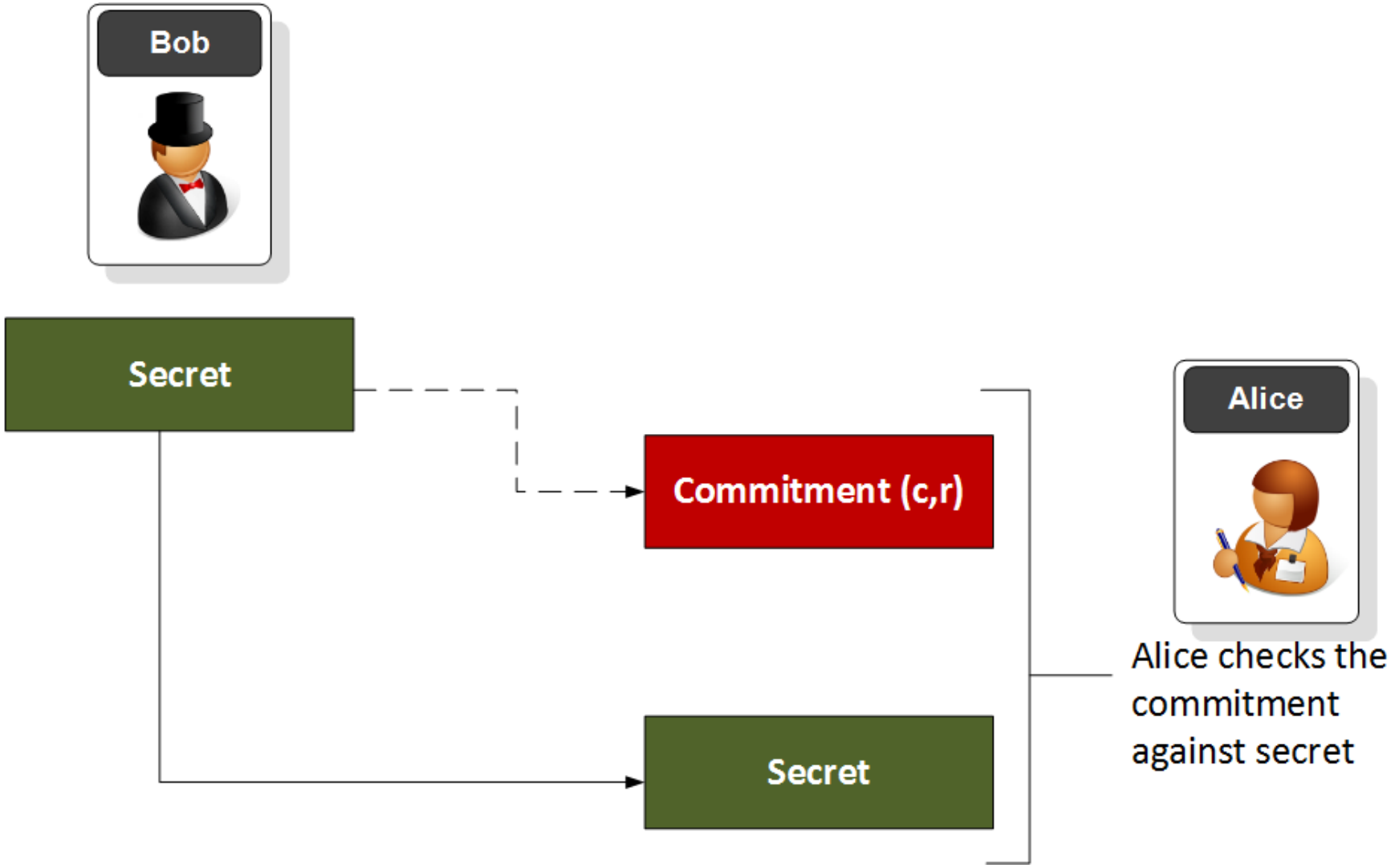
Pedersen Commitment

Prof Bill Buchanan, The Cyber Academy

http://asecuritysite.com

CYBER ACADEMY

Pedersen Commitment: Perfect Secrecy



Pedersen Commitment (Discrete Logs)

In the Pedersen commitment we take two large prime numbers (p and q) and and we create a generator value (g) which is of the order of q and a subgroup of Z_p^*. Then s becomes a secret from 0 to Z_q, and we calculate:

$$h = g^s \pmod{p}$$

The sender now creates a commitment for a message (m) and with a random number (r):

$$c = g^m h^r \pmod{p}$$

The sender can then send c to the receiver. Next, the sender will then reveal m and r for the commitment, and the receiver verifies with:

$$c = g^m h^r \pmod{p}$$

If the values match, the receiver has proven the commitment.

We can also use as homomorphic encryption, and where we multiply commitments, and which is equivalent to adding the secret values.

Pedersen Commitment (Disc

In the Pedersen commitment we take two larg
value (g) which is of the order of q and a subg
calculate:

$$h = g^s \pmod{p}$$

The sender now creates a commitment for a m

$$c = g^m h^r \pmod{p}$$

The sender can then send c to the receiver. Ne
and the receiver verifies with:

$$c = g^m h^r \pmod{p}$$

If the values match, the receiver has proven tl

We can also use as homomorphic encryption, a
to adding the secret values.

```
class verifier:
    def setup(self, security):
        p = number.getPrime(2 * security, Random.new().read)
        q = 2*p + 1



        g = number.getRandomRange(1, q-1)
        s = number.getRandomRange(1, q-1)
        print "Secret value:\t",s
        h = pow(g,s,q)

        param = (p,q,g,h)
        print "p=",p
        print "q=",q
        print "g=",g
        print "h=",h

        return param
```

Pedersen Commitment (Disc

In the Pedersen commitment we take two larg
value (g) which is of the order of q and a subg
calculate:

$$h = g^s \pmod{p}$$

The sender now creates a commitment for a m

$$c = g^m h^r \pmod{p}$$

The sender can then send c to the receiver. Ne
and the receiver verifies with:

$$c = g^m h^r \pmod{p}$$

If the values match, the receiver has proven tl

We can also use as homomorphic encryption, a
to adding the secret values.

```
class veri
    def se
        p =
        q =
        g =
        s =
        pri
        h =
        par
        pri
        pri
        pri
        ret
```

```
security = 80
msg1 = 1
msg2 = 2

if (len(sys.argv)>1):
        msg1=int(sys.argv[1])

if (len(sys.argv)>2):
        msg2=int(sys.argv[2])

v = verifier()
p = prover()

param = v.setup(security)

c1, r1 = p.commit(param, msg1)
c2, r2 = p.commit(param, msg2)

addCM = v.add(param, c1, c2)

print "\nMsg1:",msg1
print "Msg2:",msg2

print "\nc1,r1:",c1,",",r1
print "c2,r2:",c2,",",r2
print "\nWe can now multiply c1 and c2, which is same as adding Msg1 and Msg2"
print "\nCommitment of adding (Msg1+Msg2):\t",addCM

result1 = v.open(param, c1, msg1, r1)
result2 = v.open(param, c2, msg2 , r2)

print "\nResult of verifying c1:\t\t",result1
print "Result of verifying c2:\t\t",result2

result = v.open(param, addCM, msg1 + msg2 , r1, r2)

print "Result of verify Msg+Msg2:\t",result
```

Pedersen Commitment (Disc

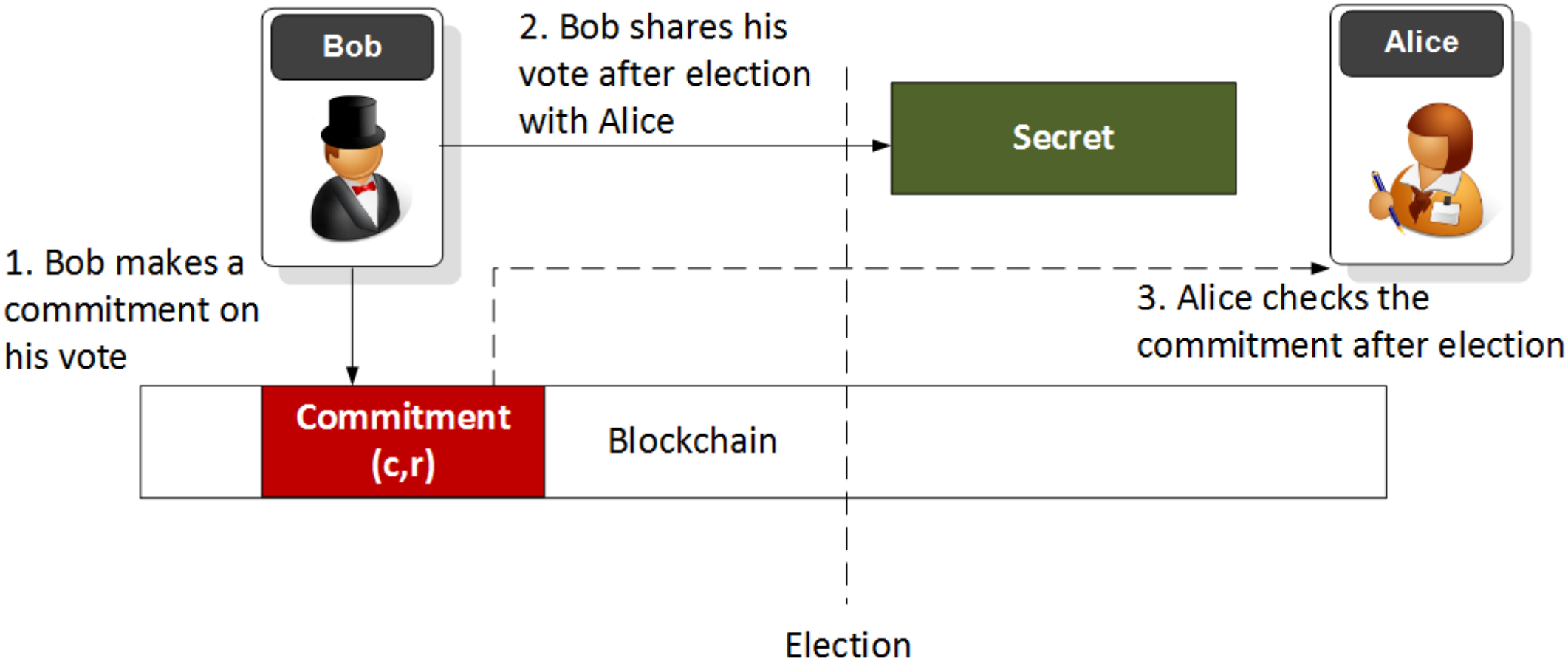
```
class veri
    def set
        n = if (len(sys.argv)>1):
```

```
security = 80
msg1 = 1
msg2 = 2
```

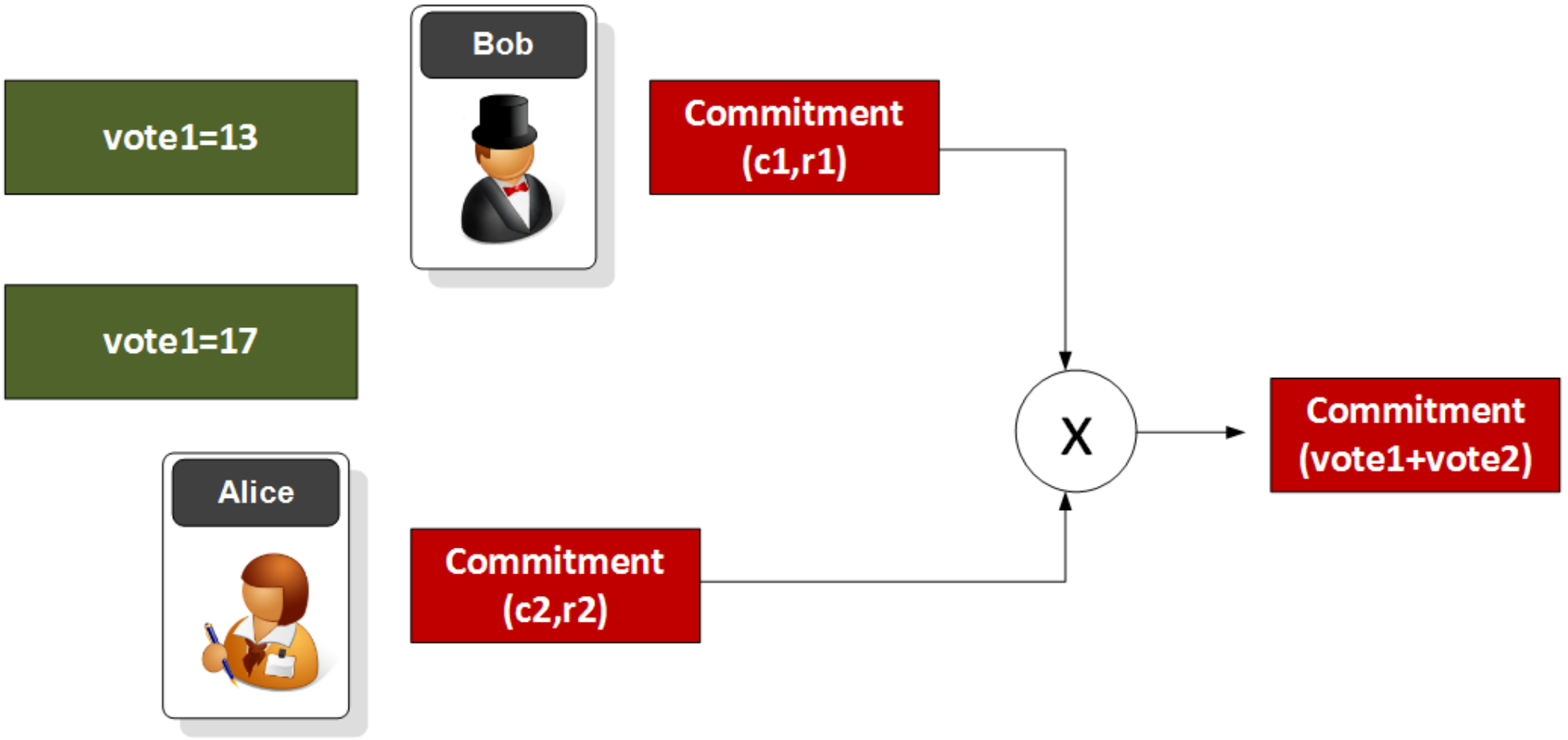
```
Secret value:   160021850381606860961606826218608521138591291344 9
p= 835633126610936170404251455173303609923742171077
q= 167126625322187234080850291034660721984748434215 5
g= 853879156962728674489578584402624767640115848839
h= 312816528258481903392461143384659987330939968504

Msg1: 70
Msg2: 70

c1,r1: 93143260618730859936276009038355099821281759741 9 , 133154063878955883975175775621859189924022928
7857
c2,r2: 377312755894089489173415862148929013782547633796 , 106229632588669033453493136803351658193427515
3224

We can now multiply c1 and c2, which is same as adding Msg1 and Msg2

Commitment of adding (Msg1+Msg2):       943416350824651526788742276019896656762684796624

Result of verifying c1:         True
Result of verifying c2:         True
Result of verify Msg+Msg2:      True
```

Msg2"

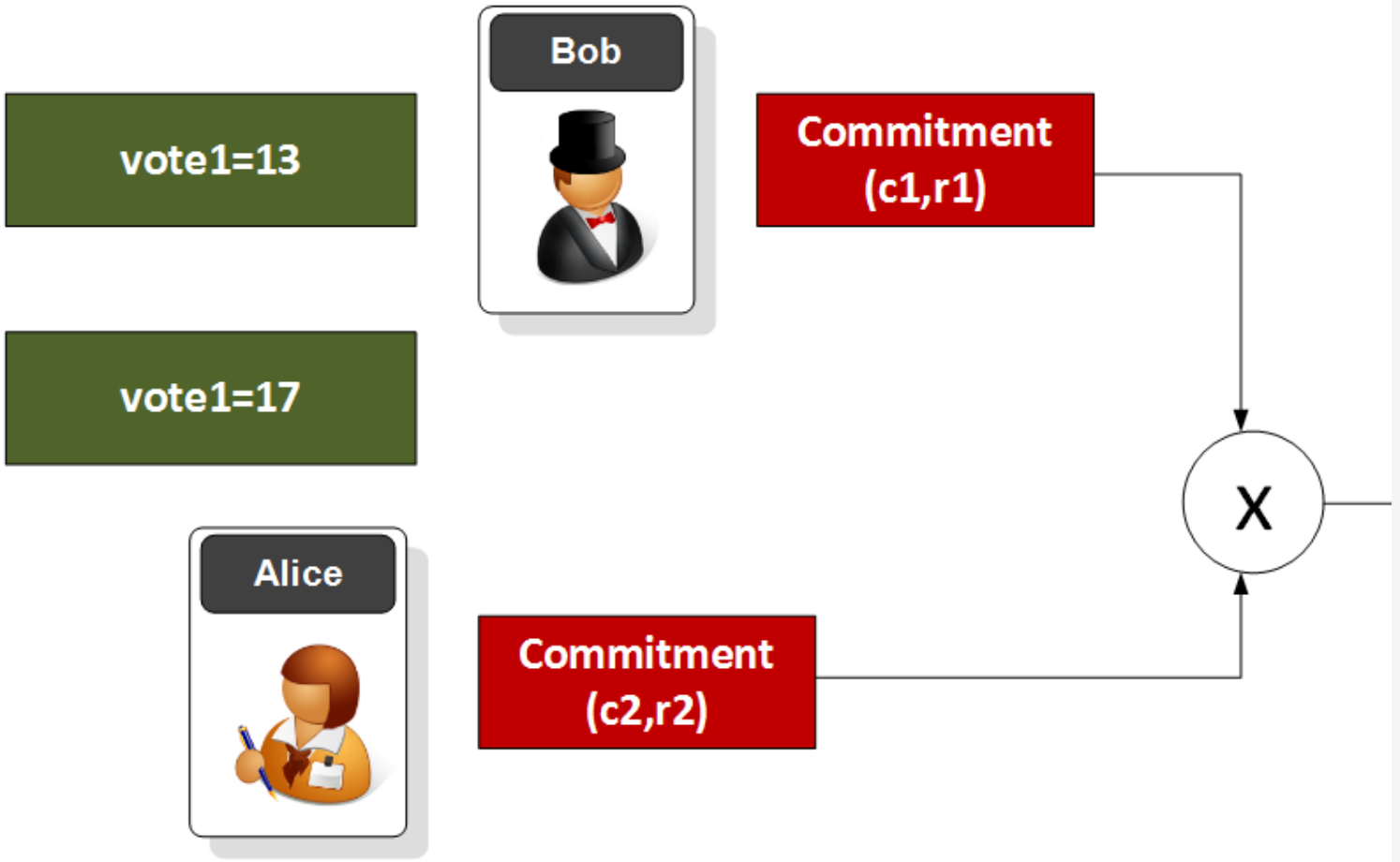
Pedersen Commitment: Voting



Pedersen Commitment: Homomorphic Encryption



Pedersen Commitment: Homomorphic Encryptic




```
    def add(self, param, *cm):
        addCM = 1
        for x in cm:
            addCM *= x
        addCM = addCM % param[1]
        return addCM

class prover:
    def commit(self, param, x):
        q,g,h = generate(param)

        r = number.getRandomRange(1, q-1)
        c = (pow(g,x,q) * pow(h,r,q)) % q
        return c, r


security = 80
msg1 = 1
msg2 = 2

if (len(sys.argv)>1):
        msg1=int(sys.argv[1])

if (len(sys.argv)>2):
        msg2=int(sys.argv[2])

v = verifier()
p = prover()

param = v.setup(security)

c1, r1 = p.commit(param, msg1)
c2, r2 = p.commit(param, msg2)

addCM = v.add(param, c1, c2)
```

Pedersen Commitment

Prof Bill Buchanan, The Cyber Academy

http://asecuritysite.com

CYBER ACADEMY