Privacy Engineering (70018)

MPC 2 - Solutions

2.1

$c[1]\ [0,0] = E_{k[\underline{1},0],k[3,0]}(k[\underline{5},0], 1)$

$c[1]\ [0,1] = E_{k[\underline{1},0],k[3,1]}(k[\underline{5},0], 1)$

$c[1]\ [1,0] = E_{k[\underline{1},1],k[3,0]}(k[\underline{5},0], 1)$

$c[1]\ [1,1] = E_{k[\underline{1},1],k[3,1]}(k[\underline{5},1], 0)$

$c[3]\ [0,0] = E_{k[\underline{5},1],k[\underline{6},1]}(k[\underline{7},1], 1)$

$c[3]\ [0,1] = E_{k[\underline{5},1],k[\underline{6},0]}(k[\underline{7},1], 1)$

$c[3]\ [1,0] = E_{k[\underline{5},0],k[\underline{6},1]}(k[\underline{7},1], 1)$

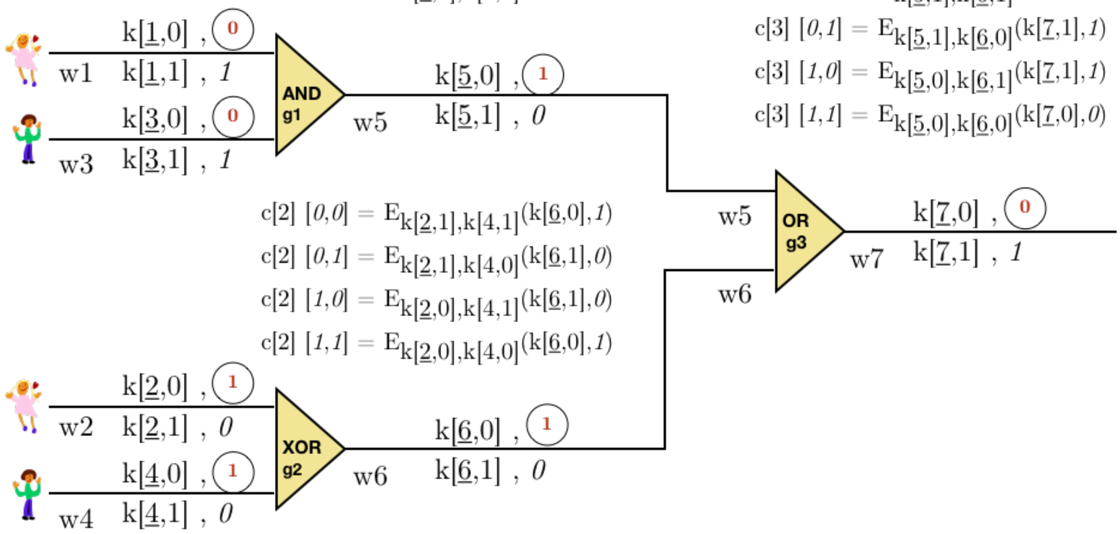
$c[3]\ [1,1] = E_{k[\underline{5},0],k[\underline{6},0]}(k[\underline{7},0], 0)$

$c[2]\ [0,0] = E_{k[\underline{2},1],k[4,1]}(k[\underline{6},0], 1)$

$c[2]\ [0,1] = E_{k[\underline{2},1],k[4,0]}(k[\underline{6},1], 0)$

$c[2]\ [1,0] = E_{k[\underline{2},0],k[4,1]}(k[\underline{6},1], 0)$

$c[2]\ [1,1] = E_{k[\underline{2},0],k[4,0]}(k[\underline{6},0], 1)$



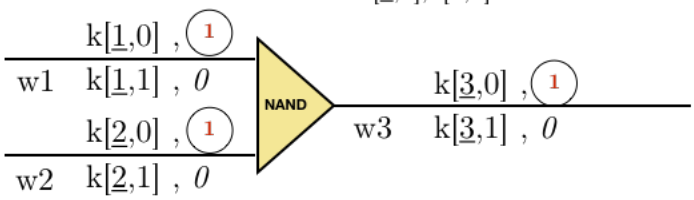
2.2

$c[0,0] = E_{k[\underline{1},1],k[2,1]}(k[\underline{3},0], 1)$

$c[0,1] = E_{k[\underline{1},1],k[2,0]}(k[\underline{3},1], 0)$

$c[1,0] = E_{k[\underline{1},0],k[2,1]}(k[\underline{3},1], 0)$

$c[1,1] = E_{k[\underline{1},0],k[2,0]}(k[\underline{3},1], 0)$



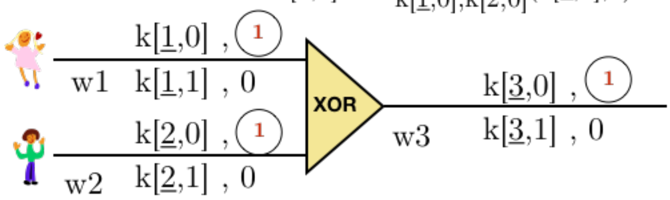
2.3

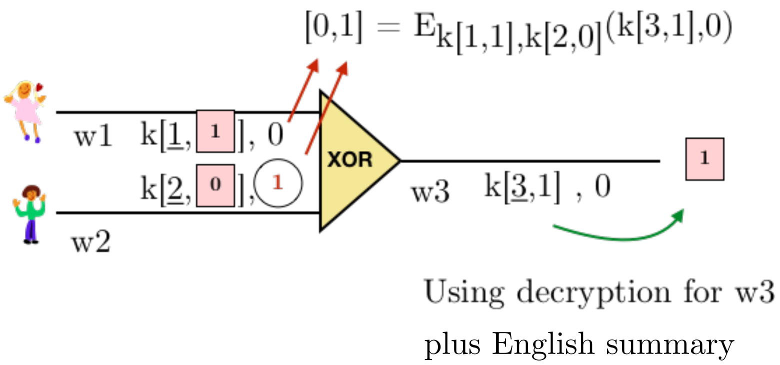
$[0,0] = E_{k[\underline{1},1],k[2,1]}(k[\underline{3},0],1)$

$[0,1] = E_{k[\underline{1},1],k[2,0]}(k[\underline{3},1],0)$

$[1,0] = E_{k[\underline{1},0],k[2,1]}(k[\underline{3},1],0)$

$[1,1] = E_{k[\underline{1},0],k[2,0]}(k[\underline{3},0],1)$



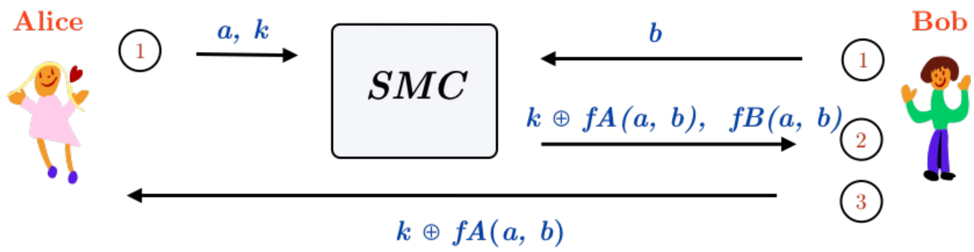


2.4 Hint: Karnaugh maps are an easy method to do this from a truth table. A good solution will be about 9-10 gates. Very many solutions are possible however.

2.5 We can replace ***fA*** and ***fB*** by a single function that satisfies

$$\boldsymbol{f(a,\ b,\ k)\ =\ k\ \oplus\ fA(a,\ b),\ \ fB(a,\ b)}$$

where ***k*** is a a secret input (a key!) as long as the maximum output possible for ***fA(a, b)*** (in bits). c.f. "One-time pad". Only Bob learns the output of this function.



At 2. Bob learns $f(a,b,k) = k \oplus fA(a,b),\ fB(a,b)$ from the MPC protocol
Bob sends first part $k \oplus fA(a,b)$ to Alice, keeps second part *fB(a,b)*

At 3. Alice computes $fA(a,b) = k \oplus (k \oplus fA(a,b))$ by xoring with secret k

2.6 i) Bob's output is $\mathrm{D}_{kb}\ (\mathrm{E}_x(M_b))$ where $x = \mathrm{D}_{privb}\ (\mathrm{E}_{pubb}(k_b)) = k_b$
giving $\mathrm{D}_{kb}\ (\mathrm{E}_{kb}(M_b)) = M_b$

ii) Alice is told $G_1{=}k_1,\ G_2{=}k_2,\ ..\ G_b{=}\mathrm{E}_{pubb}(\mathrm{k}_b),\ \ldots\ G_n{=}k_n$ However these are just binary values, she doesn't know which one Bob will use for the final decryption. We need to assume that the G values are indistinguishable e.g. are padded to the same length, otherwise Alice could assume the symmetric key lengths from the public-key encryption.

iii) If Bob attempts to decrypt using z != b, for example, if b !=*1* Bob will get:

$$\mathrm{D}_{k1}\ (\mathrm{E}_x(M_1)) = \text{rubbishMessage because x} = \mathrm{D}_{priv1}\ (k_1) = \text{rubbish key}$$

This assumes that x is an acceptable key for the Symmetric Encryption function, i.e. securely padded/truncated to length.

iv) If Alice is dishonest, she does not need to run the protocol correctly e.g. not use random numbers, could re-use values from a previous run, could use a different encryption function etc. Alice might be able determine b using the difference in the size of G elements. Alice could also encrypt the same message in step 3, essentially controlling which secret Bob gets, i.e. b is irrelevant

v) Bob can be dishonest in step 2 he could do $G_z = E_{pub_z}(k_z)$ for all keys, then he can learn all secrets M_1 to M_z . Even he can set key $k_1=k_2=k_3=...=k_n$.

2.7 i) In step 3, B is either $B = g^b$ or $B = Ag^b = g^a g^b$

In step 4, keys k_0 and k_1 are either hashed from $B = g^b$ if Bob's selection bit is $m=0$ we have

k_0 from $(g^b)^a = g^{ab}$

k_1 from $(g^b / g^a)^a = g^{ab} / g^{aa}$

or keys k_0 and k_1 are hashed from $B = g^a g^b$ if Bob's selection bit $m=1$ we have

k_0 from $(g^a g^b)^a = g^{aa+ab}$

k_1 from $(g^a g^b / g^a)^a = g^{ab}$

In both cases one of the keys is hashed from g^{ab}, Bob is also able to generate this key from the hash $A^b = g^{ab}$.

If $m=0$ Bob will correctly decrypt M_0. If $m=1$ Bob will correctly decrypt M_1.

ii) Alice is told either g^b or $g^a g^b$ which are just two random values. The keys produced by Alice are distinct, Bob is only able to decrypt 1 message.
We assume that both the DH crypto-setup, hash function and message encryption scheme are secure. Bob must not be able to compute key g^{ab} from either g^{aa+ab} or $g^a g^b$ otherwise he will know both keys.

iii) If Alice sets a=0. This will give $B= g^b$ for both m=0 and m=1. Alice doesn't learn m from this. Alice will generate $k_0= g^0$=1 and $k_1= g^0$=1. Bob will learn both messages.
If Bob sets b=0. This will give $B= g^0$=1 for m=0 and $B= g^a$ for m=1. Assuming the protocol is known to Alice then she will learn m and know which message Bob chose.

2.8 (i) For $M_0 = 1101$, $M_1 = 0100$, $b = 1$, $t = 0$, $R_0 = 0101$, $R_1 = 0011$ we have

$e = t \oplus b = 0 \oplus 1 = 1$

$C_0 = M_0 \oplus R_e = 1101 \oplus 0011 = 1110$

$$C_1 = M_1 \oplus R_{1\text{-}e} \quad = 0100 \oplus 0101 \quad = 0001$$

$$M_b = C_b \oplus R_t \quad = C_1 \oplus R_0 = 0001 \oplus 0101 = 0100 = M_1$$

(ii) At the end of the protocol Alice has no information about b, only the encrypted bit $t \oplus b$, t is randomly selected by Trent and only known to Bob.

- Bob only learns M_b. In order to decrypt $C_{1\text{-}b}$ Bob would need the random value $R_{1\text{-}t}$ which only Alice has.
- The only value that Bob controls is e, setting it to a specific value will not help learn both messages.
- Alice could set M_0=M_1 and will then know what message Bob has.
- Collusion with Trent is also possible.

(iii) Protocol for 1-from-n where n is a power of 2. b is a *log2(n)* bit value in range 0..n-1

1. Trent → Alice: R_0 to $R_{n\text{-}1}$ Random binary values each of length k
2. Trent → Bob: t, R_t Random log2(n) bit-value t in the range *0..n-1*
3. Bob → Alice: e $e = t \oplus b$
4. Alice → Bob: C_0 to $C_{n\text{-}1}$ $C_j = M_j \oplus R_{(e+j) \bmod n}$ *for j in 0..n-1*
5. Bob $M_b = C_b \oplus R_t$

2.9 (i) In step 4, if b=0 then Bob sends H_0=g^k and H_1=x/H_0

Bob will correctly decrypt M_0 i.e.

$$\begin{aligned} C_0 \oplus Hash(D^y) &= M_0 \oplus Hash(H_0^k) \oplus Hash(D^y) \\ &= M_0 \oplus Hash(g^{ky}) \oplus Hash(g^{ky}) = M_0 \end{aligned}$$

if b=1 then Bob sends H_0=x/H_1 and H_1=g^k and will correctly decrypt *M1, i.e.*

$$\begin{aligned} C_1 \oplus Hash(D^y) &= M_1 \oplus Hash(H_1^k) \oplus Hash(D^y) \\ &= M_1 \oplus Hash(g^{ky}) \oplus Hash(g^{ky}) = M_1 \end{aligned}$$

(ii) After step 2, Alice receives H0 and H1 from Bob. Alice cannot determine b since *H0* is a random element of the group.

Bob cannot learn $M_{1\text{-}b}$ since Bob would need to computer $Hash(H_{1\text{-}b}^k)$ but *Hash* is random cryptographic hash (oracle), or would need to solve the Diffie-Hellman problem on the group.

If b=0 and Bob tries to decrypt M1 he will get a bad message, i.e.

$$C_1 \oplus Hash(D^y) = M_1 \oplus Hash(H_1^k) \oplus Hash(D^y)$$

$$= M_1 \oplus Hash((x/H_0)^k) \oplus Hash(g^{ky})$$

$$= M_1 \oplus Hash((x/g^y)^k) \oplus Hash(g^{ky})$$

Bob will also get a bad message if b=1 and he tries to decrypt M0 i.e. (okay to omit)

$$C_0 \oplus Hash(D^y) = M_0 \oplus Hash(H_0^k) \oplus Hash(D^y)$$

$$= M_0 \oplus Hash((x/H_1)^k) \oplus Hash(g^{ky})$$

$$= M_0 \oplus Hash((x/g^y)^k) \oplus Hash(g^{ky})$$