1 SMART CONTRACT

The parties considered in the design of this contract include the Employer, Company where bonus is being spent at and the individual exercising the contract. The individual can either be an employee or a beneficiary of the employee, otherwise termed as the partner. For reasons that will become clear, the general design settled upon is where the contract is activated at a verified point of spending followed by the completion of a transaction with the respective employer. One could say it is a Business to Business (B2B) model. In essence the smart contract would expect a trigger containing the individual exercising the contract, the amount to be spent and information that would be used to verify whether the company is a valid point to spend.

Bonus schemes have proven to be vulnerable to manipulation as employees, especially those in management result to altering accounting information to increase their bonus [2, 3]. As much as the scope of this paper doesn't account for fixing the bonus scheme in entirety, one may want to consider the application of blockchain to make such records that directly affect bonus calculations immutable [1]. In light of some of the views of making bonus scheme free of manipulation, at least for the most part, adoption of technologies is not just enough as it has to do with the collective view of the system [4]. Some of those could however be extended through solutions such as in the manner stipulated from here on.

1.1 Aims of the design include

1.1.1 . To encourage a healthy and autonomous spending of the bonus without overloading the employee with too much information to deal with especially in the case where the individual has partners tied to the benefit.

1.1.2 . In conjunction with the first aim, protect the privacy of the partners who may not want their benefactors knowing what they are spending on.

1.1.3 . Give the employer room to negotiate and renegotiate terms of agreement with the companies without directly involving the beneficiaries.

1.1.4 . Implementing security levels especially on privileged information between the employer and the various alluded parties.

1.1.5 . Prevent inter trading between employees as even the employees wouldn't know just how much their points are worth in actual sense.

1.2 Objective of the design include:

1.2.1 . Tracking points awarded to various members as well as the expenditure by various beneficiaries with respect to the various valid points of spending.

1.2.2 . Providing a method of confirming the various sets of information such as valid companies to spend points at, beneficiaries and what they have left

1.3 Limitations of the system include:

1.3.1 . Verification of valid companies to spend at is implemented just as a list of reference. A proper oracle ought be independent and perhaps include a lot more information than the employer would have time to follow up on. An example being if an organisation's practise goes against the employer's values.

Wellington Mugendi1, Artificial Intelligence, University of Southampton
Email: 1wmn1u18@soton.ac.uk

1.3.2 . Due to the underlying blockchain technology decided upon, all companies the employer would like to get into a contract with would have to join the employer's blockchain network in order for the beneficiaries to spend in those places.

1.3.3 . Should an employee receiving benefits also be a partner to another employee, the contract will not be in a position to detect their benefits unless they get benefits on a different account. They wouldn't be able to consolidate the two accounts during spending.

2 IMPLEMENTATION

2.1 Pseudo Code

The pseudo code components of the smart contract include the verification of valid companies, the verification of beneficiaries i.e. the employees and their respective partners and how many points they have left and updating of the expenditure once it's successful. On contract trigger, various sets of information i.e. employees, partners, valid companies and expenditures details are loaded for search and verification in later stages of the contract. All the functions mentioned from here on are available in the form of a python script in the code section 2.2.

Still at initialization, three arguments are received. The arguments are the company where points will be spent, the amount needed and address of the individual exercising the contract. The first function of the contract to be executed is dubbed as verifyCompany(). This function should in actual sense perform its features over an oracle. In the future that should be the case. For the sake if this exercise, the function loops through a list of verified companies and returns True if the company is found, otherwise False is returned.

Once the company is verified, a function dubbed verifyBeneficiary() is executed. This function only checks if the address provided belongs to an employee and not the partner. Should the employee exist, their balance is calculated by verifying how many points they were awarded, how many they allotted to partners if they had any and how much they have spent already. Note, the logic does not take to account if an employee could be a partner in the case of another employee. Should the difference be positive, it is returned to the point the function was called, otherwise a False flag is sent/returned.

Should the response of verifying a beneficiary be False, verifyPartner() is executed as the individual exercising the contract could be a partner and not an employee. Should this function as well return False, the contract terminates execution. Assuming the latter is true, the balance of the individual is received at the point of call. The difference between the verifyBeneficiary() function and this is that a partner could have multiple benefactors. In that regard, the amount allotted is a summation of what was allocated by the various employee(s).

Once an amount is received, regardless of whether it be from an employee or partner, the last function is called. This function is dubbed updateExpense(). Before expenses are updated, the amount to be spent is subtracted from the amount in the individual's account. If the balance is zero or positive then the expense is updated alongside the time of execution and address of the individual. In the latter case the contract terminates on False as the response.

The algorithm doesn't have descriptive response on an error other than just False as the response. As an improvement, they could endeavour to contain a more descriptive message for both debugging and readability at use. The many loops may not be necessary if oracles are set up to execute the search outside the contract's run-time.

Wellington Mugendi1, Artificial Intelligence, University of Southampton
Email: 1wmn1u18@soton.ac.uk

2.2 Contract in Python

```
import time

class bonusContract():

  def __init__(self):

    #load list of beneficiaries {"address":'hash', "employeeCode":03245, "amount":4000}
    self.beneficiaries = [{"address":'hash', "employeeCode":"03245", "amount":4000}]

    #load list of partners and associatons to beneficiary {"address":'hash', "benefactor":"
     employeeCode=03245", "portion":"20%"}
    self.partners = [{"address":'hash1', "benefactor":"03245", "portion":20}]

    #load list of approved companies fetch data from an oracle confirming validity of the company {"
     companyName":"Amazon", "categoty":"Shopping"}
    self.validCompanies = [{"companyName":"Amazon", "categoty":"Shopping"}]

    #load list of expenditures {"address":"hash", "amount":"2000", "timestamp":time.time(), "where":"
     company"}
    self.expenditure = [{"address":"hash", "amount":1000, "timestamp":time.time(), "where":"Amazon"},{
     "address":"hash1", "amount":100, "timestamp":time.time(), "where":"Amazon"}]

  def execContract(self, byWho, toCompany, payOutAmount):

    #is company valid
    if self.verifyCompany(toCompany) == True:

      # is beneficiary valid
      response = self.verifyBeneficiary(byWho)
      #print response, '<=benef'
      if response == False:

        #check if partner instead
        response = self.verifyPartner(byWho)
        #rint response, '<=partner'

        if response == False:

          # terminate process : beneficiary not valid
          return False

        else:

          #compute expense
          return self.updateExpense(byWho, payOutAmount, toCompany, response)

      else:

        #update expense
```

```
46          return self.updateExpense(byWho, payOutAmount, toCompany, response)
47
48      else:
49
50        #terminate process : Invalid company
51        return False
52
53    def verifyCompany(self, toCompany):
54
55      #search for approval
56      response = False
57
58      for each in self.validCompanies:
59
60        if toCompany in each['companyName']:
61
62          #update found
63          response = True
64          break
65
66      #express decision
67      return response
68
69    def verifyBeneficiary(self, byWho):
70
71      #default response
72      response = False
73
74      for stack in self.beneficiaries:
75
76        #check if in list
77        if byWho in stack['address']:
78
79          #get staff employeeCode
80          empCode = 0
81          amount = 0
82          for each in self.beneficiaries:
83
84            if each['address'] == byWho:
85
86              amount = each['amount']
87              empCode = each['employeeCode']
88
89          #check if beneficiary has partners
90          #get amount benefactor gave partners
91          partner = 0
92
93          for each in self.partners:
94
95            if each['benefactor'] == empCode:
96
```

```
            allot = each['portion']

            #compute amount
            partner = partner + (amount*(allot/100))

        #compute how much benefactor has spent

        expense = 0
        for each in self.expenditure:

          if byWho in each['address']:

            #get expenses
            expense = expense + each['amount']

        # how much is benefactor left with
        response = amount - partner - expense
        break

    #results
    return response

  def verifyPartner(self, byWho):

    #default setup for response
    response = False

    #search beneficiary
    for each in self.partners:

      #partner portion and benefactor
      if byWho in each['address']:

        #found, extract info and exit
        benefactor = each['benefactor']
        portion = each['portion']
        break

    if 'benefactor' in locals():

      #find expenses and alloted portion
      #alloted amount

      for each in self.beneficiaries:

        #filter by employeeCode
        if benefactor in each['employeeCode']:

          #get amount : there could be more than one benefactor dont break
          alloted = each['amount']*(portion/100)

```

```
148       #does alloted exist: find expenses
149       if 'alloted' in locals():
150
151         #expenses
152         totExp = 0
153         for each in self.expenditure:
154
155           #filter by address
156           if byWho in each['address']:
157
158             #sum expense
159             totExp = totExp + each['amount']
160
161         #update remainder
162         response = alloted - totExp
163
164     #reporting results
165     return response
166
167   def updateExpense(self, byWho, amount, toCompany, inStore):
168
169     #init respone
170     respone = False
171
172     #check if balance is positive
173     if (inStore - amount) > -1:
174
175       #update
176       self.expenditure.append({"address":byWho, "amount":amount, "timestamp":time.time(), "where":
      toCompany})
177       respone = True
178
179     #terminate
180     return respone
181
182 #testing phase
183 '''
184 cont = bonusContract()
185 print cont.execContract('hash1', 'Amazon', 500)
186 '''
```

Wellington Mugendi1, Artificial Intelligence, University of Southampton
Email: 1wmn1u18@soton.ac.uk

REFERENCES

[1] Androulaki, E., Bortnikov, V., Cachin, C., Caro, A. D., Enyeart, D., Muralidharan, S., Murthy, C., Smith, K., Sorniotti, A., Cocco, S. W., and Yellick, J. Hyperledger Fabric : A Distributed Operating System for Permissioned Blockchains.

[2] Healy, P. M. THE EFFECT OF BONUS SCHEMES ON ACCOUNTING DECISIONS* Paul M. HEALY. 85–107.

[3] Holthausen, R. W., Larcker, D. F., and Sloan, R. G. *Annual bonus schemes and the manipulation of earnings*, vol. 19. 1995.

[4] Turner, R. Innovation in fi eld force bonuses : Enhancing motivation through a structured process-based IMPORTANCE OF VARIABLE A SUCCESSFUL BONUS. 126–135.