2 \equiv 80 \equiv M^3 \; mod \; 115$$
$$C_3 \equiv 65 \equiv M^3 \; mod \; 187$$

  We calculate $N = \; n_1 n_2 n_3 = 1870935$ and:

$$N_1 = \frac{N}{n_1} = 115 * 187 = 21505, d_1 = 21505^{-1}\ mod\ 87 = 49$$
$$N_2 = \frac{N}{n_2} = 87 * 187 = 16269, d_1 = 16269^{-1}\ mod\ 115 = 49$$
$$N_3 = \frac{N}{n_3} = 87 * 115 = 10005, d_1 = 10005^{-1}\ mod\ 187 = 2$$

$$x \equiv c_1 N_1 d_1 + c_2 N_2 d_2 + c_3 N_3 d_3\ mod\ N$$
$$x \equiv 43 * 21505 * 49 + 80 * 16269 * 49 + 65 * 10005 * 2\ mod\ 1870935$$
$$x \equiv 1000\ mod\ 1870935$$

We know *M* is the cube root of *x* and hence $M = \sqrt[3]{1000} = 10$.