3. MPC - Garbled Circuits

Naranker Dulay

n.dulay@imperial.ac.uk

https://www.doc.ic.ac.uk/~nd/peng

“Security/Privacy is about replacing Trust with Mathematics” ?

November 2020 (v1)

Secret Sharing vs Garbled Circuits

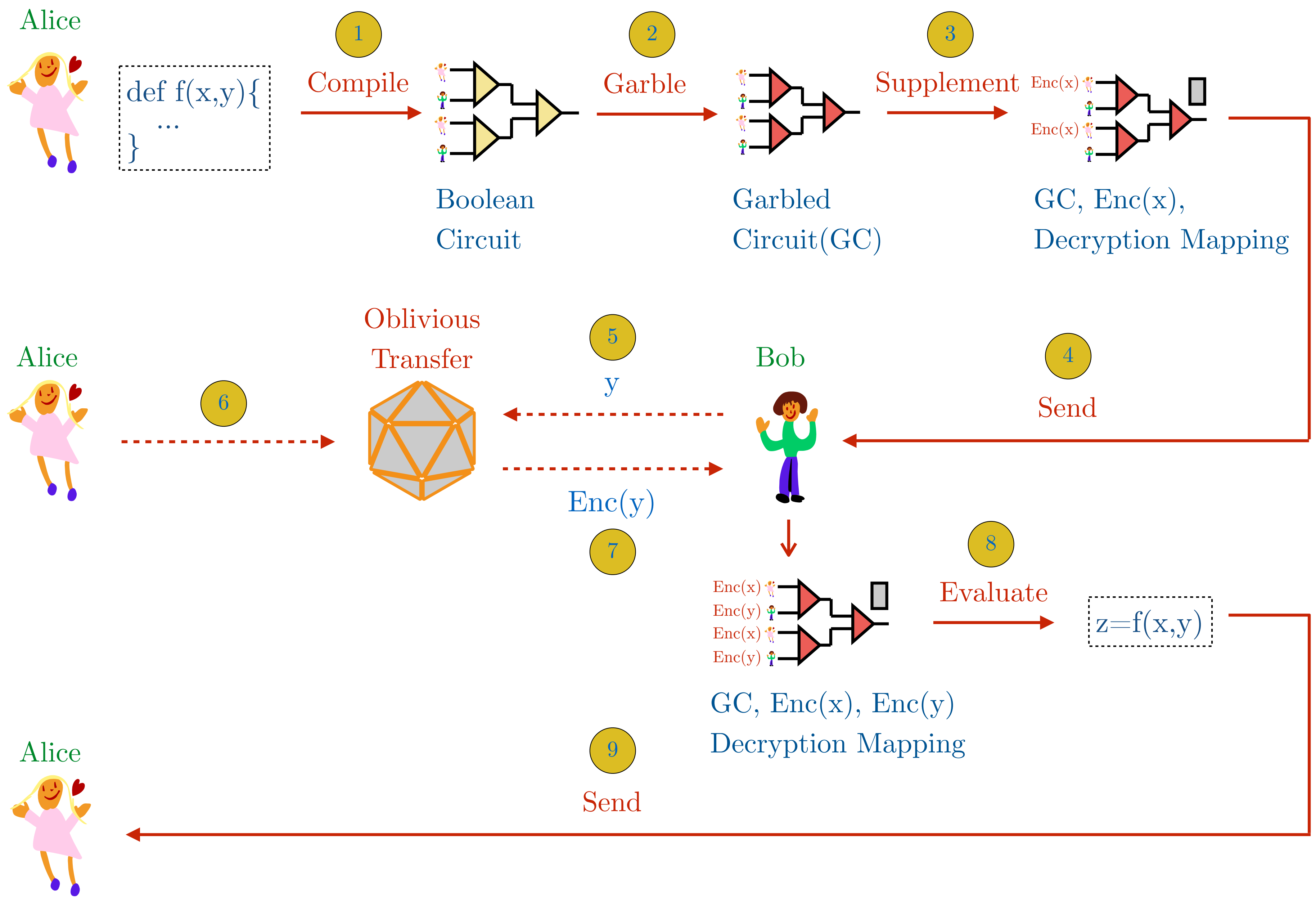
Secret Sharing (SS)

- Express the function to be computed as an arithmetic circuit, e.g. addition and multiplication gates.
- Secret sharing is better suited for multi-party (3+) outsourced service setups.
- Secret sharing has high communication costs so we need keep the number of parties low.
- We'll look at the BGW secret sharing protocol

Garbled Circuits

- Express the function to be computed as an "encrypted" boolean circuit, e.g. with ANDs, ORs, XORs, etc
- Garbled circuits are more efficient than for 2-party joint processing setups. They get too complex for more parties.
- We will look Yao's Garbled Circuits protocol for 2-parties.

Two-party Garbled Circuit Overview

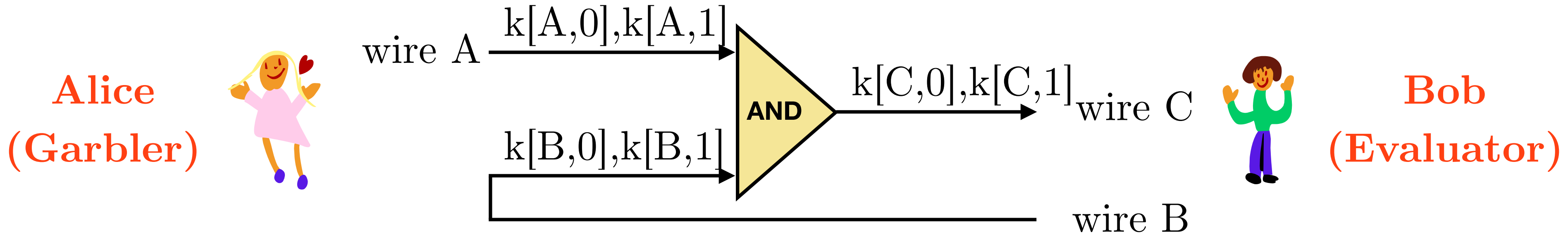


Notes

- Garbled (i.e. encrypted) circuits are a general approach used to perform a privacy-preserving multi-party computation.
- Given two parties computations, Alice and Bob, we have:
- **Aim**: To compute a function $f(x, y)$ where x is Alice's input and y is Bob's input such that x and y remain private unless revealed by the function f.
- Alice will be the circuit garbler, Bob the circuit evaluator.
- 1. Alice compiles f into an acyclic boolean circuit, composed of logic gates like AND, OR, XOR etc. For tiny functions we can do this manually using techniques such as a *Karnaugh Maps*.
- 2. Alice garbles the circuit to produce a garbled circuit, essentially each gate will carry out its operation on encrypted input bits producing an encrypted output bit.
- 3-4. Alice sends the Garbled circuit plus (i) the encrypted bits for *its* input x, and (ii) a decryption mapping table that allows Bob to map the encrypted output of the circuit to its plaintext.
- 5-7. Bob initiates an oblivious transfer protocol (see later) with Alice for each bit of its input y. Essentially, this results in Bob obtaining from Alice, encrypted bits for each plaintext bit of y, without Alice learning y.
- 8-9. Bob now evaluates the circuit using the encrypted bits for x and y as inputs to the garbled circuit.

  Bob then uses the decryption mapping table that Alice sent to map the encrypted output to its plaintext. Bob shares the plaintext result with Alice.

One Gate Garbled Circuit (Alice)



Garbled (Encrypted) Truth Table for AND

$GT\ [0,0] = E_{k[A,0],k[B,0]}$ (k[C,AND(0,0)])

$GT\ [0,1] = E_{k[A,0],k[B,1]}$ (k[C,AND(0,1)])

$GT\ [1,0] = E_{k[A,1],k[B,0]}$ (k[C,AND(1,0)])

$GT\ [1,1] = E_{k[A,1],k[B,1]}$ (k[C,AND(1,1)])

Notes

- For each of three wires of the AND gate (2 input wires, 1 output wire):
- Alice creates 2 random keys.
- One key corresponds to 0, the other key to 1. For example, for wire A, the random keys are k[A,0] and k[A,1] for 0 and 1 respectively.
- Alice also computes a garbled truth table for the gate, where each entry in the garbled table is the *key* that corresponds to output of the logical operation, doubly-encrypted with the appropriate input keys.
- For example, the AND garbled truth table entry for Wire A input 1 and Wire B input 0, is Key k[C,0] doubly-encrypted by Keys k[A,1] and k[B,0]

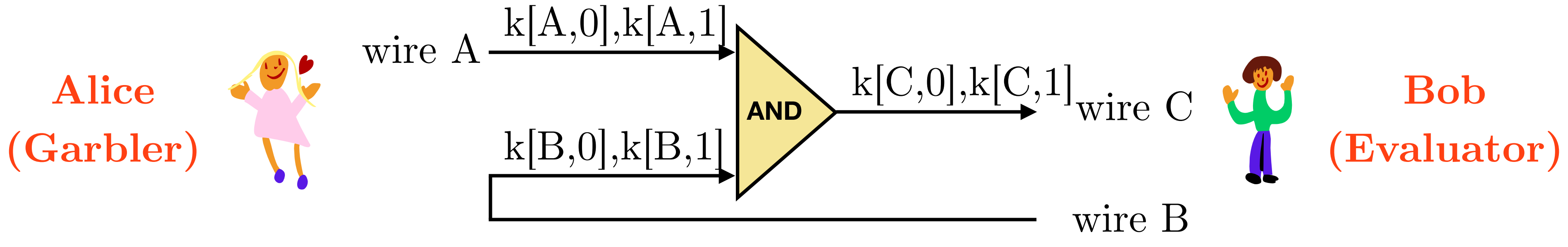
  $$\mathrm{GT}[1,0] = \mathrm{E}_{\mathrm{k}[A,1],\ \mathrm{k}[B,0]}\ (\ \mathrm{k}[C,\mathrm{AND}(1,0)]\ )$$

  $$\mathrm{GT}[1,0] = \mathrm{E}_{\mathrm{k}[A,1],\ \mathrm{k}[B,0]}\ (\ \mathrm{k}[C,\ 0]\ )$$

XOR in a Nutshell

- XOR is my favourite logical gate. There are so multiple ways of understanding it:
- $$\begin{aligned} A \oplus A &= 0 \\ A \oplus \text{not } A &= 1 \\ \text{i.e.}\quad 0 \oplus 0 &= 0 \\ 0 \oplus 1 &= 1 \\ 1 \oplus 0 &= 1 \\ 1 \oplus 1 &= 0 \end{aligned}$$
  0 if same
  1 if different
  Truth table
- $$\begin{aligned} A \oplus 0 &= A \\ A \oplus 1 &= \text{not } A \end{aligned}$$
  0 for Passthrough
  1 for Invert
- $$\begin{aligned} &A \oplus K \\ &A \oplus K \oplus K = A \end{aligned}$$
  Once to encrypt
  Twice to decrypt
- Exercise. Write a sequence of variable assignments to swap the values in two variables A and B without using a 3rd variable, i.e. by using A and B only.

Randomly Permutate GT (Alice)



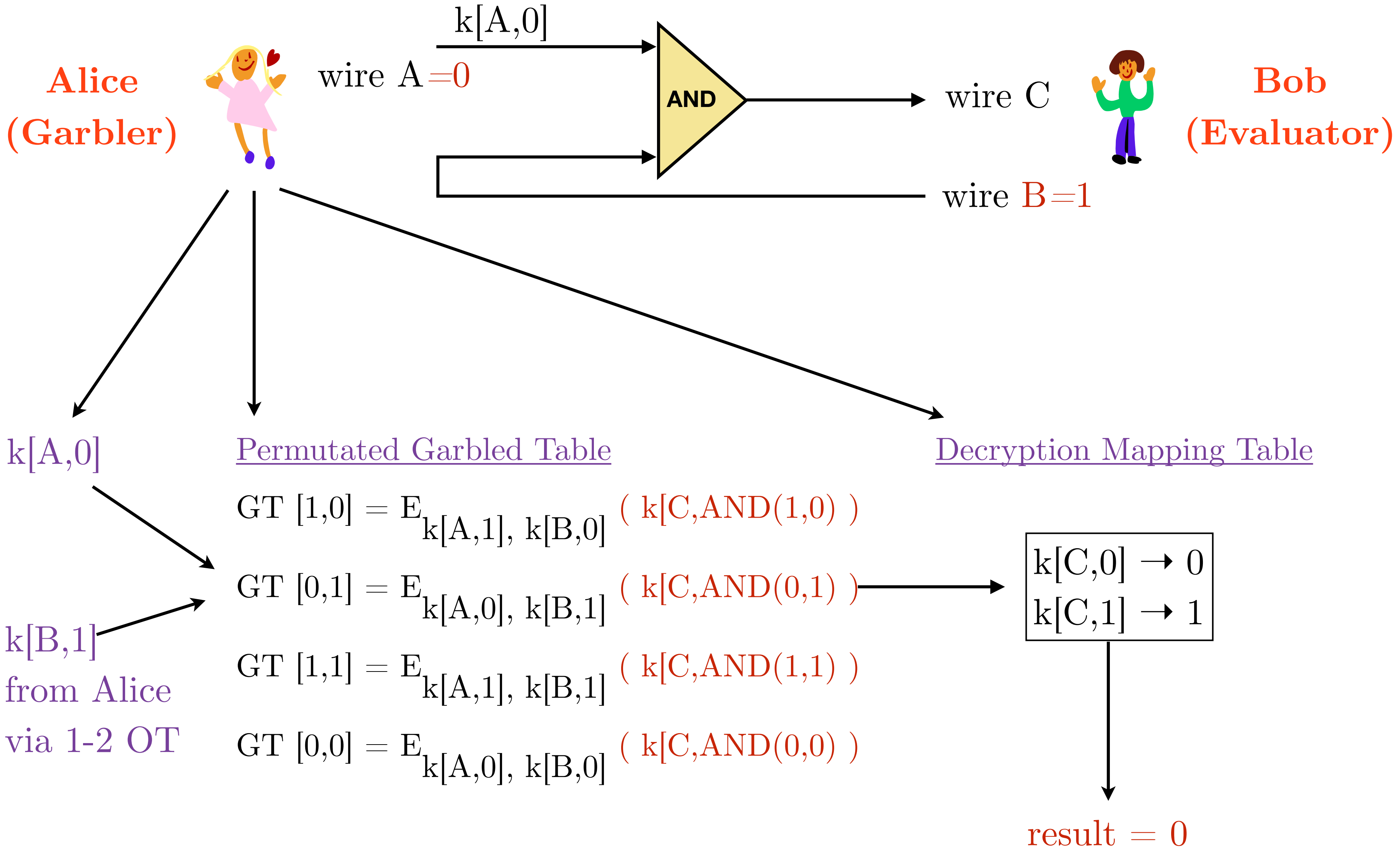
Garbled (Encrypted) Truth Table for AND $\longrightarrow$ Randomly permutated Garbled Table for AND

Garbled (Encrypted) Truth Table for AND	Randomly permutated Garbled Table for AND
GT [0,0] = $E_{k[A,0],k[B,0]}$ (k[C,AND(0,0))	GT [1,0] = $E_{k[A,1],k[B,0]}$ (k[C,AND(1,0))
GT [0,1] = $E_{k[A,0],k[B,1]}$ (k[C,AND(0,1))	GT [0,1] = $E_{k[A,0],k[B,1]}$ (k[C,AND(0,1))
GT [1,0] = $E_{k[A,1],k[B,0]}$ (k[C,AND(1,0))	GT [1,1] = $E_{k[A,1],k[B,1]}$ (k[C,AND(1,1))
GT [1,1] = $E_{k[A,1],k[B,1]}$ (k[C,AND(1,1))	GT [0,0] = $E_{k[A,0],k[B,0]}$ (k[C,AND(0,0))

Notes

- Alice randomly permutates the entries of the Garbled Table, since the truth-table order will leak information that could be used to learn Alice's input.
- Alice will send the permutated garbled gate table to Bob.
- Bob won't know which row of the garbled table corresponds to which row in the original table.
- Alice also sends the key corresponding to her plaintext input bit to Bob.
- For example, if Alice's input bit is 0, Alice will send k[A,0]. Bob will not know that this corresponds to 0 since all wire keys are random.
- Alice also sends a Decryption Mapping Table that maps the keys of the circuit's output wire to their plaintext values i.e. the mapping would be:
  k[C,0] corresponds to 0
  k[C,1] corresponds to 1.

1-gate GC: Evaluation (Bob)



Permutated Garbled Table

$GT\ [1,0] = E_{k[A,1],\ k[B,0]}\ (\ k[C,AND(1,0)\ )$

$GT\ [0,1] = E_{k[A,0],\ k[B,1]}\ (\ k[C,AND(0,1)\ )$

$GT\ [1,1] = E_{k[A,1],\ k[B,1]}\ (\ k[C,AND(1,1)\ )$

$GT\ [0,0] = E_{k[A,0],\ k[B,0]}\ (\ k[C,AND(0,0)\ )$

Decryption Mapping Table

$k[C,0] \rightarrow 0$

$k[C,1] \rightarrow 1$

result = 0

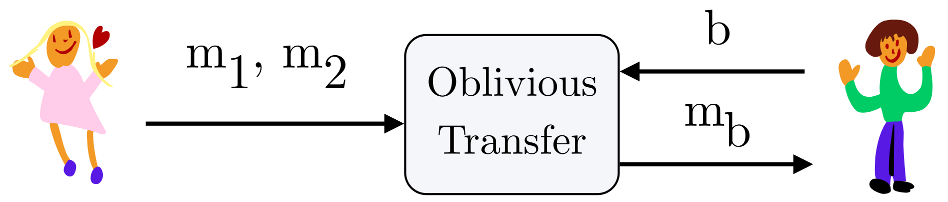
Notes

- Recall Alice sends the following toBob:
  1. The key(s) corresponding to Alice's input(s). There is only one input bit (a). Let's assume that the input is 0, so the key that Alice sends for wire A is k[a,0].
  2. The permutated garbled tables for the circuit. There is only 1 GT for our simple circuit (for AND).
  3. A decryption mapping table mapping the 2 possible keys for output wire c to their plaintext bits..
- Bob first runs a 1-from-2 oblivious transfer (OT) protocol with Alice for each of Bob input bits. This will get the keys that corresponds to Bob's input bits from Alice without Alice learning which keys Bob got! (see 1-from-2 OT later).
- For our example, if Bob's input bit is m. Alice's input to the OT protocol will be the keys Kb0 and Kb1.
- Bob's input to the OT protocol will be m, *suitably protected* from Alice. On completion Alice will send Kbm to Bob
- *Alice does not learn* m and will not know if she sent Kb0 or Kb1 and send it to Bob.
- If Alice had sent Ka0 to Bob and Bob's input (m) was 1 then Bob would doubly decrypt the garbled gate entry for 0,1 first using using Ka0 then using Kb1 to get Kc0
- To complete the evaluation Bob uses the decryption mapping table to see that Kc0 corresponds to 0.
- Bob thus learns the final value and can sent it to Alice.

- For bigger circuits, Alice needs to send all keys for all her inputs and mappings for output wires.
- Alice and Bob must carry out 1-from-2 OTs for each of Bob's input bits. The OTs can be done in parallel.
- In bigger circuits, Bob does not learn any intermediate value, since he is only given the decryption mapping for the circuits output wires, not for any intermediate output wires.
- Of course for a trivial 1-gate AND, if b was 1, and c was 1, then a must be 1 (i.e. Bob learns Alice's input in this case). But if the output was 0 he does not know Alice's input. Similarly for Alice. It always instructive to look at the simplest functions and see what's inferable.
- Note: The keys and garbled tables for a circuit can only be used once. New keys and garbled tables are needed each time the MPC is run.

- *But what is an Oblivious Transfer?*
- *And how does Bob know which row of a permutated garbled table it should it decrypt, since the permutations for each garbled table in the circuit are random?*

1-from-2 Oblivious Transfer protocol



Illustrative example

Alice generates two public-private key pairs (Pub1, Priv1), (Pub2, Priv2)

Alice → Bob:	Pub1, Pub2	Bob generates a symmetric key k and randomly chooses Pub1 say
Bob → Alice:	c=$E_{\text{Pub1}}(k)$	Alice does D_{Priv1}(c)= k Bob's key and D_{Priv2}(c)=u some random value

Alice → Bob:	c1=$E_k(m1)$,
	c2=$E_u(m2)$
		Bob does D_k(c1) = $D_k(E_k(m1))$ = $m1$
		and D_k(c2) = $D_k(E_u(m2))$ = gibberish

Notes

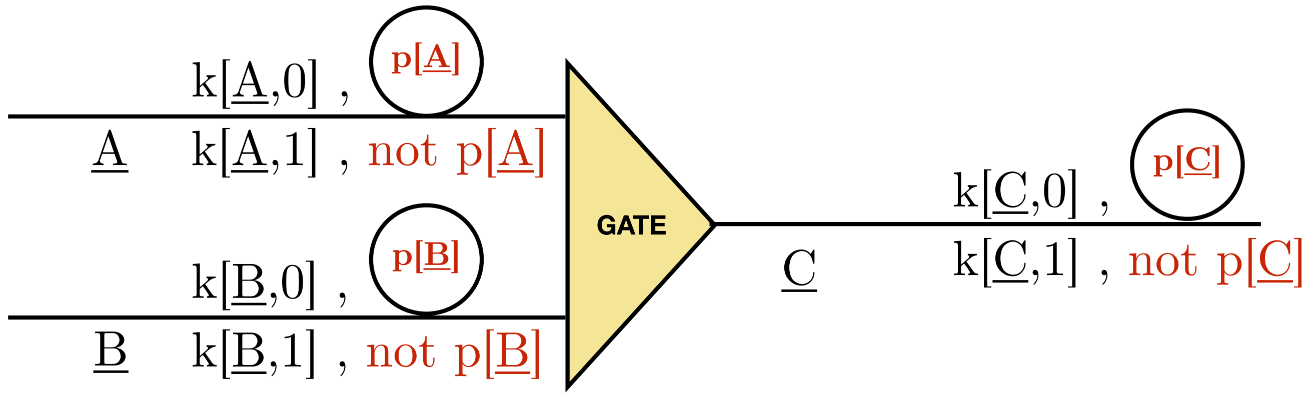
- In a *1–from-2* oblivious transfer protocol, the sender (Alice) has two messages m0 and m1, and the receiver (Bob) has one selection bit b.
- On completion the Bob receives mb, without Alice learning b.
- Alice does not learn which message Bob received.
- Bob does does not learn what the other message was.
- We can extend this to protocols for *1-from-n* oblivious transfers and *k-from-n* oblivious transfers.

- In the illustrative example, Bob learns *m1*.
- Alice doesn't know which message Bob knows.
- Note: in the example Alice's messages need to have a structure that Bob can recognise to distinguish *m1* (the good message) from the gibberish message.
- To prevent Alice from cheating (e.g. using the same message m1=m2), the protocol should require that Alice send Priv1 and Priv2 to Bob for verification on completion. Bob will be able to decrypt the other message though.
- There are better interesting examples in the tutorial.

Point-and-Permutate p-bit

Each wire W also has a **random 0 or 1** value **p[W]** that is used to randomly permutate and index the garbled tables. This ensures that Bob will not need to decrypt all four rows of the garbled table to find the correct row.

For a gate with input wires A, B and output wire C we now have:

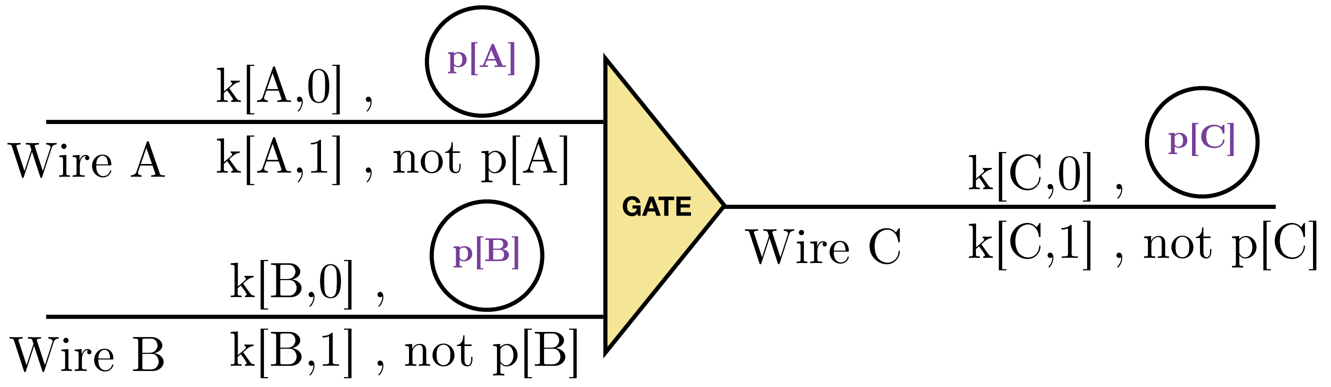


Notes

- Question: how does Bob know which entry of a permutated garbled table should be decrypted, since the permutations for the garbled table is random?
- Bob could decrypt each entry if the underlying encryption method makes it obvious when decrypting which is the "correct" ciphertext. This is both inefficient and inelegant.
- The trick is to pair a random point-and-permutate bit and its inverse (*p-bit*) to each of the keys of the wire (the 0 key and 1 key).
- If a wire A had the keys k[A,0], k[A,1], we would now have (k[A, 0], *p-bit*) and (k[A,1], *not p-bit*).
- Bob will use the *p-bit* elements of the input wire pair to index the garbled table and select the correct keys.

- For example, in order to decrypt a gate with wires A and B and whose p-bit elements are pA and pB, Bob will doubly decrypt the entry at position [pA, pB] with the keys key[wireA, pA] and key[wireB, pB]
- The *p-bits* are sometimes called the colouring-bits or select-bits.

Permutated Garbled Table Construction



[a,b]	x=a ⊕ p[A], y=b ⊕ p[B]		

[0,0]	x=0 ⊕ p[A], y=0 ⊕ p[B]	x=a ⊕ p[A]
		y=b ⊕ p[B]
		z=GATE(x, y)
		z'=z ⊕ p[C]
			$E_{k[A,x],k[B,y]}(k[C,z], z')$

[0,1]	x=0 ⊕ p[A], y=1 ⊕ p[B]		
[1,0]	x=1 ⊕ p[A], y=0 ⊕ p[B]		
[1,1]	x=1 ⊕ p[A], y=1 ⊕ p[B]		

Notes

- The construction of a permutated garbled truth table is little more complicated.
- Essentially for each garbled table entry $[a,b]$, the garbler uses $[x,y]$ instead. $[x,y]$ is used to (i) select the keys to use for double-encryption i.e. key[A,x] and key[B,y], and (ii) as inputs for the gate function i.e. z=GATE(x,y)
- The result of the gate, z, is used to determine the (key, pbit) pair (i.e key[C,z], z xor pbit[C]) to use at the gate that the output wire is connected to.
- Here is the equivalent pseudo-code:

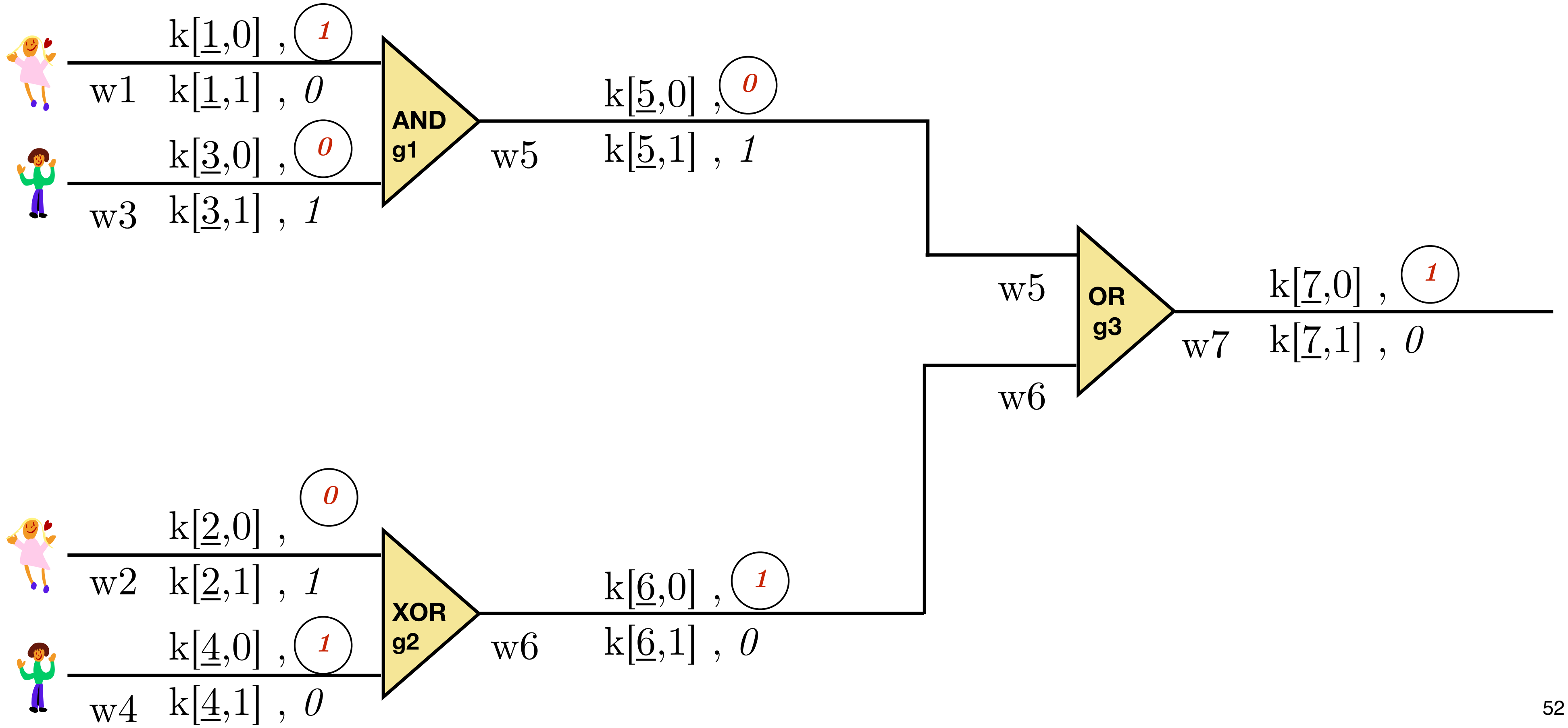
```
foreach permutated PGT with input wires A, B and output wire C
   for a = 0..1
     for b = 0..1      // construct permutated truth table entries
        x  = a xor pbit[A]
        y  = b xor pbit[B]
        z  = GATE (x,y)           // e.g. AND, OR, XOR, NAND ...
        z' = z xor pbit[C]
        PGT[a][b] = ENCRYPT$_{key[A,x],key[B,y]}$( key[C,z], z'] )
```

Example: Point-and-Permutate p-bit

For example, the circuit below is labelled with the following random p-bits:

p[<u>1</u>]=p[<u>4</u>]=p[<u>6</u>]=p[<u>7</u>]= *1*   and p[<u>2</u>]=p[<u>3</u>]=p[<u>5</u>]= *0*

For each wire w, the value in the circle is the p-bit for key 0 (i.e p[w]), the value below is its inverse for key 1 (i.e. **not** p[w]).



Example: Permutated Garbled Tables

Garbled tables for $p[\underline{1}]=p[\underline{4}]=p[\underline{6}]=p[\underline{7}]=\textcircled{\textit{1}}$ and $p[\underline{2}]=p[\underline{3}]=p[\underline{5}]=\textcircled{\textit{0}}$

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

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

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

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

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

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

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

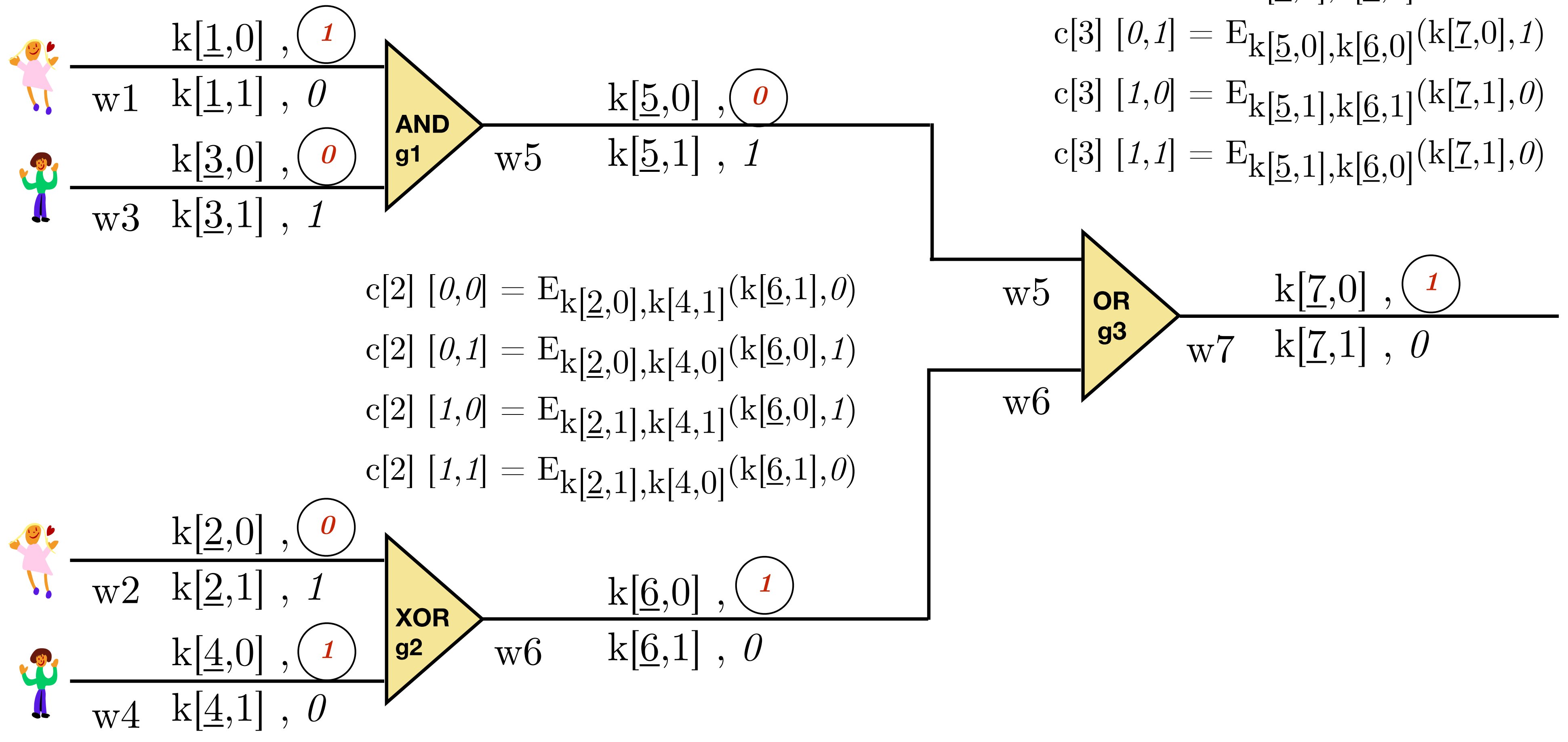
$c[3]\ [\mathit{1},\mathit{1}] = E_{k[\underline{5},1],k[\underline{6},0]}(k[\underline{7},1],\mathit{0})$

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

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

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

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



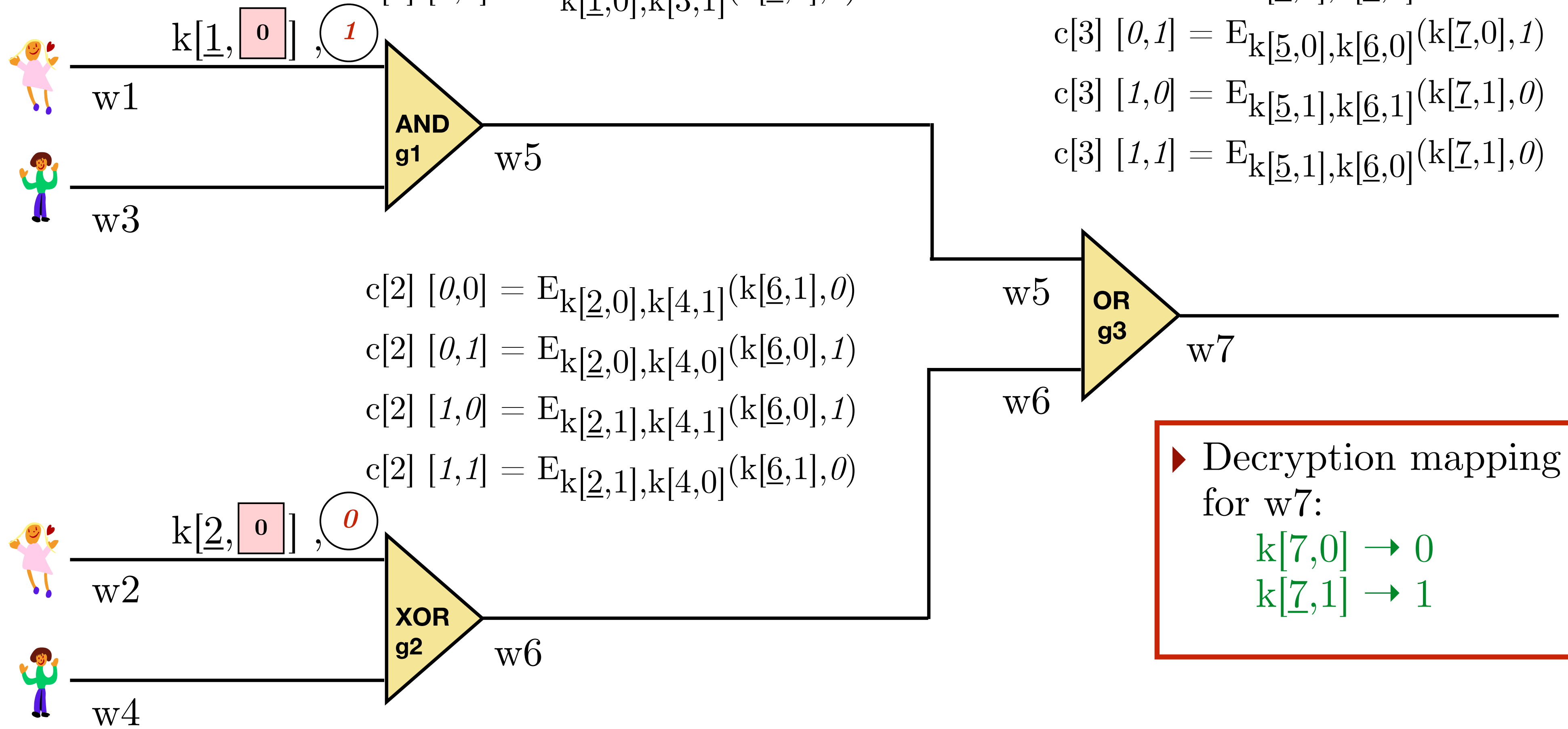
Circuit Transfer (Alice to Bob)

- For plaintext inputs **Alice:** $v[\underline{1}] = \boxed{0}$, $v[\underline{2}] = \boxed{0}$ **Bob:** $v[\underline{3}] = \boxed{1}$, $v[\underline{4}] = \boxed{0}$

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

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

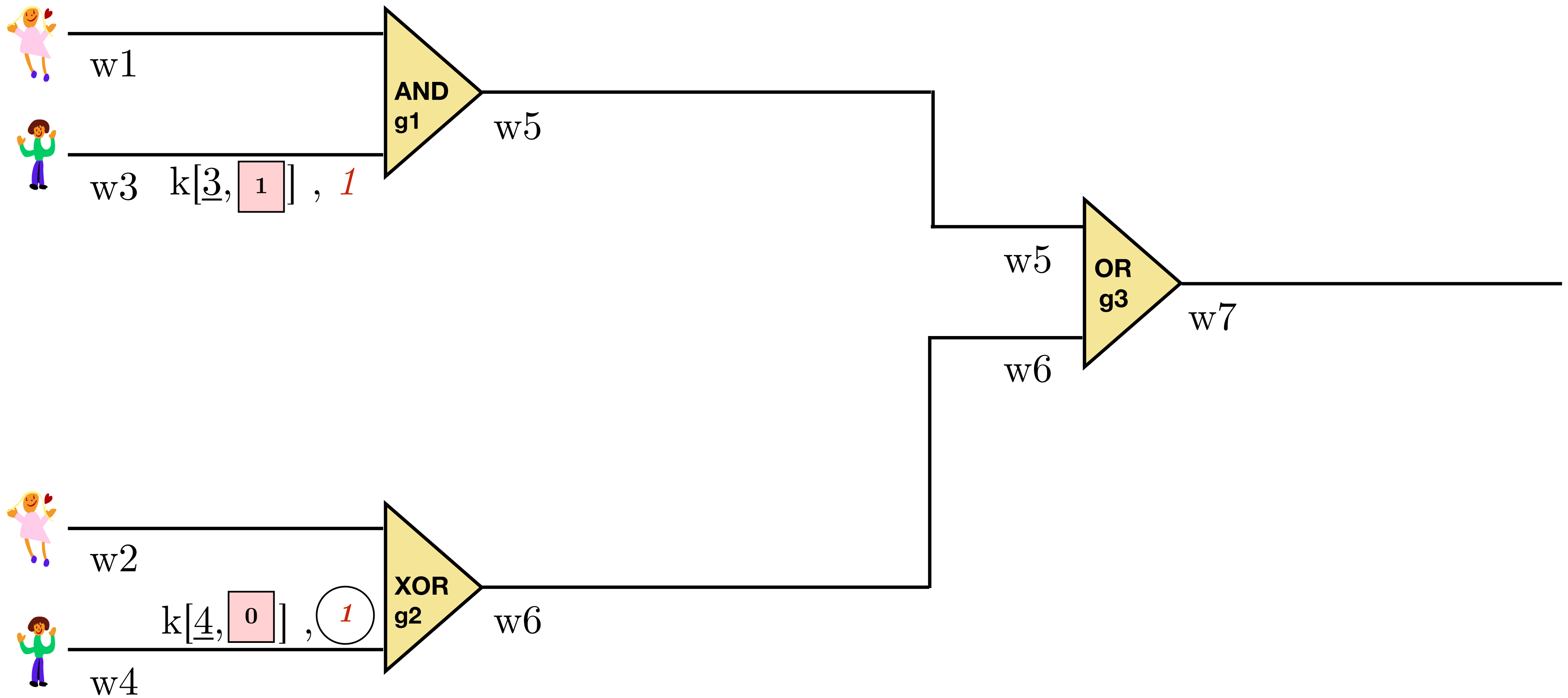
$$c[2]\ [\mathit{0},0] = E_{k[\underline{2},0],k[4,1]}(k[\underline{6},1],\mathit{0})$$
$$c[2]\ [\mathit{0},\mathit{1}] = E_{k[\underline{2},0],k[4,0]}(k[\underline{6},0],\mathit{1})$$
$$c[2]\ [\mathit{1},\mathit{0}] = E_{k[\underline{2},1],k[4,1]}(k[\underline{6},0],\mathit{1})$$
$$c[2]\ [\mathit{1},\mathit{1}]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



- Decryption mapping for w7:
  $k[7,0] \rightarrow 0$
  $k[\underline{7},1] \rightarrow 1$

Oblivious Transfers (Alice and Bob)

- Bob engages in an oblivious transfer with Alice using input v[3]= $\boxed{1}$ to privately select one of Alice's keys/pbits for wire3 i.e. from k[$\underline{3}$,0], 1 and k[$\underline{3}$,1], 1. Bob learns k[$\underline{3}$, $\boxed{1}$] , *1*
- Bob also engages in a 2nd oblivious transfer using input v[4]= $\boxed{0}$ to privately select one of Alice's keys/bits for wire4 i.e. from k[$\underline{4}$,0], 1 and k[$\underline{4}$,1], 0. Bob learns k[$\underline{4}$, $\boxed{0}$] , $\textcircled{\textit{1}}$



Garbled Circuit (known by Bob)

- On completion of the Oblivious Transfers Bob knows:

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

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

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

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

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

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

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

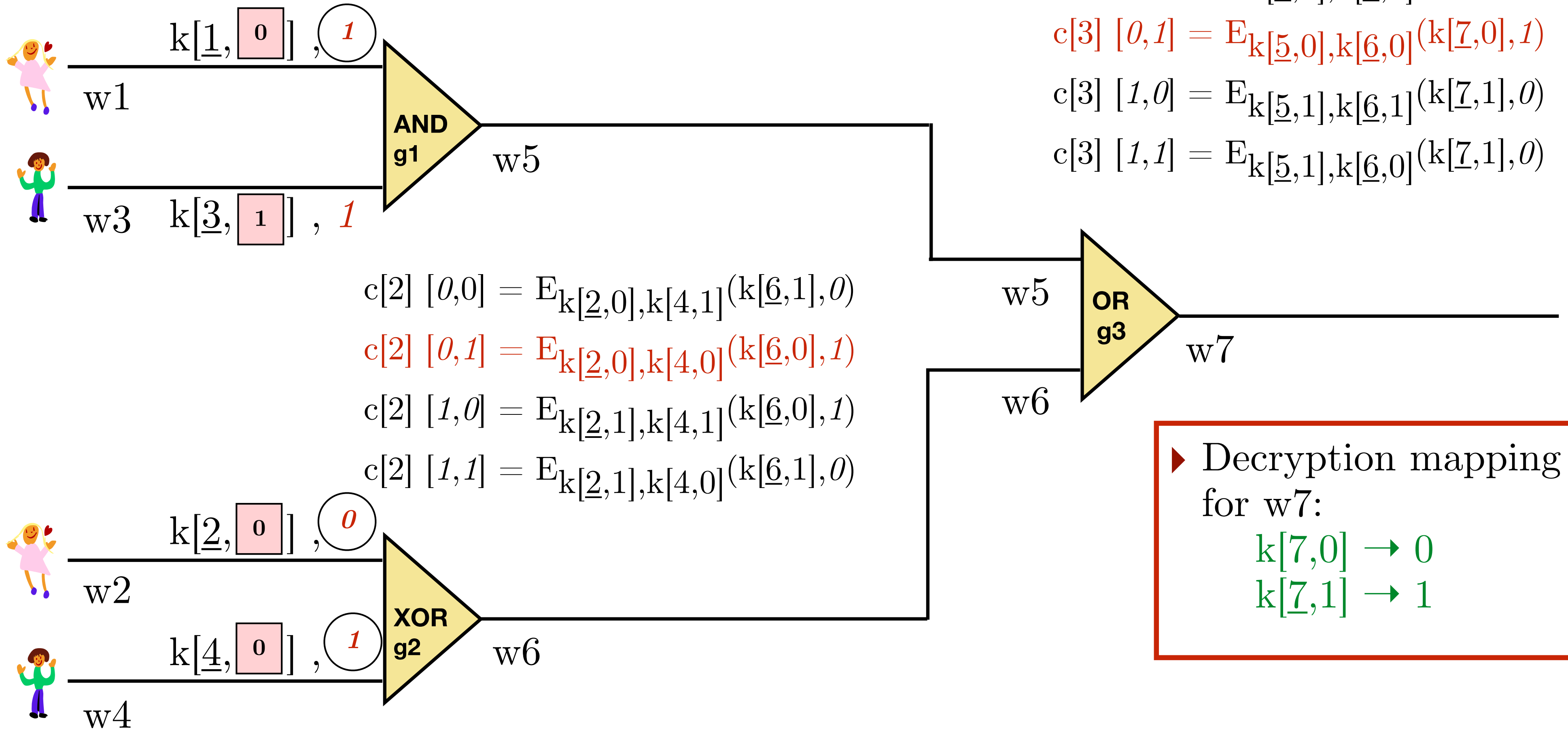
$c[3]\ [\mathit{1},\mathit{1}] = E_{k[\underline{5},1],k[\underline{6},0]}(k[\underline{7},1],\mathit{0})$

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

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

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

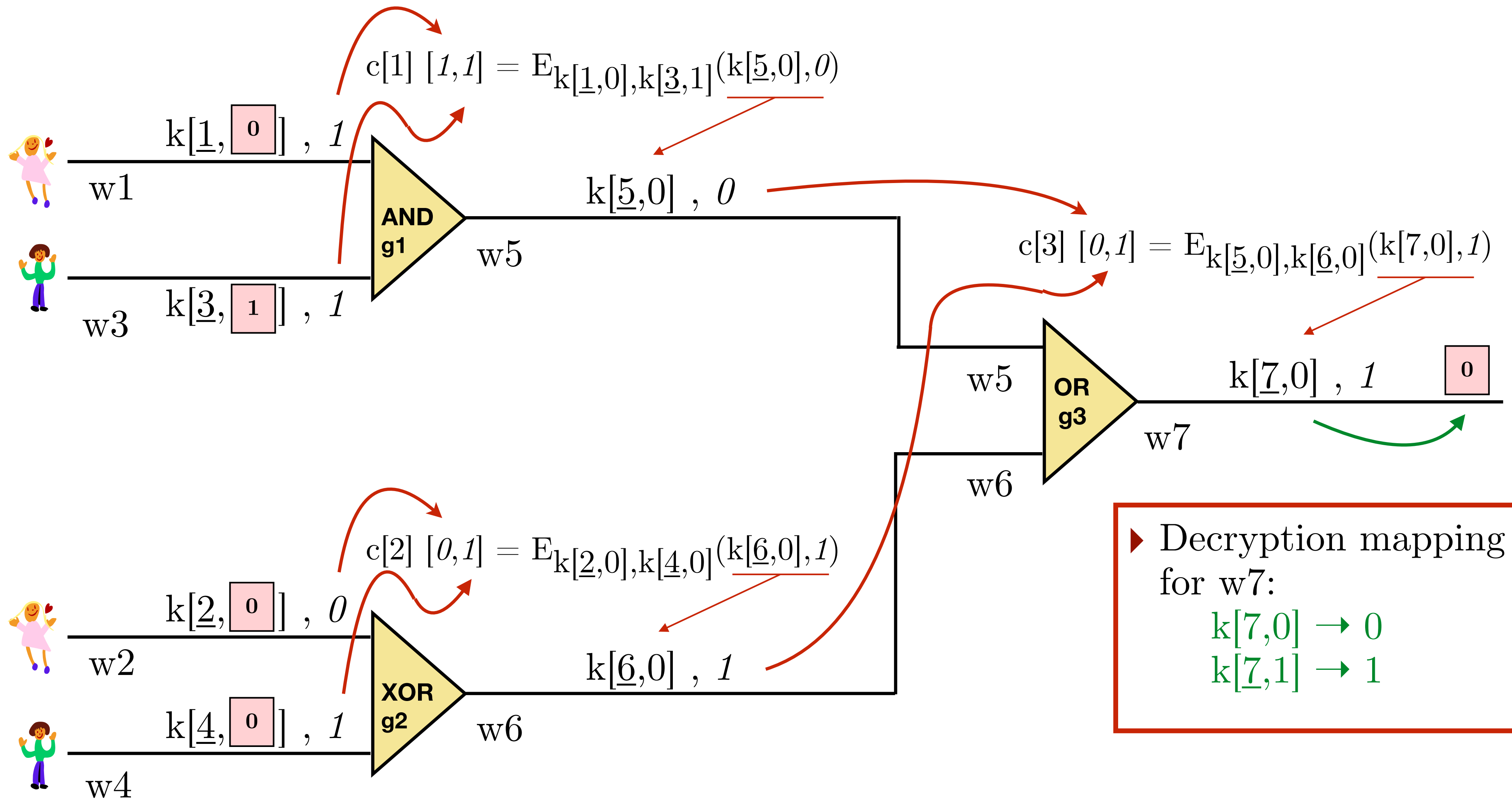
$c[2]\ [\mathit{1},\mathit{1}] = E_{k[\underline{2},1],k[4,0]}(k[\underline{6},1],\mathit{0})$



- Decryption mapping for w7:
  $k[7,0] \rightarrow 0$
  $k[\underline{7},1] \rightarrow 1$

Garbled Circuit Evaluation by Bob

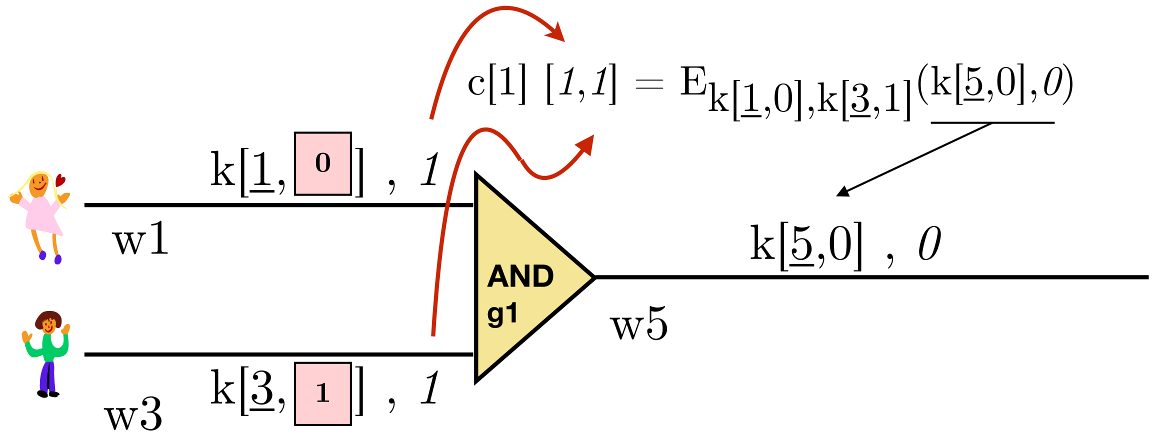
- Plaintext bits **Alice:** v[1]= $\boxed{0}$, v[2]= $\boxed{0}$ **Bob:** inputs v[3]= $\boxed{1}$, v[4]= $\boxed{0}$


c[1] [1,1] = E_k[1,0],k[3,1](k[5,0],0)
k[1, 0] , 1
w1
k[3, 1] , 1
w3
AND g1
k[5,0] , 0
w5
c[3] [0,1] = E_k[5,0],k[6,0](k[7,0],1)
w5
OR g3
w6
k[7,0] , 1
0
w7
c[2] [0,1] = E_k[2,0],k[4,0](k[6,0],1)
k[2, 0] , 0
w2
k[4, 0] , 1
w4
XOR g2
k[6,0] , 1
w6

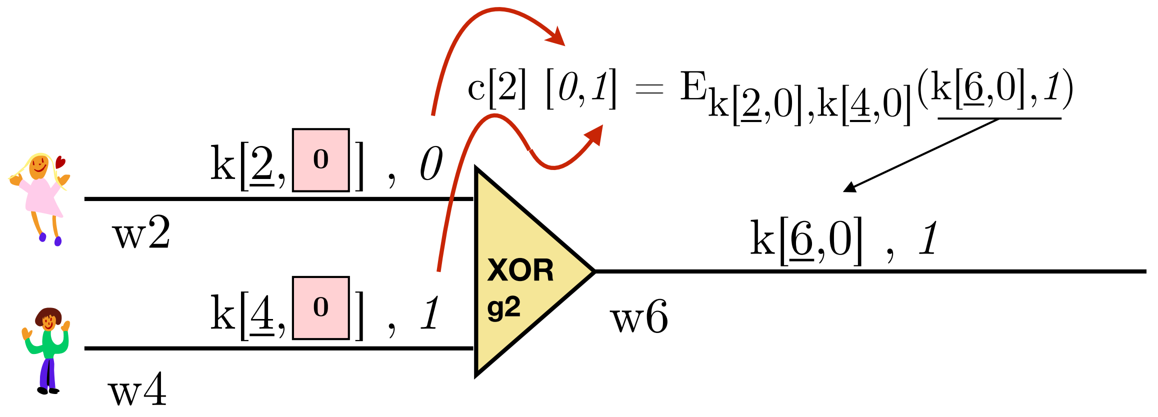
- Decryption mapping for w7:
  - $k[7,0] \rightarrow 0$
  - $k[7,1] \rightarrow 1$

Garbled Circuit Evaluation 2

- For the AND gate Bob sees the random p-bit element for w1 is *1*, and for w3 is *1*. Bob uses these to index the gate's garbled table to get $E_{k[\underline{1},0],k[\underline{3},1]}(k[\underline{5},0],\mathit{0})$ Bob then decrypts this using $k[\underline{1},0]$ and $k[\underline{3},1]$ to get the pair $k[\underline{5},0],\mathit{0}$

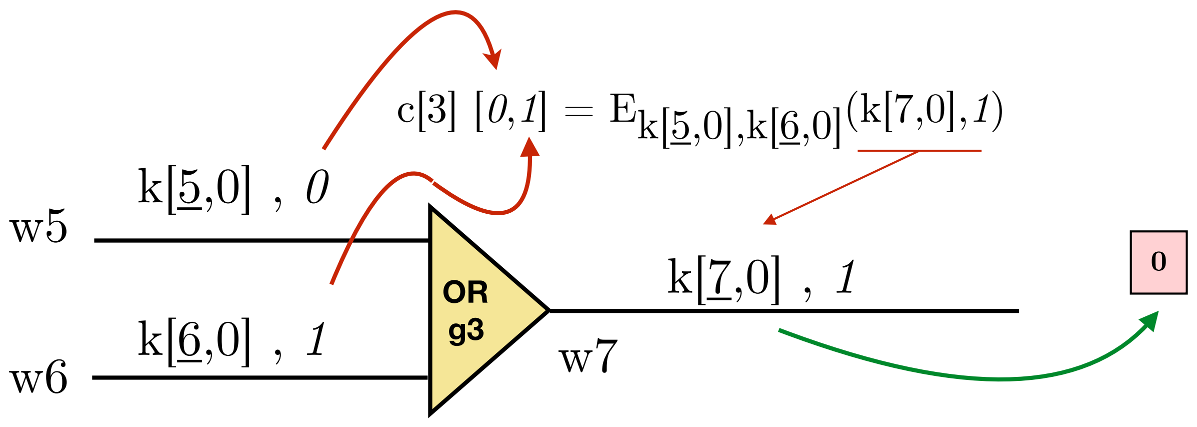


- For the XOR gate Bob sees the random p-bit element for w2 is *0*, and for w4 is *1*. Bob uses these to index the gate's garbled table to get $E_{k[\underline{2},0],k[\underline{4},0]}(k[\underline{6},0],\mathit{1})$ Bob then decrypts this using $k[\underline{2},0]$ and $k[\underline{4},0]$ to get the pair $k[\underline{6},0],\mathit{1}$



Garbled Circuit Evaluation 3

- For the OR gate Bob sees decrypted p-bit element for wire w5 is *0*, and for w6 is *1*. Bob uses these to index the gate's garbled table to get $E_{k[\underline{5},0],k[\underline{6},0]}(k[7,0],\mathit{1})$
  Bob then decrypts this using $k[\underline{5},0]$ and $k[\underline{6},0]$ to get the pair $k[\underline{7},0],\mathit{1}$
- Finally, Bob uses the decryption mapping to determine that k[7,0] corresponds to 0.
- So result of the circuit is 0


$c[3]\ [\mathit{0},\mathit{1}] = E_{k[\underline{5},0],k[\underline{6},0]}(\underline{k[7,0],\mathit{1}})$
w5
$k[\underline{5},0]\ ,\ \mathit{0}$
w6
$k[\underline{6},0]\ ,\ \mathit{1}$
OR
g3
$k[\underline{7},0]\ ,\ \mathit{1}$
w7
0

- Double check: f({w1,w2},{w3, w4}) = (w1 AND w3) OR (w2 XOR w4)
  f({0,0}, {1,0}) =(0 AND 1) OR (0 XOR 0) = 0 OR 0 = 0

Garbled Circuit Performance

- Runs in a constant number of rounds.
- Computation cost dominated by encryption function used. Typically hardware-assisted AES is used.
- High communication costs - time to transfer circuit, plus time to complete oblivious transfers (later can be done in parallel).

Example performance circa 2017

- Evaluation speeds in excess of 10M gates per second per core, for CPUs with AES hardware.
- Circuit size: ~22M gates for 16x16 floating-point matrix multiply.

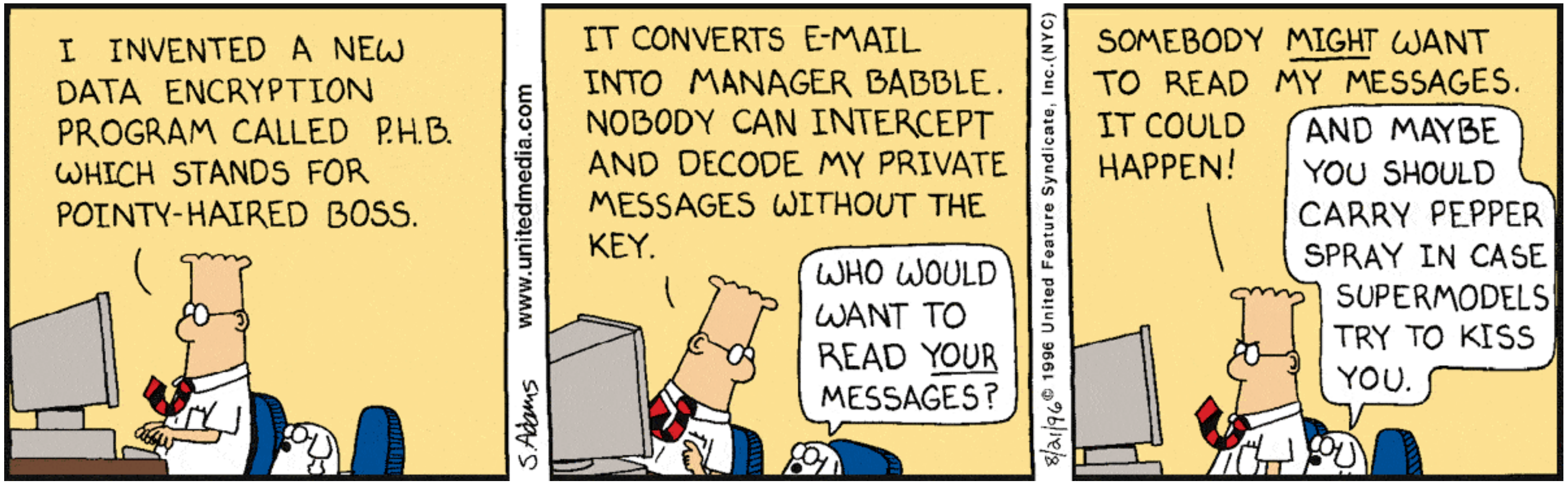
Compilation

- Circuit compilation techniques are similar to hardware synthesis for combinatorial logic.
- There are many compiler optimisations that are used: point-and-permutate, row-reduction, free-XORs, half-gates, pipelining, fixed key garbling. Compilers include - Fairplay, TinyGarble, Frigate, ObliVM, Obilv-C, CBM-GC, PCF.

Malicious Adversaries

- Both Alice and Bob could cheat at various stages. Techniques such as *zero-knowledge proofs* and *cut-and-choose* can be used but have high overheads.

Manager Babble



I INVENTED A NEW DATA ENCRYPTION PROGRAM CALLED P.H.B. WHICH STANDS FOR POINTY-HAIRED BOSS.
www.unitedmedia.com
S. Adams
IT CONVERTS E-MAIL INTO MANAGER BABBLE. NOBODY CAN INTERCEPT AND DECODE MY PRIVATE MESSAGES WITHOUT THE KEY.
WHO WOULD WANT TO READ YOUR MESSAGES?
8/21/96 © 1996 United Feature Syndicate, Inc.(NYC)
SOMEBODY MIGHT WANT TO READ MY MESSAGES. IT COULD HAPPEN!
AND MAYBE YOU SHOULD CARRY PEPPER SPRAY IN CASE SUPERMODELS TRY TO KISS YOU.

3. Fully Homomorphic Encryption (FHE)

Naranker Dulay

n.dulay@imperial.ac.uk

https://www.doc.ic.ac.uk/~nd/peng

November 2020 (v1)

Fully Homomorphic Encryption (FHE)

Computations on encrypted data!

Informally, given ciphertexts ***C1*** to ***Cn*** for plaintexts ***P1*** to ***Pn.***

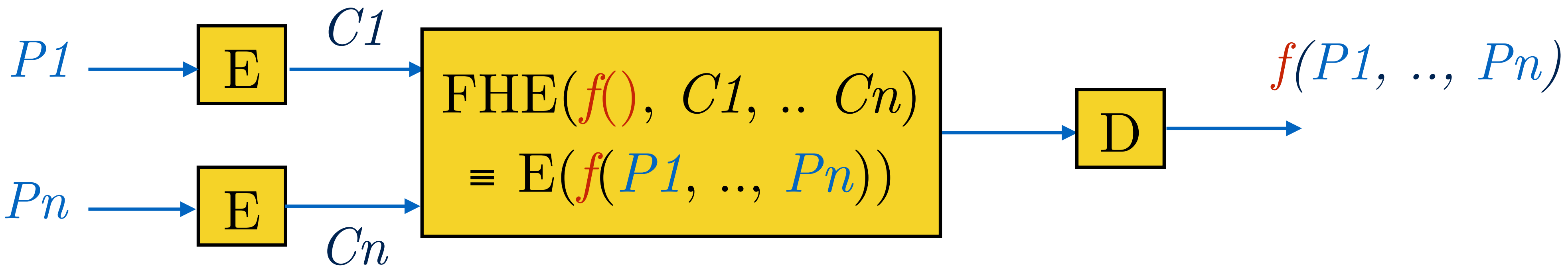
FHE allows anyone (e.g. untrusted cloud providers), not just the key holder, to output a ciphertext with the encrypted value of ***f(P1,.. Pn)*** for **ANY FUNCTION *f*.**

Inputs, outputs and intermediate values in FHE are always encrypted. No information about ***P1***, ***Pn*** or ***f*** is leaked.

Wow!

- The idea of FHE was first postulated *Ron Rivest* and *Leonard Adelman* in 1978, a few months after the RSA paper.
- What uses can you think of for FHE if it was available?

Fully Homomorphic Encryption (FHE)



Encryption (Public)/ Decryption (Private) key(s) omitted

Partially Homomorphic Encryption

- Similar to FHE but for a specific / known function. Can be very efficient.
- For example, here's a totally insecure homomorphic scheme (to get the idea):

1. to multiply 2 positive real numbers: $\boldsymbol{z = x \times y}$
2. take their logs (E): $\mathbf{log}(\boldsymbol{x}), \mathbf{log}(\boldsymbol{y})$
3. add their logs (FHE): $\mathbf{log}(\boldsymbol{x}) + \mathbf{log}(\boldsymbol{y}) = \mathbf{log}(\boldsymbol{z})$
4. take the anti-log (D): $\boldsymbol{z} = \mathbf{antilog}(\mathbf{log}(\boldsymbol{z}))$

- RSA is a better example for multiplication. The homomorphic property is

$$\mathbf{E}(\boldsymbol{x}) \times \mathbf{E}(\boldsymbol{y}) = \boldsymbol{x}^e \times \boldsymbol{y}^e \ \mathbf{mod\ m} = (\boldsymbol{xy})^e \ \mathbf{mod\ m} = \mathbf{E}(\boldsymbol{x} \times \boldsymbol{y})$$

- Other examples of partially homomorphic schemes include ElGamal for integer multiplication, Paillier for integer addition, ...

Fully Homomorphic Encryption?

But is fully homomorphic encryption possible? Supports a Turing complete set of operations (additions and multiplications)

- Yes !
- The **first** fully homomorphic scheme was devised in 2009 by *Craig Gentry* for his PhD (31 years after it was first postulated).
- Gentry's scheme is totally impractical but very cool and theoretically important.
- There have been many proposals since. State-of-the-Art FHE computations are x1000's slower than MPC schemes (Secret sharing / GCs).

Gentry's FHE: A very informal explanation 1

Gentry's scheme uses **lattice-based cryptography** and supports both *ciphertext addition* and *ciphertext multiplication* which are then used to define any function.

FHE (Circuit, C1, .., Cn)

1. **Circuit** represents a boolean circuit for the function that we want to evaluate. Boolean circuits can be easily changed to use addition and multiplication gates.

2. The circuit can *expand at runtime*, it's not static as in Yao's circuits

3. **Encryption is probabilistic. Each ciphertext value has a little random noise.** However decryption results in the correct plaintext value.

Gentry's FHE: A very informal explanation 1

We'll use the following notation to show ciphertext values as points on the real number line. ***This is only an analogy to help illustrate the idea.***

```
C1   C1    C2    C2
@____?     ?_____@
```

@ is the true value, ? the noisy value

- **Addition under Gentry's FHE '*doubles*' the noise.**

```
                      C3 = C1 + C2
C1   C1      C2    C2        C3         C3
@____?       ?_____@         @__________?
```

- **Multiplication under Gentry's FHE '*squares*' the noise.**

```
                      C3 = C1 x C2
C1   C1      C2    C2                  C3                      C3
@____?       ?_____@                   ?_______________________@
```

- **As more operations are performed, noise (i.e. errors) will grow leading to wrong results**

Gentry's FHE: A very informal explanation 2

In order to control the noise, Gentry resets ciphertext values if they get too noisy (reach a noise threshold). **But how can the ciphertext be reset?**

By first decrypting a noisy ciphertext value (i.e. to get an accurate value) and then re-encrypting it so that it has less noise!

For example, if C3 after multiplication was too noisy

C3 C3

?_____________________@

decrypting then re-encrypting C3 would become

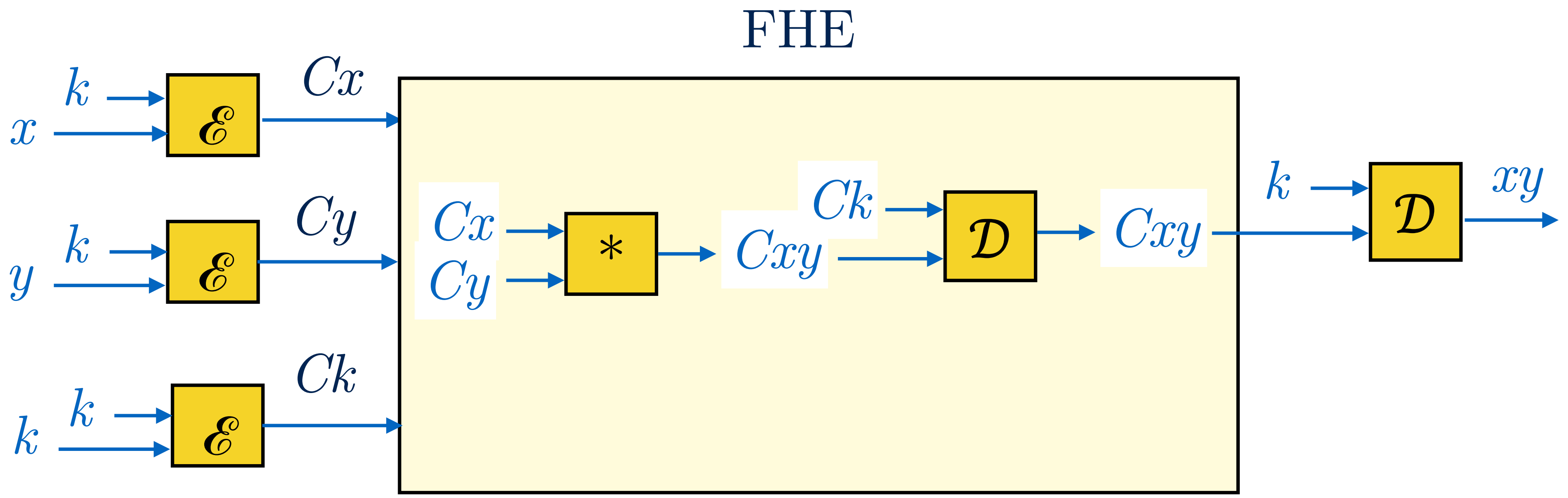
C3 C3

@__?

- **But decryption needs a secret decryption key!** And the whole point of FHE is not to give FHE (e.g. the evaluator) secret keys!
  What did Gentry do?

Gentry's FHE: A very informal explanation 3

- Recall FHE is capable of executing **ANY function** so why not *decrypt* also?
- So Gentry encrypts the secret key (with itself) and passes it to FHE so that the noise reset is done by decrypting with the encrypted key within FHE!!!
- i.e. when *decrypt* is executed within FHE it produces a new encryption of the ciphertext but with only a small amount of noise. Wow!!
- For example, if FHE did a multiply and the result was too noisy, we can reset with:

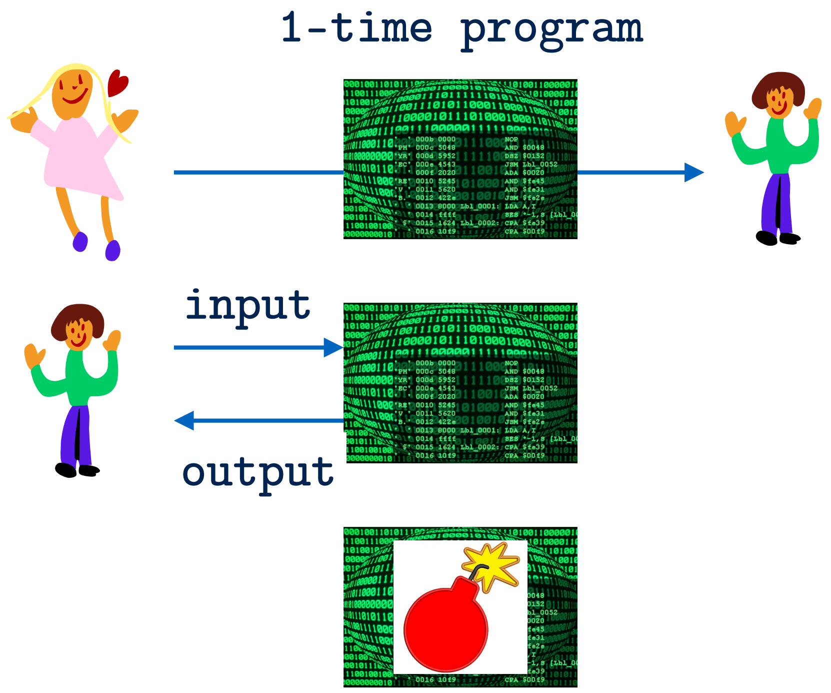


MPC vs FHE

	MPC	FHE
Computation Overhead	Low	High
Communication Overhead	High	Low
Practical?	Increasing no. of applications	Mostly impractical.

“https://en.wikipedia.org/wiki/Homomorphic_encryption” has a nice non-technical survey of the evolution of FHE

One-Time Programs

- Goldwasser, Kalai and Rothblum (GKR) proposed a new type of computer program called a **1-time program,** that *takes some input, executes it, and then self-destructs!*
- 

  Only the result of the program is disclosed, **nothing about the program's implementation**.
- We can extend the idea to ***k-time*** programs.
- What applications/uses can you think of for 1-time or k-time programs?

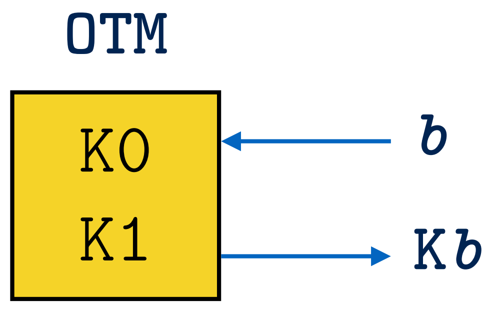
One-Time Programs

- It's not possible to do this with software only since it's easy to copy and re-run software.
- GKR proposed interfacing 1-time programs with a simple but ingenious tamperproof device (see next slide)
- And went on to show that for every input length, any standard program (i.e. **any Turing machine**) can be efficiently compiled into an equivalent **1-time program**.
- They also showed that they could also construct 1-time zero-knowledge proofs - *efficiently convert a classical witness for an NP statement into a 1-time zero-knowledge proof for the statement.*

One-Time Memory (OTM)

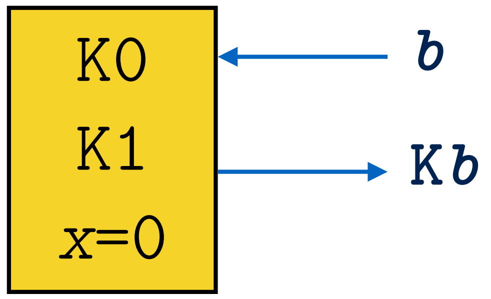
- Does not perform any computation.
- Tamperproof and can withstand ***side-channel attacks*.**
- (1) Locations that are never accessed are never leaked by a side-channel
  (2) Locations that are accessed are immediately leaked
  (3) Also includes a single tamper-proof bit
- More specifically, GKR proposed OTM inspired by **1-from-2 Oblivious Transfers!**

<u>Version 1</u>



- Initialise OTM with two keys K0, K1.
  Accepts single bit input b.
  Outputs Kb and then erases K0 and K1
  Erasing K0,K1 might leak by a side-channel.

<u>Version 2</u>

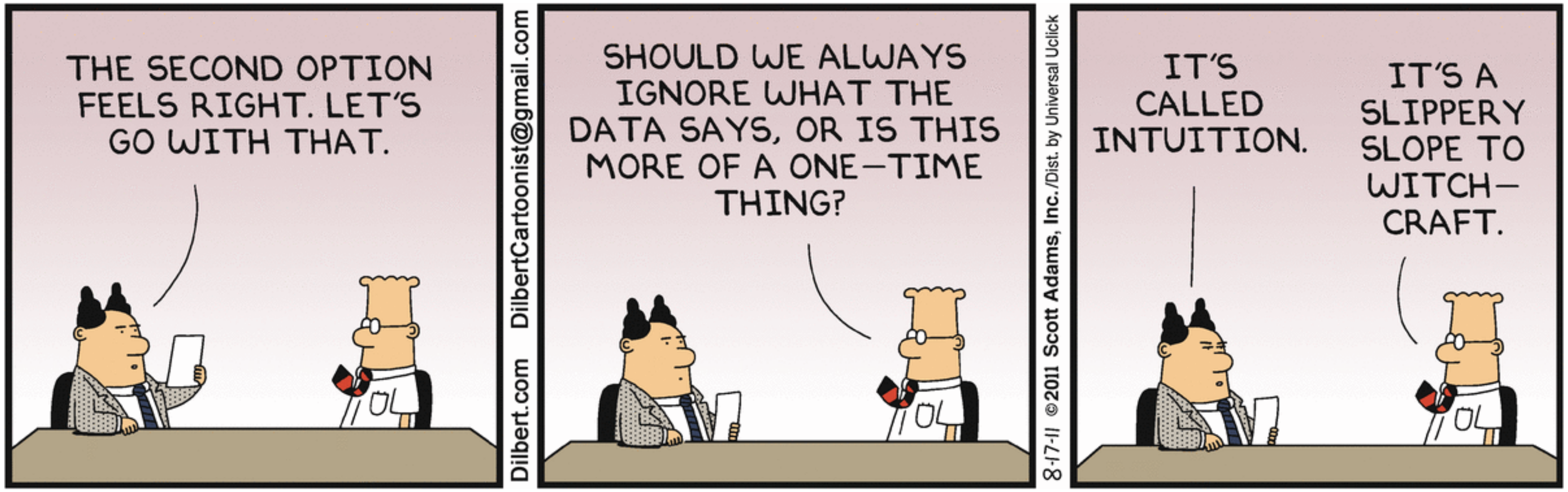


- Initialise OTM with two keys K0, K1 and a tamperproof bit x set to 0.
  if x=0 **then** {set x to 1, accept b, output Kb}
  elif x=1 **then** outputs error
  K$_{1\text{-}b}$ is never accessed in this version.

One-Time Program Compiler

- But how do we construct a 1-time program from a standard program?
- **By using Yao's Garbled circuits**!!
- (1) Convert program into a boolean (logic) circuit (on inputs of length n)
  (2) Garble
  (3) Use n OTMs and put the i-th key pair in the i-th OTM.
  (4) Retrieve keys from OTMs during evaluation of circuit.
- Unfortunately Yao's construction is only secure for ***honest-but-curious adversaries*** so we need to handle ***malicious adversaries*** for example, an adversary could be adaptive in the choice of its inputs which may depend on the garbling itself or the revealed keys. The issue is that Yao's construction requires the input to be known in order to evaluate the circuit.
- To resolve this and make the system handle malicious adversaries, GKR add a mechanism to **delay the choosing of outputs until a malicious adversary specifies inputs,** essentially adds a random bit that performs XORs on outputs only once the adversary specifies inputs (see paper for further details).

Intuition


THE SECOND OPTION FEELS RIGHT. LET'S GO WITH THAT.
Dilbert.com DilbertCartoonist@gmail.com
SHOULD WE ALWAYS IGNORE WHAT THE DATA SAYS, OR IS THIS MORE OF A ONE-TIME THING?
8-17-11 ©2011 Scott Adams, Inc./Dist. by Universal Uclick
IT'S CALLED INTUITION.
IT'S A SLIPPERY SLOPE TO WITCH-CRAFT.