Учреждение образования

«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра информатики

Тема отчёта:
«Реализация криптографических атак с помощью машинного обучения на физически неклонируемые функции»

Выполнил: Демидов Дмитрий Александрович
магистрант кафедры информатики
группа №858642

Проверил: магистр технических наук
Стержанов Максим Валерьевич

Минск 2019

Задача

При применении к ФНФ с однобитовыми выходами каждому вызову C назначается вероятность $p(C,t|\vec{w})$ такая, что он генерирует выходной сигнал $t\in\{-1,1\}$. Таким образом, вектор $\vec{w}$ кодирует соответствующие внутренние параметры ФНФ. Вероятность определяется логистической сигмоидой, действующей на функцию $f(\vec{w})$, параметризованную вектором $\vec{w}$ как $p(C,t|\vec{w})=\sigma(tf)=(1+e^{-tf})^{-1}$. Таким образом, f через $f=0$ определяет границу решения с равными выходными вероятностями.

Модель, которая могла бы предсказывать ответы по запросам, которых нет в обучающей выборке.

```
def into_features_vect(chall):
    "Transforms a challenge into a feature vector"
    phi = []
    for i in range(1,len(chall)):
        s = sum(chall[i:])
        if s % 2 == 0:
            phi.append(1)
        else:
            phi.append(-1)
    phi.append(1)
    return phi
```

(продолжение на следующей странице)

```
class Stage:
    _delay_out_a = 0.
    _delay_out_b = 0.
    _selector = 0

    def __init__(self,delay_a,delay_b):
        self._delay_out_a = delay_a
        self._delay_out_b = delay_b

    def set_selector(self,s):
        self._selector = s

    def get_output(self,delay_in_a, delay_in_b):
        if self._selector == 0:
            return (delay_in_a  + self._delay_out_a,
                    delay_in_b  + self._delay_out_b)

        return (delay_in_b  + self._delay_out_a,
                delay_in_a  + self._delay_out_b)

class ArbiterPUF:

    def __init__(self,n):
        self._stages = []

        for _ in range(n):
            d1 = random.random()
            d2 = random.random()
            self._stages.append(Stage(d1,d2))

    def get_output(self,chall):
        # Set challenge
        for stage,bit in zip(self._stages,chall):
            stage.set_selector(bit)

        # Compute output
        delay = (0,0)
        for s in self._stages:
            delay = s.get_output(delay[0],delay[1])

        return 0 if delay[0] < delay[1] else 1
```

```
N  = 32     # Size of the PUF
LS = 600    # Size learning set
TS = 10000  # Size testing set
apuf = ArbiterPUF(N)

# Creating training suite
learningX = [[random.choice([0,1]) for _ in range(N)] for _ in range(LS)]
learningY = [apuf.get_output(chall) for chall in learningX] # Outputs PUF

# Creating testing suite
testingX = [[random.choice([0,1]) for _ in range(N)] for _ in range(TS)]
testingY = [apuf.get_output(chall) for chall in testingX]

# Convert challenges into feature vectors
learningX = [into_features_vect(c) for c in learningX]
testingX = [into_features_vect(c) for c in testingX]
```

3 различных алгоритма

```
lr = LogisticRegression(solver='lbfgs')
lr.fit(learningX, learningY)
print("Score arbiter PUF (%d stages): %f" % (N,lr.score(testingX,testingY)))
```

```
Score arbiter PUF (32 stages): 0.975500
```

```
svc = SVC(gamma='scale')
svc.fit(learningX, learningY)
print("Score arbiter PUF (%d stages): %f" % (N, svc.score(testingX, testingY)))
```

```
Score arbiter PUF (32 stages): 0.927400
```

```
gb = GradientBoostingClassifier()
gb.fit(learningX, learningY)
print("Score arbiter PUF (%d stages): %f" % (N, gb.score(testingX, testingY)))
```

```
Score arbiter PUF (32 stages): 0.866400
```

Какой размер обучающей выборки необходим, чтобы достигнуть доли правильных ответов минимум 0.95? Как зависит доля правильных ответов от N?