KEY EXCHANGE ALGORITHM flowcharts **SumTwoPoints** TheFirstPoint Input value: x_i, y_i, x_j, y_j, p, a Input values: a, b, p Output value: x_k, y_k (stored in array) Output values: XFirst, YFirst, Order Subfunctions: Subfunctions: XPart, YPart LambdaSame, XSame, YSame LambdaDiff, XDiff, YDiff Recognizer This application decides (arrayA of x values) which method will be use call **Recognizer** for the sum of points. Inputs: x_i, y_i, x_j, y_j Outputs: value of decisior (arrayB of y values) ECDH PHASE A ECDH_PHASE_BB ECDH PHASE C ECDH_PHASE_BA call **LambdaSame** Aplication for compute the generator Set of several applications, whose task is to compute Autonomous aplication for select Aplication for select a secret key select a secret key from the common key and point (coordinates) suitable value for variables p, a and psudo-random number and compute from psudo-random number and b. Second purpose is compute a Input values: MySecKey, XFirst, YFirst, XObtain, YObtain, compute a point for sharing. a point for sharing. Preparation of iteration values generator and order. Variable Input values: Order, XFirst, YFirst, a Input values: *p*, *a*, *b* p, a, order "Throttle" represents max size of call **XSame** Output values: $x_k, y_k, SecKey$ Output values: $x_k, y_k, SecKey$ Y_iter = ((p - 1) / 2) + 1) input values for p, a, b. Output values: CommonKey, CommonPoint (coordinates) Output values: *p*, *a*, *b*, *XFirst*, *YFirst*, SecretKey TheFirstPoint PointComp SecretKey RecognizerSecKey call **YSame** PointComp Common_Key Subfunctions: PointComp return 66 *(refusal value)* primeTester, checkValAB Order = 0TheFirstPoint PointComp VerifyOfPoint $array[0] = X_k$ $array[1] = Y_k$ call **LambdaSame** call **SecretKey** call TheFirstPoint call RecognizerSecKey return 66 *(refusal value)* k = 0; k <= X_ite call **XSame** Throttle = 1000 call **CommonKey** Value alignment iterator = SecKey (copy value) call **SecretKey** iterator += 1 (increase value due the first position call **YSame** occupied by the first point) Value alignment Value alignment erator = MutualKey (copy value) return 66 (refusal value) Order *= 2 (two points for each x's Do While loop erator += 1 (increase value due the first position i = 0; i <= Y_iter Order += 1 (add zero point) occupied by the first point) Value alignment iterator = SecKey (copy value) iterator += 1 (increase value due the first position is call **PointComp** occupied by the first point) Do While loop call **CommonKey** Output **Order** value Do While loop Stored results in array call **PointComp** Common Point (coordinates stored in arra p = rand() % ThrottleStored results in call **primeTester** TESTmodulo == 2 Value of iteration primeTester iter = (p - 1) / 2Test a candidate Test if values \boldsymbol{a} and \boldsymbol{b} are on the curve number *n* for primality Input values: p, a, b Input values: p call **checkValAB** number = i^2 checkValAB Test of values **a** and **b** according to resY %= mod pthe formula (see below), that must not be equal to zero. nums [i] = number % pInput values: a, b, presult == 0 a = rand() % Throttle b = rand() % Throttleresult = 2 (refusal value) call **checkValAB** result = 2 *(refusal value)* Return 1 (approval value) call **TheFirstPoint PointComp** RecognizerSecKey Input value: XFirst, YFirst, p, a, iteration call **PointComp** Application for recognizing the secre Output value: $STOPval, x_k, y_k$ key from the other party. Based on **Subfunctions:** knowledge of generator of group (the LambdaSame, XSame, YSame fist point). The computed order of LambdaDiff, XDiff, YDiff computation represents the secret Recognizer key of the other party. call **VerifyOfPoint** result = 2 (refusal value) Input values: XFirst, YFirst, XObtain, YObtain, p, a Due to rule 2P = P + POutput values: ObtainSecKey Subfunctions: TheFirstPoint SecretKey PointComp result = 1 (approval value) stopVal == 66 Stored results in array $array[0] = X_k$ Due to rule 2P = P + $array[1] = y_k$ $x_j = x_i$ $y_j = y_i$ Stored results in **XPart** STOPval = 1 stopVal != 2 Select SecretKey from Inputs: p, a, b pseudo-random number Outputs: array of all xInput value: order call **LambdaSame** value == 1 Output value: SecKey Initial value of xreturn value of counter $\chi = 0$ call SumTwoPoints Common_Key rand() % (order - 2 Application for compute the common key from knowledge of our secret key and Refresing values x_i and y_j from SumTwoPoints recognized secret key from opposite party. call **YSame** LambdaSame $x_i = \text{array}[0]$ Application based on $y_i = array[1]$ Application based on Input values: OurSecKey, ObtainSecKey, order formula: Output values: CommonSecKey $\lambda = \frac{3x^2 + a}{2t} \mod p$ reparation for next round (e.g. 3P = 2P + I $\lambda = \frac{y - x_i}{x_j - x_i} \bmod p$ Subfunctions: STOPval = 1counter++ $comp = x^3 + ax + b$ $y_k = y_k$ result += 2 value == 2 nums[i] = comp % pApplication based on Application based on array[0] == XObtain formula: formula: CommonKey = MySecKey · ObtainSecKey $x_k = (\lambda^2 - 2x_i) \bmod p$ $x_k = (\lambda^2 - x_i - x_j) \mod p$ result += 3 CommonKey = CommonKey % Order

STOPval = 66

StopVal = 2

retrun value Common Key

Application based on

formula:

 $y_k = (\lambda \cdot (x_i - x_k) - y_i) \bmod p$

*Seckey = result

Application based on

formula:

 $y_k = (\lambda \cdot (x_i - x_k) - y_i) \mod p$