分組名單（不足 5 個人空著就好）：

姓名	學號
施柏江	110550108
王振倫	110550068
林英碩	110550117
呂偉祥	110550115

1. Name of the paper:

   New Directions in Cryptography

2. Summary:

   這篇論文探討了密碼學領域的兩個新發展方向，即公鑰加密系統和單向認證系統，並分析了它們與傳統密碼學技術的差異。公鑰加密系統試圖解決如何在不安全的通訊途徑上進行安全的通訊，在雙方不共享私鑰的情況下，仍能確保通訊內容的機密性和完整性的方法。

   文章中加密系統的實作方式使用一對密鑰，即公鑰與私鑰。公鑰可以公開分發，而私鑰則由擁有者保密，且確保第三方無法使用公鑰來計算出私鑰，使用者可以用對方的公鑰加密訊息，而只有擁有對應私鑰的人才能解密。這樣避免了預先共享密鑰的需求，也能夠簡化密鑰管理的難題。這套系統為解決安全通訊中的密鑰分配問題提供了一個有效的解決方案，對於現代密碼學和訊息安全技術的發展奠定了基礎。

   此論文也討論了密碼學與計算複雜性理論的結合，為構建安全的密碼學系統提供了新的理論打底。在文章的最後，作者甚至還從歷史的角度去分析了未來密碼學可能的發展方向。

3. Strength(s) of the paper:

   這篇論文是第一個提出公鑰密碼學概念的論文，提升了在不安全的通訊途徑上進行安全通訊的可行性。而這個概念解決了現實中的許多問題，特別是在密鑰分發和管理方面，降低了實際應用情況下的安全風險，這讓訊息安全在各種應用場景中得以實現，提升了整體系統的安全性，也讓用戶更容易操作。

   此外，論文中的數學理論和分析非常嚴謹，為後續的研究和實際應用提供了穩固的理論。這些理論也被廣泛應用於現代加密演算法中，我們可以在 Google 學術搜尋網站上看到，這篇論文已被上千份學術文章所引用，可見此論文的影響性之巨大。除此之外，文章中的數學說明非常直觀，讓很多複雜

的概念能夠被更廣泛的讀者群眾去理解和閱讀，這在推廣和應用上具有重要意義。

我們能夠發覺文章作者大力推動密碼學和訊息安全領域的發展，可謂是學術界的教科書，對現代加密技術的發展也有很大的影響，許多現代加密技術和標準大抵都有基於這篇論文的理論和方法而去發展。

4. Weakness(es) of the paper

雖然這篇論文提出了公鑰密碼學的概念，但在具體實現細節上描述地不夠詳盡。例如論文中沒有提供如何生成安全的密鑰對的具體演算法，也沒有詳細描述如何在實際系統中實現這些理論，當時的安全性分析主要基於理論推導，而應該未經過實際的攻擊測試和大量實驗驗證。隨著時間推移，面對現代技術的進步和攻擊手段的多樣化，這些過去的分析在現今標準下可能會略顯不足。

此外，公鑰密碼學在計算複雜度上比傳統的對稱加密高得許多，這在當時計算資源有限的情況下應該是一個滿大的應用障礙，雖然現今計算能力有所提高，但中間的計算成本依然是公鑰加密的一個考量因素。

最後，在這篇論文發表的時間點，量子運算還沒有成為資安界的威脅。然而現代的研究表明，量子運算可能能夠破解現有的公鑰加密算法，這在當時是沒有辦法被考慮到的。

5. Your own reflection, which can include but not limited to:

A. What did you learn from this paper?

我們從這篇論文中學到了公鑰加密系統的原理及重要性，透過這套系統能夠讓我們在公開的環境上和陌生人通訊，在不損失其安全性的同時也不用透過物理方式來交換密鑰，解決了傳統加密系統的限制，大大的提升了加密技術的應用範圍。

此外，我們還學到了單向函數在加密系統中的重要性，這種容易計算但難以反向計算的函數奠定了公鑰加密系統的基礎。

B. How would you improve or extend the work if you were the author?

在上面的弱點中我們提到，在論文發表的年代可能不會被現今的各種五花八門攻擊手段測試過安全性。因此，如果我們是作者，會希望為此研究建立一個完善的測試環境，此環境能夠模擬不同類型的攻擊，像是中間人攻擊、暴力破解等等，主要也會包含現代的攻擊手法。藉由這些模擬攻

擊，我們可以更加全面地了解這系統在應用時的安全性是否存在漏洞，不管有沒有都能讓此研究更加完善至臻。

C. What are the unsolved questions that you want to investigate?

論文中提到：”If an algorithm whose complexity grew as log2(q) were to be found, our system would be broken.”，以此我們想到了以下幾個問題：

(1) 提出論文的時候，看起來計算離散對數的最佳已知算法的複雜性是隨 q 的平方根增長的。然而，在現今的運算能力下是否存在一個複雜性是隨 log2(q)增長的算法？

(2) 是否會存在未發現的數學工具或技術可以提高計算對數效率的方法？這個工具或技術或許會對於整個密碼領域的造成很大的影響。

(3) 隨著量子計算技術的蓬勃發展，量子計算是否可以會被用來計算此對數問題？又或者是我們是否可以應用這樣的能力，再進一步提高整個算法的安全性，來應對可能的量子計算攻擊？

D. What are the broader impacts of this proposed technology?

透過這篇論文提出的公鑰加密系統，除了可以讓陌生人之間公開進行安全的通訊以外，也讓密鑰分配的方式得到大大的簡化，不再需要利用物理方式直接把密鑰送到其他人的手中，顯著提升加密溝通的效率，對現代大多加密通訊技術都有很大的影響。

6. Realization of a technical specification or algorithm as a program:

(1) Diffie-Hellman key exchange:

Code implementation:

```
# Diffie-Hellman key exchange
import random

p = 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD1
p *= 0x29024E088A67CC74020BBEA63B139B22514A08798E3404DD
p *= 0xEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245
p *= 0xE485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7ED
p *= 0xEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE65381
p *= 0xFFFFFFFFFFFFFFFF

g = 2
# User A generates private key a and calculate public key A
a = random.randint(1, p - 1)
A = pow(g, a, p)
# User B generates private key b and calculate public key B
b = random.randint(1, p - 1)
B = pow(g, b, p)
# User A calculate shared key sA
sA = pow(B, a, p)
# User B calculate shared key sB
sB = pow(A, b, p)
print(f"User A's private key: {a}")
print(f"User A's public key: {A}")
print(f"User B's private key: {b}")
print(f"User B's public key: {B}")
print(f"User A's calculated shared key': {sA}")
print(f"User B's calculated shared key': {sB}")

# Confirm that the shared keys are the same
assert sA == sB, "Shared keys are different!"
print("Shared keys are the same, key exchanged successfully!")
```

Steps:

i. 定義一個很大的質數 p，以及一個小於 p 的整數 g

ii. 隨機生成界於 1 到 p − 1 之間的私鑰 a 跟 b

iii. 生成公鑰 $\begin{cases} A = g^a \ mod \ \mathrm{p} \\ B = g^b \ mod \ p \end{cases}$

iv. 生成共享鑰 $\begin{cases} sA = B^a \ mod \ p \\ sB = A^b \ mod \ p \end{cases}$

v. 如果共享鑰一樣即成功

Diffie-Hellman key exchange output:

```
User A's private key: 11936964989315979554285776543059619708070731535369003812365380812852895101351533113421078554485125223031733746113152540294563803424644127934279750861125089751403701997763414813369691702160938919139399993115033995498795544464300813888552694033772068593186397844856278253504361526323696717370532118237297142931

User A's public key: 90377551849653093455313445088254057751092980136489735694254124945296767351275344015682469343571925930004432197912539469149597987492780369171454660613018008089176897372465705306825243348625885137940505266777060518745881462844977207582587472597917394186285288877896057071196338461647267838104030886444193675 23

User B's private key: 77952856833184232840998531279775465164001206161827980353125398361570373160726346603137049579726159727464944212594100582677359914984563264945611125529049824131401227253652349398196771225976469704639111413725477254309750960368156089840812976699124207638505455279013512978065777988251234146943122914076363371 00

User B's public key: 20395391698425540240424412959775380486252509742338287232564961646199025384197140167197651661709944850368743929987147007794840873050046657682190912946590389180470158797042260951953580382579017690162978007676996701862386638762198662710983637035075101464576465213869091310811949580030016232099466443553306873501

User A's calculated shared key': 68344108917200706149478616468723907203054694923610697550444446601554898924150068953090624576331500998163599950335685377828196267573503427002974804330878381645186386018079754492048454622340895765494629101175910224506979233223760306572656643559035375693132856304066580177559400583739721848084332935 9895621001

User B's calculated shared key': 68344108917200706149478616468723907203054694923610697550444446601554898924150068953090624576331500998163599950335685377828196267573503427002974804330878381645186386018079754492048454622340895765494629101175910224506979233223760306572656643559035375693132856304066580177559400583739721848084332935 9895621001

Shared keys are the same, key exchanged successfully!
```

(2) RSA: `pip install sympy` first

Code implementation:

```python
# RSA
import random
from sympy import isprime, gcd, mod_inverse


def generate_prime_candidate(length):
    p = random.getrandbits(length)
    # apply a mask to set MSB and LSB to 1
    p |= (1 << length - 1) | 1
    return p


def generate_prime(length=1024):
    p = 0
```

```
    while not isprime(p):
        p = generate_prime_candidate(length)
    return p

def generate_keypair(keysize):
    p = generate_prime(keysize)
    q = generate_prime(keysize)
    n = p * q
    phi = (p - 1) * (q - 1)

    e = random.randrange(1, phi)
    g = gcd(e, phi)
    while g != 1:
        e = random.randrange(1, phi)
        g = gcd(e, phi)

    d = mod_inverse(e, phi)
    return ((e, n), (d, n))

def encrypt(keypair, plaintext):
    key, n = keypair
    cipher = [pow(ord(char), key, n) for char in plaintext]
    return cipher

def decrypt(keypair, ciphertext):
    key, n = keypair
    plain = [chr(pow(char, key, n)) for char in ciphertext]
    return ''.join(plain)

message = "Cryptography Engineering"

# Key generation
public, private = generate_keypair(512)

print("Public key:", public)
print("Private key:", private)
```

```python
# Encryption
encrypted_msg = encrypt(public, message)
print("Encrypted message:", encrypted_msg)

# Decryption
decrypted_msg = decrypt(private, encrypted_msg)
print("Plaintext:", message)
print("Decrypted message:", decrypted_msg)

assert message == decrypted_msg, "Wrong message!"
print("Plaintext and decrypted message are the same, decrypted successfully!")
```

RSA output:

```
Plaintext: Cryptography Engineering
Decrypted message: Cryptography Engineering
Plaintext and decrypted message are the same, decrypted successfully!
```

這是我們實作出大家熟知的 RSA 演算法的部分，利用大質數分別產生公鑰以及私鑰，這正是這篇論文所提到的公鑰加密系統，上面的程式即利用公鑰加密後再用私鑰解密的過程。