Secure ("Private") Multi-Party Computation

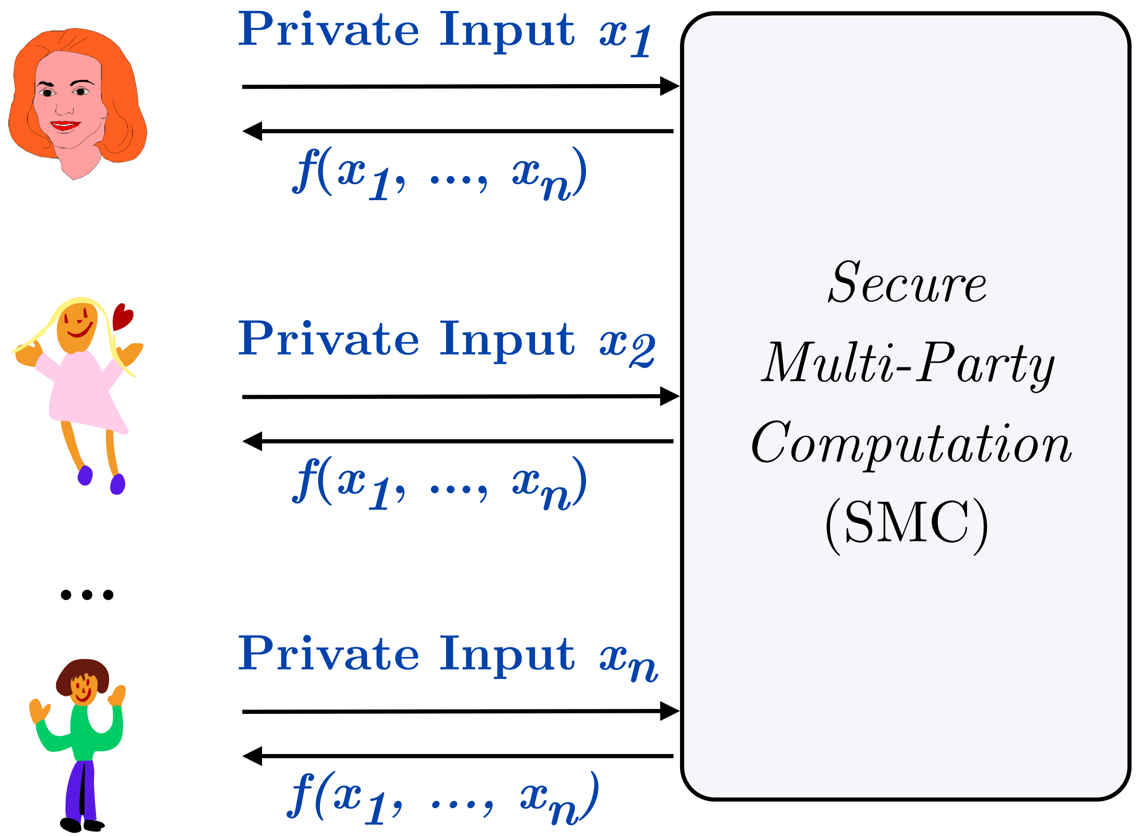
Introduction

Naranker Dulay

n.dulay@imperial.ac.uk

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

Overview



- Given n parties, we wish to design a protocol which computes a function f on their input's such that their inputs remain private **unless revealed by the function, e.g. f(x, y)=x+y)**
- f could be different for each party, but is typically the same.

- SMC is used when parties don't trust either each other **or a 3rd party.**
- Results must be **equivalent to using a trusted 3rd party**. The formal security properties of SMC protocols are typically based on this.

Example: Secure Average Salary?

- A group of people want to calculate their average salary without anyone learning the salary of anyone else.

$E_X(M)$. Encrypt M with X's public key.
$D_X(M)$. Decrypt M with X's private key.

Salaries of Alice, Bob, Carol, Dave are a, b, c, d

Alice generates a large secret random number r and then initiates the protocol:

Alice → Bob:	$E_{Bob}(a+r)$	Bob decrypts to get $a+r$
Bob → Carol:	$E_{Carol}(a+r+b)$	Carol decrypts to get $a+r+b$
Carol → Dave:	$E_{Dave}(a+r+b+c)$	Dave decrypts to get $a+r+b+c$
Dave → Alice:	$E_{Alice}(a+r+b+c+d)$	Alice decrypts to get $a+r+b+c+d$

Alice subtracts r to get total $a+b+c+d$, divides by 4 and informs everyone.

- **Identify 3 or more situations where this protocol fails?**

Example: Secure Average Salary problems

- If anyone lies, the answer will be wrong.
- For two parties each will know others salary, so we need at least 3 parties.
- For 3 parties, if 2 collude, they can work out salary of third party.
- More generally if the two parties before and after another collude they can deduce the result of middle party.
- Alice will know average before others and could lie (or not transmit answer).
- Anyone could increase/decrease value being passed around.
- If all salaries are zero, they will know each other's result.
- If one party lies and enters 0, liar will know the average of others, but others will not know salary of liar.
- What if parties know each other, and everyone knows that Alice has a high earning job. Should she take part in the protocol?

- *Others problems?*

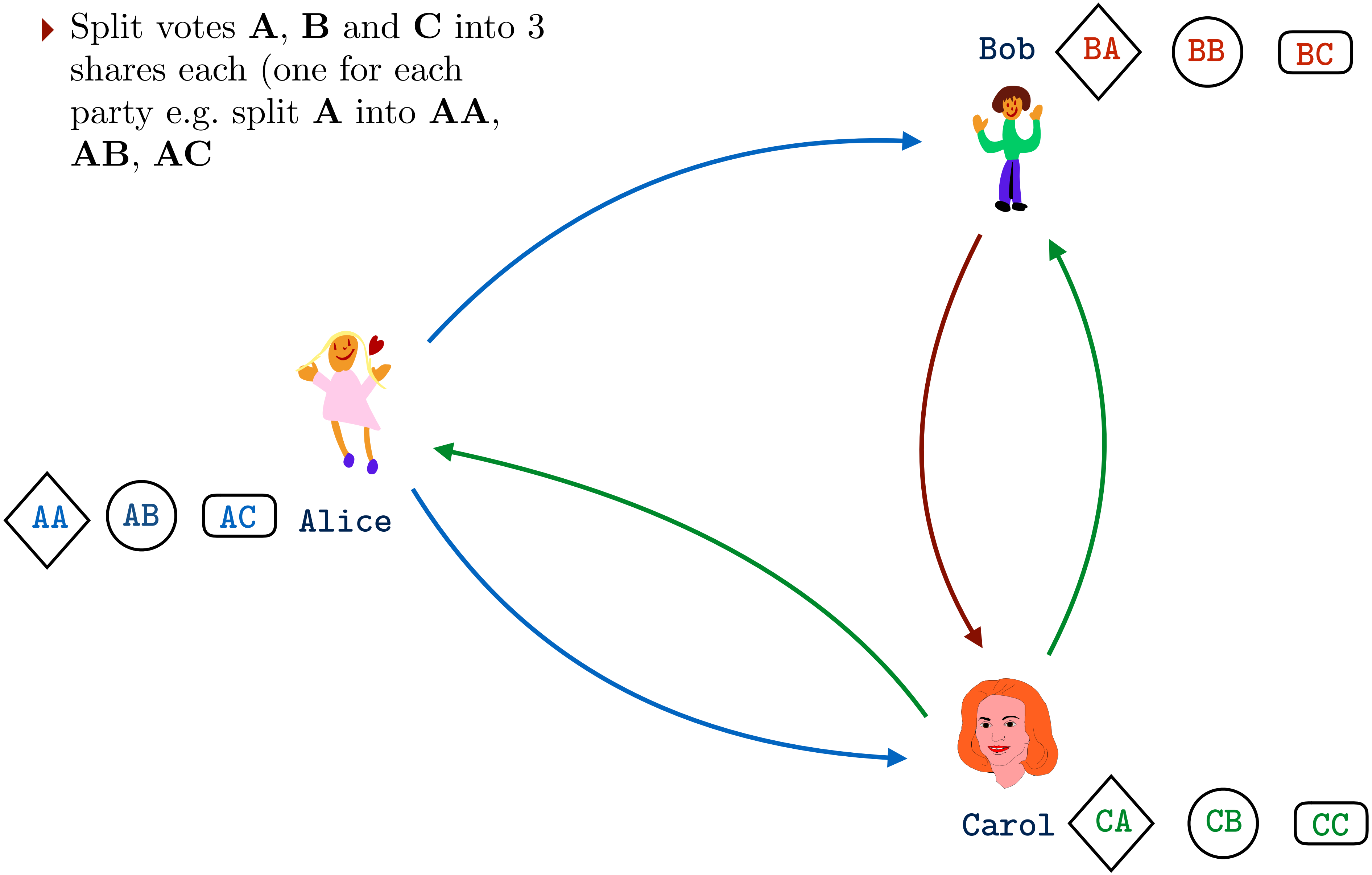
Example: Simple "Secure" Voting

- We want to compute the number of yes votes (yes=1, no=0) for have 3 voters Alice, Bob and Carol with votes A, B and C respectively.

- Alice generates 2 random numbers AB and AC in the range 0..p (p is a large prime agreed in advance). Alice computes $AA{=}A$ - AB - AC *mod p*
i.e. $A{=}A\underline{A}{+}A\underline{B}{+}A\underline{C}$ *mod p* XY values are the **shares** of A (the **secret**)
- Alice sends $(A\underline{A},\ A\underline{C})$ to B, and sends $(A\underline{A},\ A\underline{B})$ to C.
- Bob and Carol generate shares and distribute them in a similar fashion.
- Alice now knows $(AA,\ AB,\ AC)$, $(BB,\ BC)$, $(CC,\ CB)$
Bob now knows $(BB,\ BA,\ BC)$, $(AA,\ AC)$, $(CC,\ CA)$
Carol now knows $(CC,\ CA,\ CB)$, $(AA,\ AB)$, $(BB,\ BA)$
- Alice sums shares for Bob and Carol and broadcasts
$\mathrm{sB} = A\underline{B} + B\underline{B} + C\underline{B}$ mod p and $\mathrm{sC} = A\underline{C} + B\underline{C} + C\underline{C}$ mod p
Bob sums shares for Alice and Carol and broadcasts
$\mathrm{sA} = A\underline{A} + B\underline{A} + C\underline{A}$ mod p and $\mathrm{sC} = A\underline{C} + B\underline{C} + C\underline{C}$ mod p
Carol sums shares for Alice and Bob and broadcasts
$\mathrm{sA} = A\underline{A} + B\underline{A} + C\underline{A}$ mod p and $\mathrm{sB} = A\underline{B} + B\underline{B} + C\underline{B}$ mod p
- Everyone computes final tally Votes = sA + sB + sC mod p.

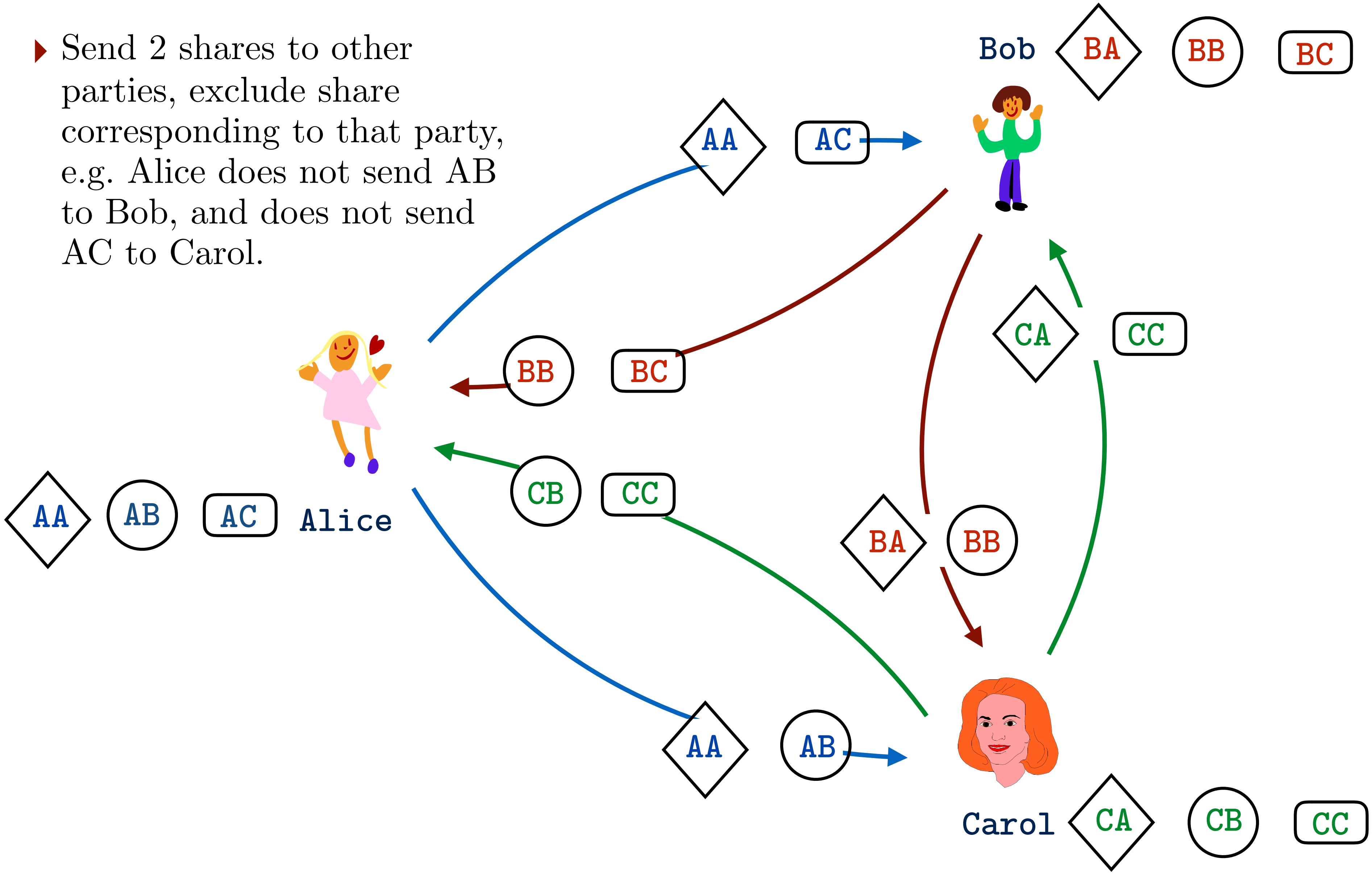
Example: Vote to Shares

- Split votes **A**, **B** and **C** into 3 shares each (one for each party e.g. split **A** into **AA**, **AB**, **AC**

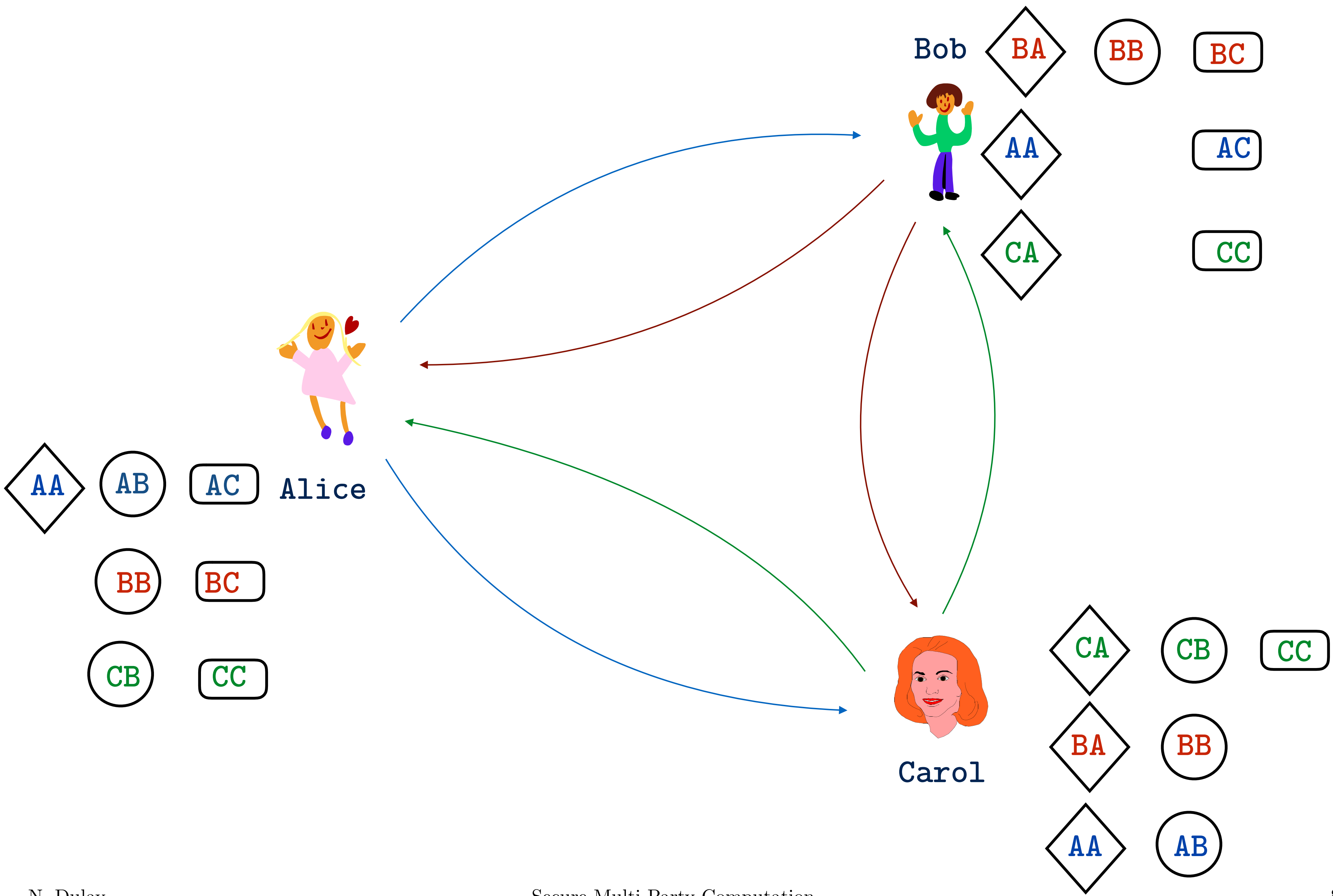


Example: Share Exchange

- Send 2 shares to other parties, exclude share corresponding to that party, e.g. Alice does not send AB to Bob, and does not send AC to Carol.

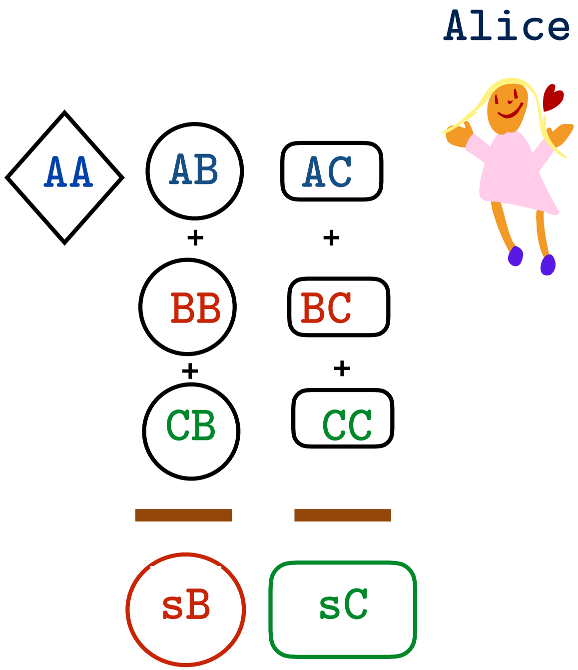


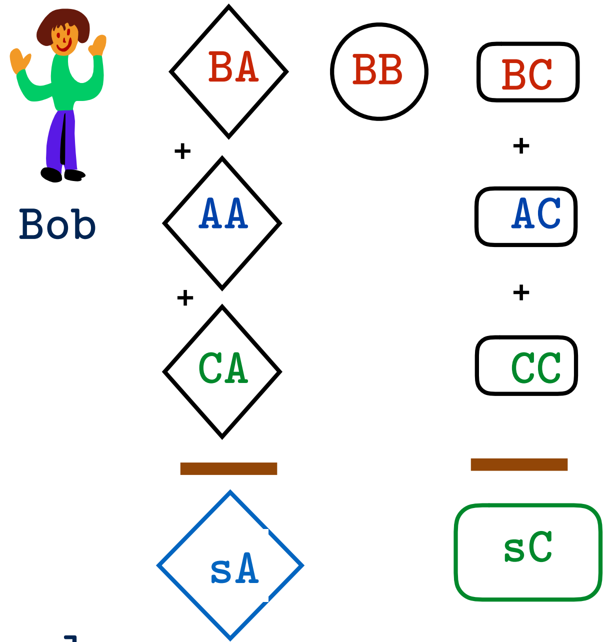
Example: Share Exchange 2

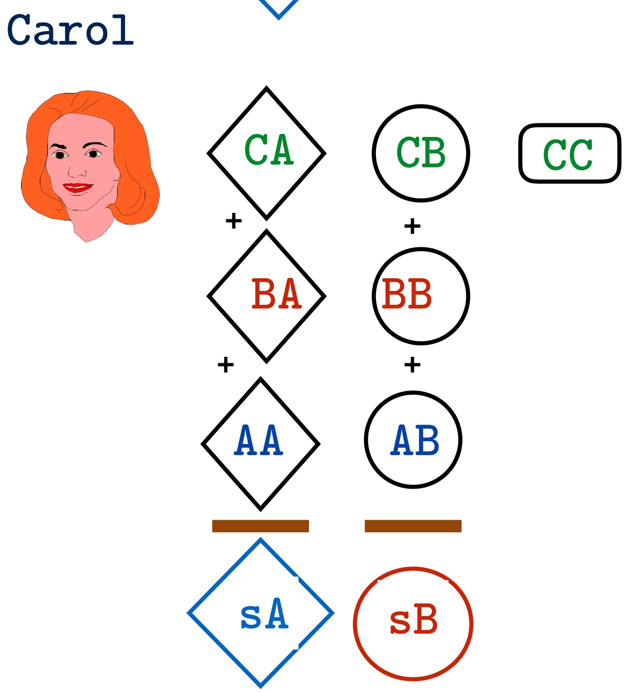


Example: Sum shares for each party

- Sum shares for other parties, e.g. Alice produces sum **sB** for Bob and sum **sC** for Carol.

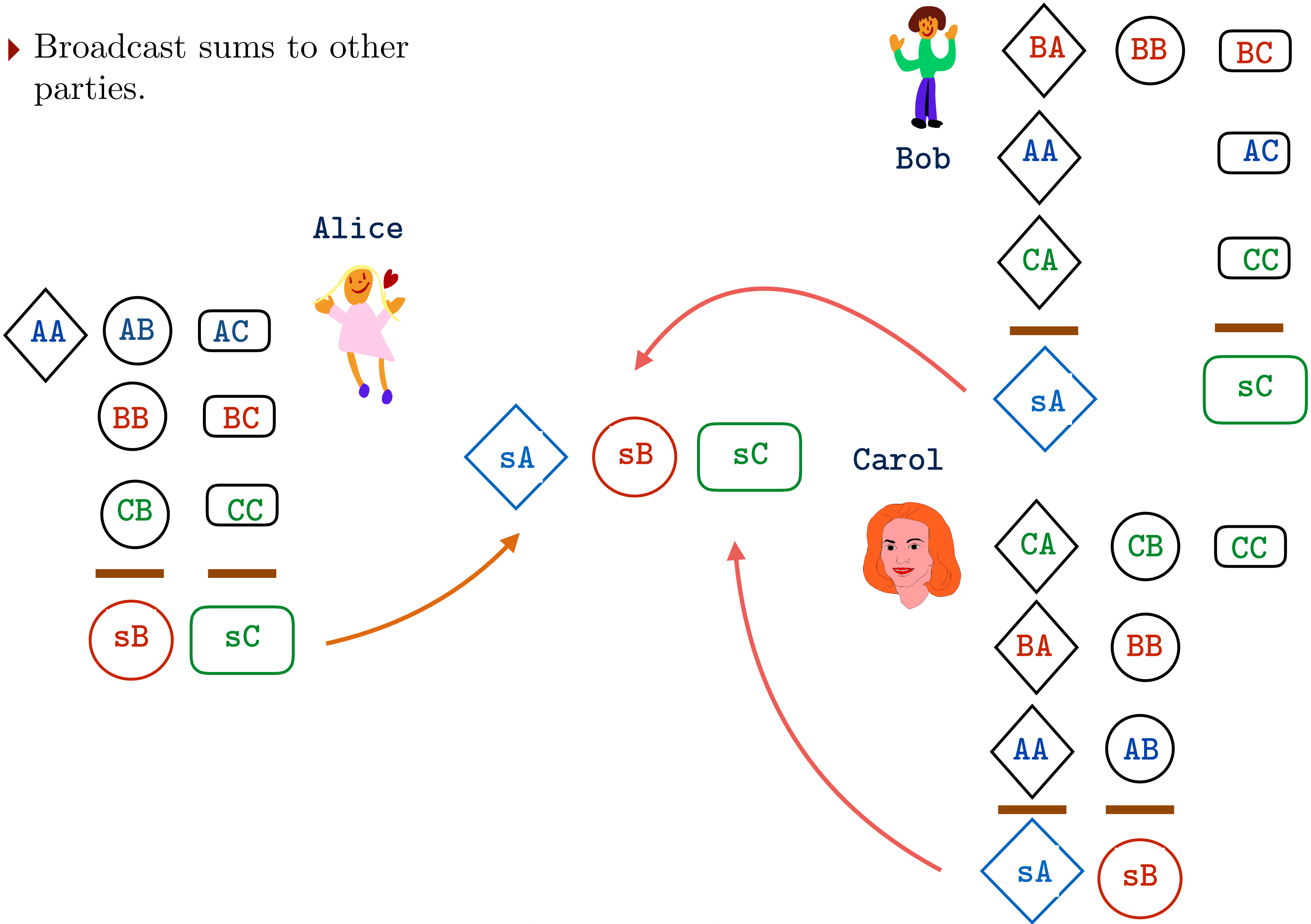






Example: Sum Broadcast

- Broadcast sums to other parties.



Example: Tally partial sums

- Everyone computes final tally of `Votes = sA + sB + sC`

Alice **Bob** **Carol**

`sA` `sB` `sC`

`Votes =` `sA` `+` `sB` `+` `sC`

Expanding

`Votes =` `AA + BA + CA` `+ AB + BB + CB` `+ AC + BC + CC`

Rearranging

`Votes =` `AA + AB + AC` `+ BA + BB + BC` `+ CA + CB + CC`

`Votes =` `A` `+` `B` `+` `C`

Example: Simple Secure Voting

- Why does this work?
- Let's replace Alice, Bob and Carol by 1, 2 and 3 then we have:

$$\sum_{X}^{3} vX \bmod p = \sum_{X}^{3}\sum_{Y}^{3} vXY \bmod p = \sum_{Y}^{3}\sum_{X}^{3} vXY \bmod p = \sum_{Y}^{3} sY \bmod p = Votes$$

- Note this works not just for binary values but for any integers - the protocol is really a **multiparty addition protocol -** so we could use it to compute the average salary also.

RSA in a nutshell

- **Plaintext** m

  Large primes p and q

  Composite $n = \mathbf{p \times q}$ — n is public

  Find e and d such that — $\boldsymbol{e \times d = 1} \mod (p\text{-}1)(q\text{-}1)$

  Encryption key e — **Public Key** $= (e, n)$

  Decryption key d — **Private Key** $= (d, n)$

- **Encryption** $= m^e \mod n$
  $= c$

- **Decryption** $= c^d \mod n$
  $= (m^e)^d \mod n$
  $= m^{ed} \mod n$
  $= m \mod n$

Example: Yao's Millionaires' problem

Two millionaires Alice and Bob want to know who is richer without disclosing their wealth.

Let's say Alice has $\boldsymbol{a}$=£4M, Bob has $\boldsymbol{b}$=£3M, $\boldsymbol{a}$ and $\boldsymbol{b}$ are integers in the range 1..6

- Bob chooses a large random number $\boldsymbol{r}$ (Bob's secret) and encrypts it with Alice's public key $E_{pubA}(\boldsymbol{r})$
- Bob sends $C=E_{pubA}(\boldsymbol{r})$ - $\boldsymbol{b}$ to Alice. C is the encryption of Bob's secret.
- Alice decrypts 6 values $Y_k=D_{privA}(C+k)$ for k=1..6 i.e. for each million, all look random.
- Alice then generates a large random prime $\boldsymbol{p}$ (but smaller than R) and reduces the decrypted values mod p, $Z_k = Y_k \bmod \boldsymbol{p}$, for k=1..6 and checks that all values differ by at least 2, otherwise starts again with a new random prime p.
- Alice sends $\boldsymbol{p}$ plus List=[$\mathbf{Z_1}$, $\mathbf{Z_2}$, $\mathbf{Z_3}$, $\mathbf{Z_4}$, $\mathbf{Z_5+1}$, $\mathbf{Z_6+1}$] Note: all values after the $\boldsymbol{a}$-th position have 1 added.
- Bob checks if the b-th value is his secret, i.e List[$\boldsymbol{b}$] ≡ r (mod $\boldsymbol{p}$)
- **If it is then $\boldsymbol{a}$ >= $\boldsymbol{b}$** otherwise $\boldsymbol{a} < \boldsymbol{b}$. Bobs tells Alice.

Alice   Bob

Wealth a : {1..6} = 4
Public Key pubA= (e, n)
Private Key privA= (d, n)

Wealth b : {1..6} =3

r = large random number (Bob's secret)

C = E_{pubA}(r) - b = r^e - b

<----- C

Y[1] = D_{privA}(C + 1)
     = D_{privA}(EpubA(r) - b + 1)
Y[2] = D_{privA}(EpubA(r) - b + 2)
Y[3] = DprivA(EpubA(r) - b + 3) = DprivA(EpubA(r)) = r
Y[4] = DprivA(EpubA(r) - b + 4)
Y[5] = DprivA(EpubA(r) - b + 5)
Y[6] = DprivA(EpubA(r) - b + 6)
p   = large random prime
Z[1] = Y[1] mod p
Z[2] = Y[2] mod p
Z[3] = Y[3] mod p = r mod p
Z[4] = Y[4] mod p
Z[5] = Y[5] mod p
Z[6] = Y[6] mod p

----- > p, Z[1], Z[2], Z[3] = r mod p, Z[4], Z[5]+1, Z[6]+1

list[b] ≡ r mod p?

r mod p ≡ r mod p is True

∴ Alice >= Bob

Essentially Alice and Bob have computed the predicate a >= b, neither knows the others wealth

Example

- Alice uses RSA giving n=19x29=551, e=5, d=101. Public key(5,551)
- Bobs chooses large random number r<n. Bob encrypts r using Alice's public key (i.e. $E_A(R)=R^e$ mod N) and sends $E_A(R)$-***b*** to Alice.

  If was r=123. Then $E_A(123)=123^5$ mod 551=16 and Bob sends 16-3=13
- Alice generates the following values $D_A(C+k)$ for k=1..6 i.e. each million.
  $D_A(14)=127$, $D_A(15)=250$, $D_A(16)=123$, $D_A(17)=365$, $D_A(18)=113$, $D_A(19)=304$
  Alice does not know which value corresponds to b=£3M.
- Alice picks a large random prime p (less than R).
- Alice computes list of values mod p. Lets assume p=47
  [127, 250, 123, 365, 113, 304] mod 47=[33,15,29,36,19,22] values differ by at least 2.
- Alice sends p and list of decrypted values but adds 1 to each position corresponding to millions greater than hers (***a***=4) i.e. £5M and £6M (last two in list).
  Alice sends p=47, List=[33, 15, 29, 36, 20, 23] to Bob.
- Bob checks if List[***b***] ≡ r mod p i.e. 29 = 123 mod 47. It is therefore Alice(***a***) >=Bob(***b***) otherwise Bob(***b***)>Alice(***a***). Bobs tells Alice.
- Effectively Alice adds 1 to values greater than hers (***a***=4). Bob checks if the one in his position (***b***=3) has one added to it. If it has then he is richer than Alice.

Other Examples

- **Private Auctions.** N bidders make offers. Determine highest bidder without revealing offers.
- **Private Set Intersection.** N data providers with common data, e.g. customer records. They wish to determine common customers
- **Poker.** With no dealer
- Others?

- ***Danish sugar beet auction (2008)***
- Several thousand Danish sugar beet farmers.
- Goal was to find market clearing price (price per unit of commodity).
- Each buyer/seller specifies how much they are willing to buy/sell at each price point.
- Auctioneer computes supply/demand at each price point.
- Aim is to find price where supply=demand (supply grows, demand decreases with increasing price). All bids at the price accepted.
- Auctioneer is implemented as a 3-party SMC protocol using a secret sharing scheme.

SMC Approaches

- ***Garbled Circuits (e.g. Yao)***
  - Express computation as boolean circuit encrypted gates
  - Good for 2 parties, e.g. secure function evaluation
  - Assumes encryption scheme is secure.
  - Can be made secure against a malicious adversary.
  - Interactive - Alice and Bob communicate

- ***Secret Sharing Schemes (e.g. Shamir)***
  - Arithmetic circuit
  - Better for 3 or more parties
  - Can be made secure against a malicious adversary.
  - Based on perfect security

- ***Fully homomorphic encryption (FHE)***
  - Computation on encrypted data - Only decrypt at end.
  - Non-interactive - Little communication, more computation.

PROPERTIES:

- **Privacy:** Inputs remain private
- **Correct**: Output is correct
- **Fairness**: Everyone learns output.
- **Fault-tolerance**: Tolerance to attacks and faults.
- **Performance-Resources**: Computation, Communication, Latency

Adversarial Models

- ***Honest-but-curious* (Semi-Honest) Adversary**
  - Follows protocol, but is interested (i.e. curious) in breaking privacy of other parties.
  - Can collude with other parties (said to be a ***corrupt*** party)
- ***Malicious* (Byzantine) Adversary**
  - Can deviate from the protocol e.g. lie about inputs, quit protocol early.
  - Can also collude with other parties
- **Asynchronous Communication**
  - One party might learn result of computation before others and may be able to prevent others from acquiring the result, e.g. by not forwarding it them
- ***Fair* SMC protocol**
  - When protocol is secure against malicious adversary not forwarding last message.

- For **honest-but-curious parties**, we can tolerant $n/2$ corrupt parties
- For **malicious parties,** we can tolerate:

  - **$n/3$ corrupt parties** with ***perfect security*** (i.e. an adversary with infinite computing power can learn nothing about the plaintext from the ciphertext)

  - **$n/2$ corrupt parties** with a computational assumption e.g. adversary cannot break symmetric encryption scheme.

XOR in a nutshell

$A \oplus A = 0$	**0 if same**
$A \oplus$ not $A = 1$	**1 if different**
i.e $0 \oplus 0 = 0$	
$0 \oplus 1 = 1$	
$1 \oplus 0 = 1$	
$1 \oplus 1 = 0$	

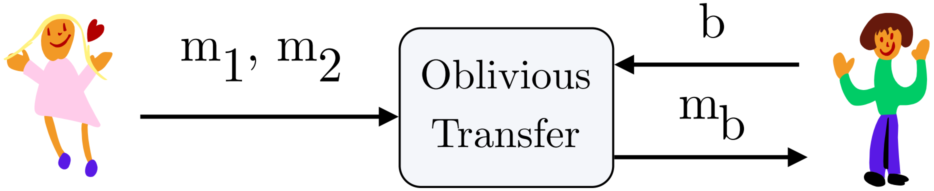
$A \oplus 0 = A$	**0 Passthrough**
$A \oplus 1 =$ not A	**1 Invert**

$A \oplus K$	**Once is encrypt**
$A \oplus K \oplus K = A$	**Twice is decrypt**

- Write a sequence of variable assignments to swap the values in two variables A and B without using a 3rd variable, i.e. only using A and B.

Oblivious Transfer (1-from-2 OT)

- Alice sends Bob two messages. Bob learns one of the two (not both). Alice doesn't know which message Bob learnt. (secure selection of 1 value)



Example

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

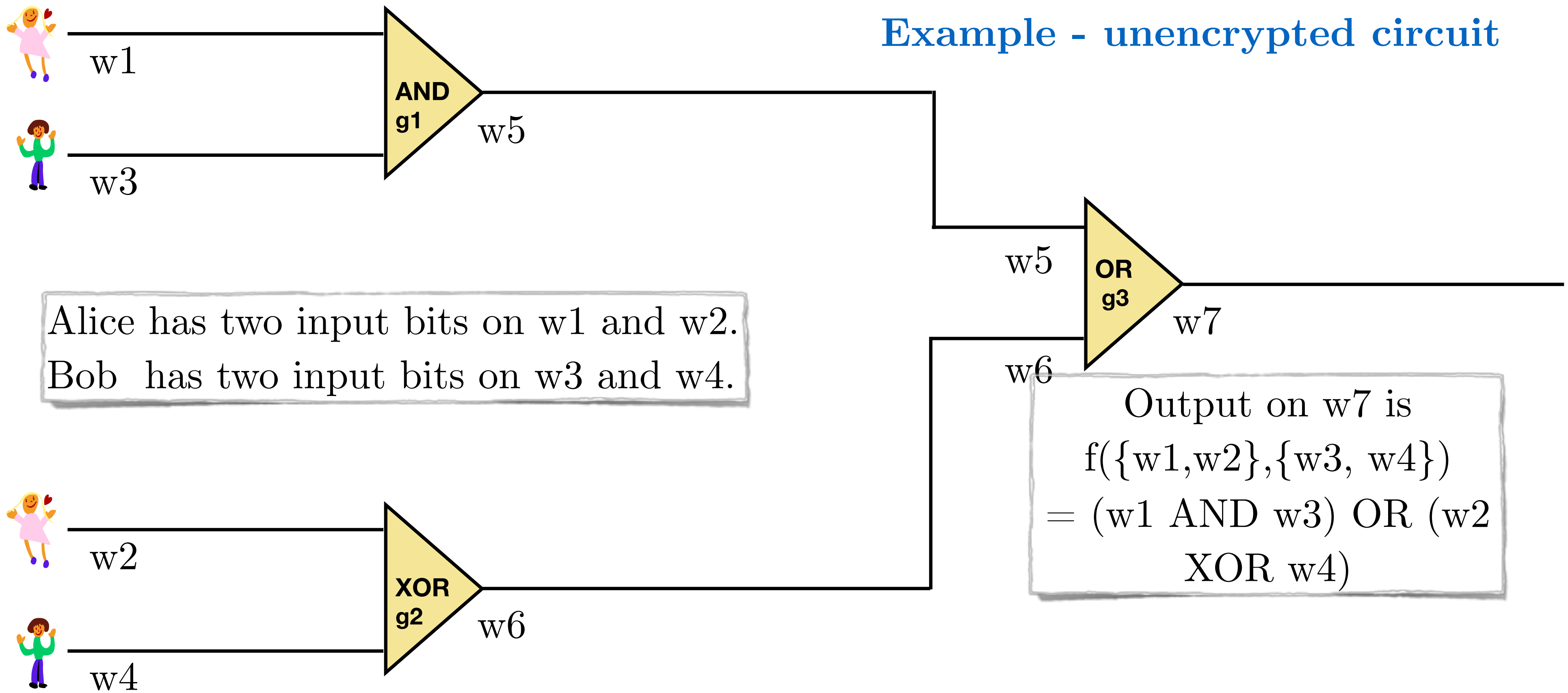
Alice → Bob:	Pub1, Pub2	Bob generates a symmetric key K and randomly chooses Pub1 say
Bob → Alice:	c=E_{Pub1}(K)	Alice does D_{Priv1}(c)=Good key K and D_{Priv2}(c)=Rubbish key U
Alice → Bob:	c1=E_K(m1), c2=E_U(m2)	Bob does D_K(c1)=Good message m1 and D_K(c2)=Rubbish message

Bob now knows m1. Alice doesn't know which message Bob knows. To prevent Alice cheating (e.g. using same message m1=m2), Alice should send Priv1 and Priv2 to Bob for verification *after* protocol is done (Bob will be able to decrypt other message though).

Secure Function Evaluation— Yao's Garbled Circuits (2-party)

- 1. Alice creates a cleverly encrypted boolean circuit of the function to be evaluated called a Garbled circuit and sends it to Bob along with keys for her inputs, plus a mapping table to "decrypt" the outputs of the circuit.
- 2. Bob applies his inputs to the garbled circuit, effectively evaluating each gate to get the outputs and using the mapping table on the circuit outputs.

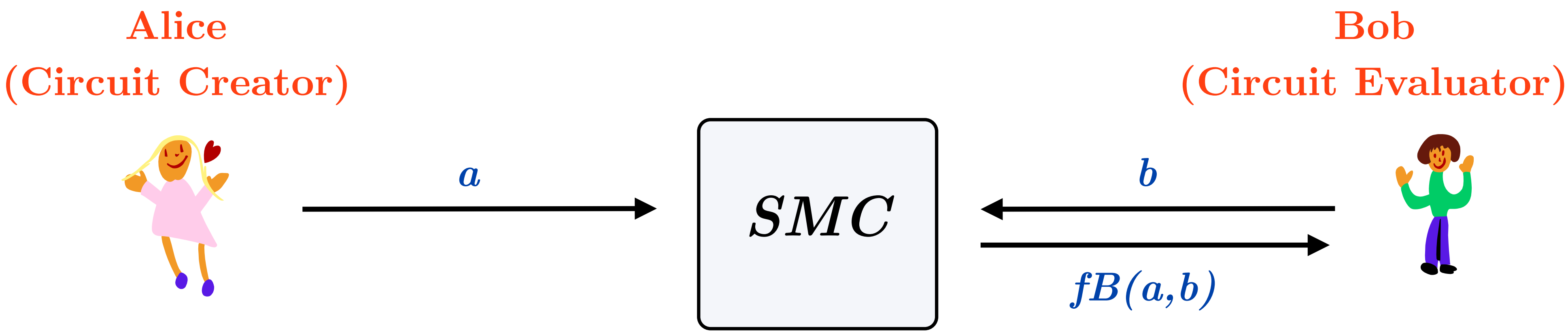
Example - unencrypted circuit



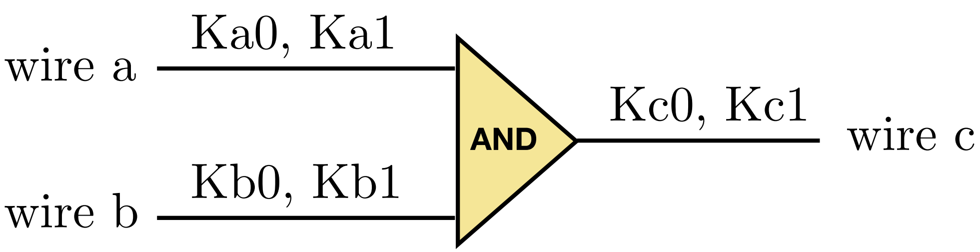
Alice has two input bits on w1 and w2.
Bob has two input bits on w3 and w4.

Output on w7 is f({w1,w2},{w3, w4}) = (w1 AND w3) OR (w2 XOR w4)

Yao's Two-party Circuit Evaluation 1

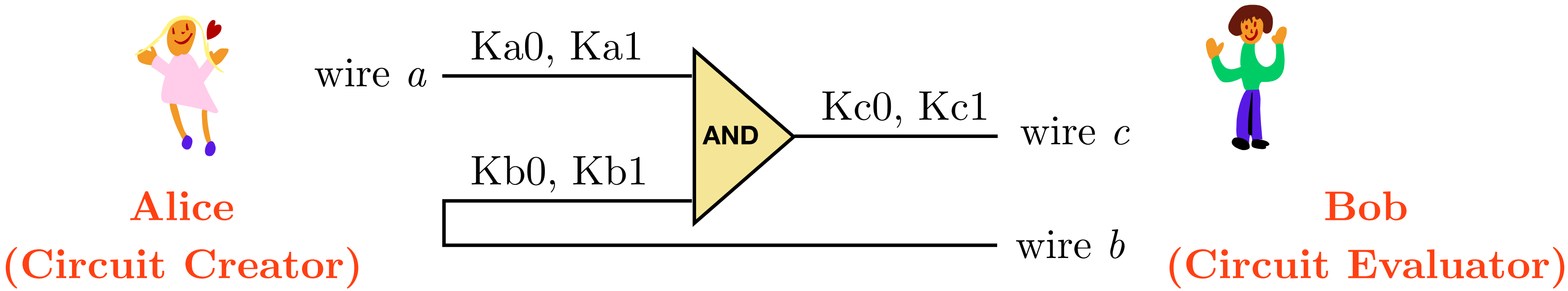


- Bob wants to evaluate $fB(a, b)$.
- Alice creates a boolean circuit using logic gates (e.g. AND, OR, NOT, NAND, NOR, XOR) and wires corresponding to f and sends it to Bob
- For each wire, Alice creates 2 random keys. One key corresponds to the encryption of the value 0, the other to the encryption of the value 1. For each gate we compute a *garbled table* for the gate (e.g. AND below)



a	b	c	Garbled **AND**
Ka0	Kb0	Kc0	$E_{Ka0}(E_{Kb0}(Kc0))$
Ka0	Kb1	Kc0	$E_{Ka0}(E_{Kb1}(Kc0))$
Ka1	Kb0	Kc0	$E_{Ka1}(E_{Kb0}(Kc0))$
Ka1	Kb1	Kc1	$E_{Ka1}(E_{Kb1}(Kc1))$

Yao's Two-party Circuit Evaluation 2



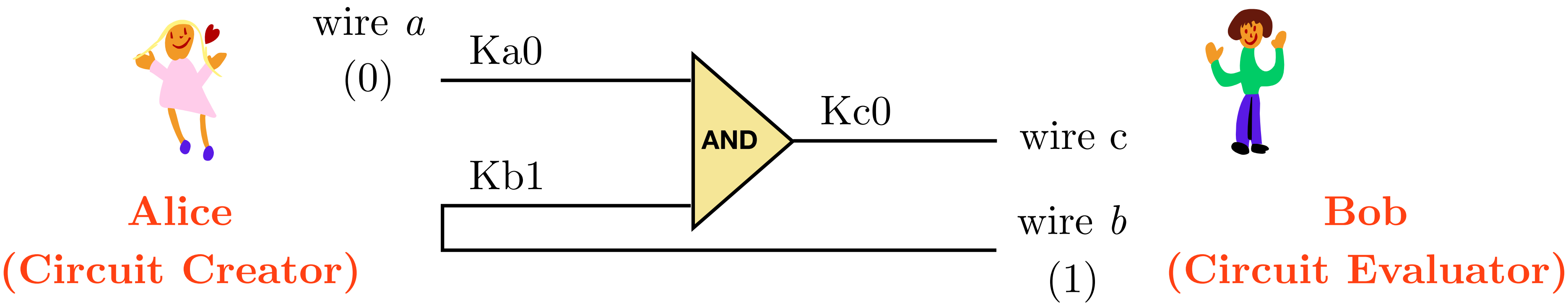
Garbled AND
$E_{Ka0}(E_{Kb0}(Kc0))$
$E_{Ka0}(E_{Kb1}(Kc0))$
$E_{Ka1}(E_{Kb0}(Kc0))$
$E_{Ka1}(E_{Kb1}(Kc1))$

Ka0 or Ka1 as appropriate

Garbled AND
$E_{Ka1}(E_{Kb0}(Kc0))$
$E_{Ka0}(E_{Kb1}(Kc0))$
$E_{Ka1}(E_{Kb1}(Kc1))$
$E_{Ka0}(E_{Kb0}(Kc0))$

- Alice randomly permutates the rows and sends the Garbled table to Bob. Bob doesn't know which row of the garbled table corresponds to which row in the original table.
- Alice also sends the key corresponding to her input bit to Bob. For example, if the bit is 0 Alice sends Ka0. Note: Bob doesn't know that this corresponds to 0 since it's random.

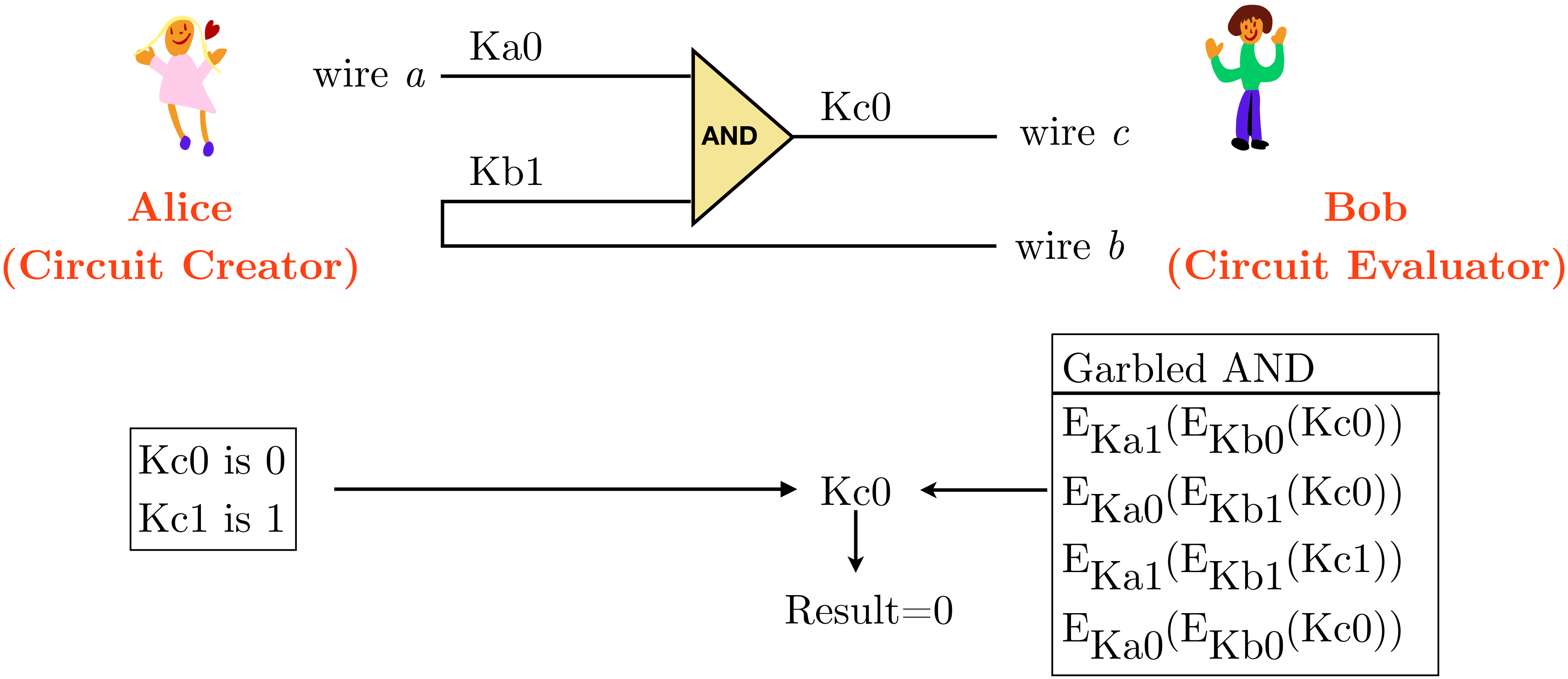
Yao's Two-party Circuit Evaluation 3



Garbled AND
$E_{Ka1}(E_{Kb0}(Kc0))$
$E_{Ka0}(E_{Kb1}(Kc0))$
$E_{Ka1}(E_{Kb1}(Kc1))$
$E_{Ka0}(E_{Kb0}(Kc0))$

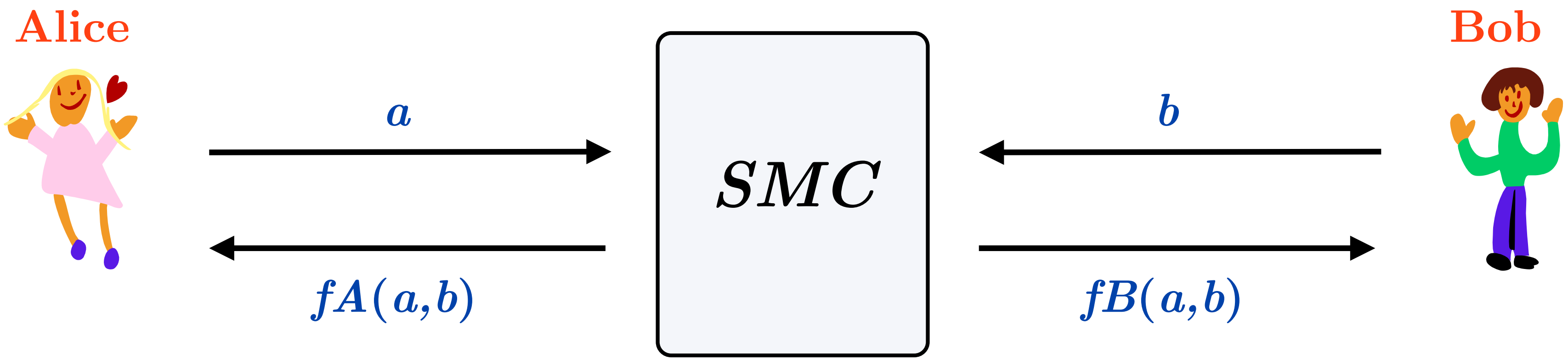
- Alice and Bob run an oblivious transfer (OT) protocol
- Alice's input to the OT protocol are keys Kb0 and Kb1
- Bob's input to OT his is 1-bit input $<b>$, which is used to select Kb0 or Kb1
- Bob learns Kb$<b>$. Alice learns nothing.
- If Alice had sent Ka0 to Bob and Bob's input (b) was 1 then Bob would effectively decrypt using Ka0 followed by Kb1 to get Kc0

Yao's Two-party Circuit Evaluation 4



- To complete the evaluation Alice tells Bob the Keys for the output wire, i.e. Kc0 is 0, Kc1 is 1, so that Bob can learn the final value.
- For a bigger circuit, Bob does not learn any intermediate value, since he only has the final key and the final value.
- Of course for a 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 learns nothing.

Two-Party Case: Separate functions



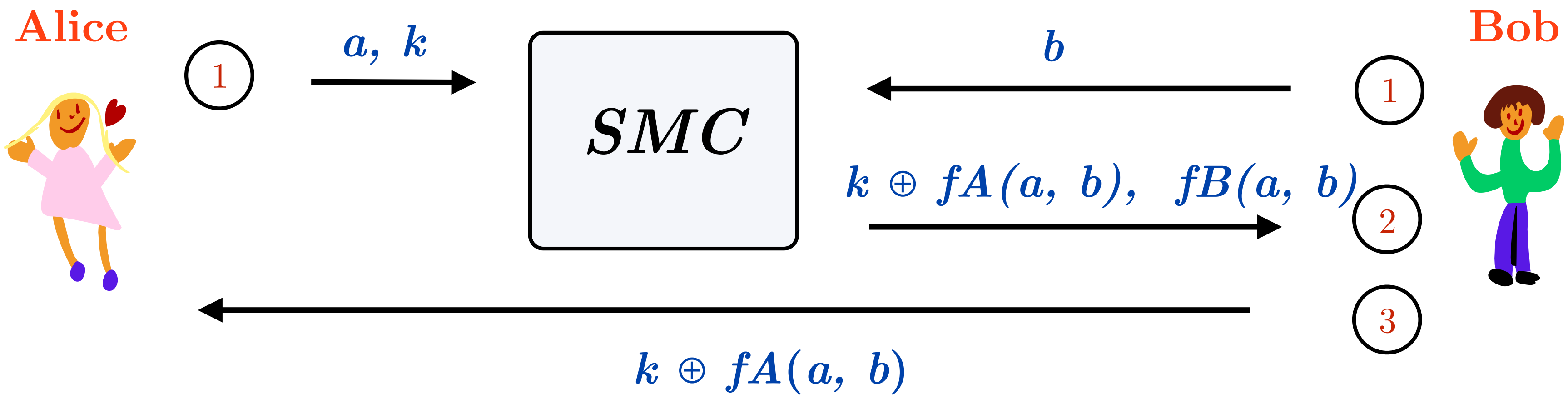
- Alice wishes to compute $fA(a,\ b)$ without Bob learning a or $fA(a,\ b)$
- Bob wishes to compute $fB(a,\ b)$ without Alice learning b or $fB(a,\ b)$

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

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

- 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”. $\oplus$ *is XOR*
- Only Bob learns the output of this function.

Two-Party Case Continued



- At 2. Bob learns $f(a,b,k) = k \oplus fA(a,b),\ fB(a,b)$ from SMC protocol

  Bob sends first part $k \oplus fA(a,b)$ to Alice, keeps second part $fB(a,b)$

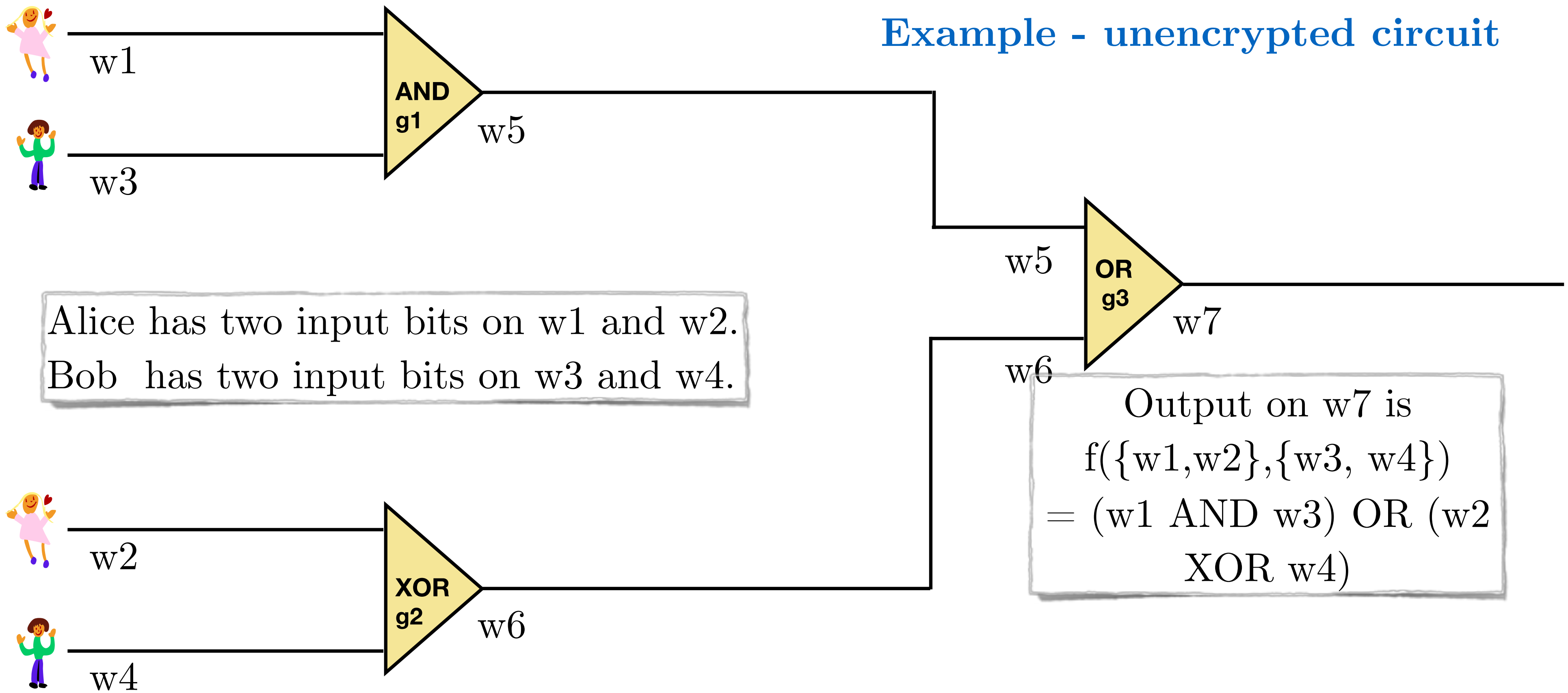
- At 3. Alice computes $fA(a,b) = k \oplus (k \oplus fA(a,b))$ by xoring with secret k

- *This approach is used where Bob will compute both results but does not know Alice's result since it is encrypted with Alice's key k.*

3-Gate Binary Circuit: Wires and Gates

- Binary circuit: a set of wires {w1, ..., wn} and a set of gates {g1, ..., gm}
- Each gate is a function with values on two (one for NOT) input wires and produces the value on one output wire (AND, OR, XOR, NOT, NAND, NOR)

Example: Alice and Bob have two inputs bits: Alice's on w1,w2. Bob's on w3,w4.
Output is f({w1,w2},{w3, w4}) = (w1 AND w3) OR (w2 XOR w4)

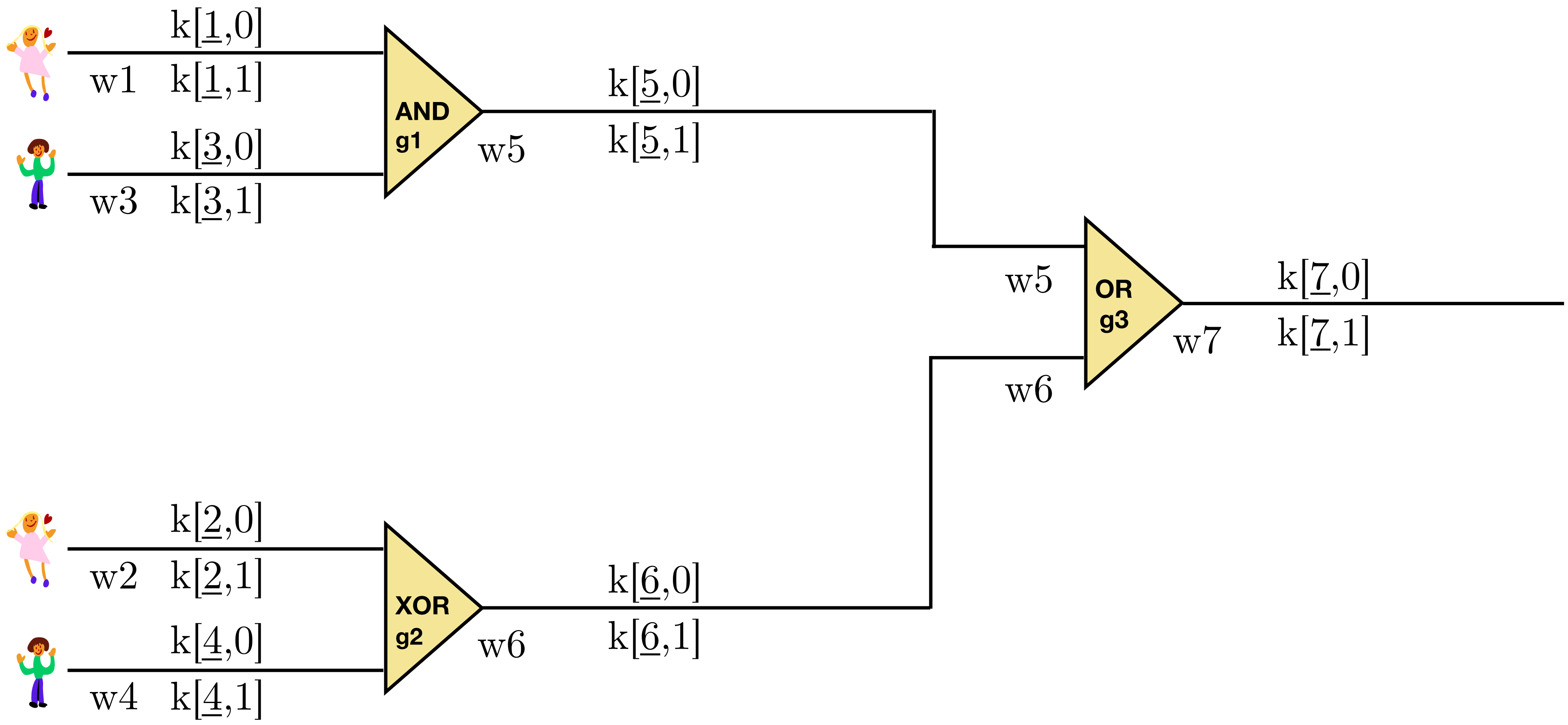


Keys

- Each wire w has two random keys

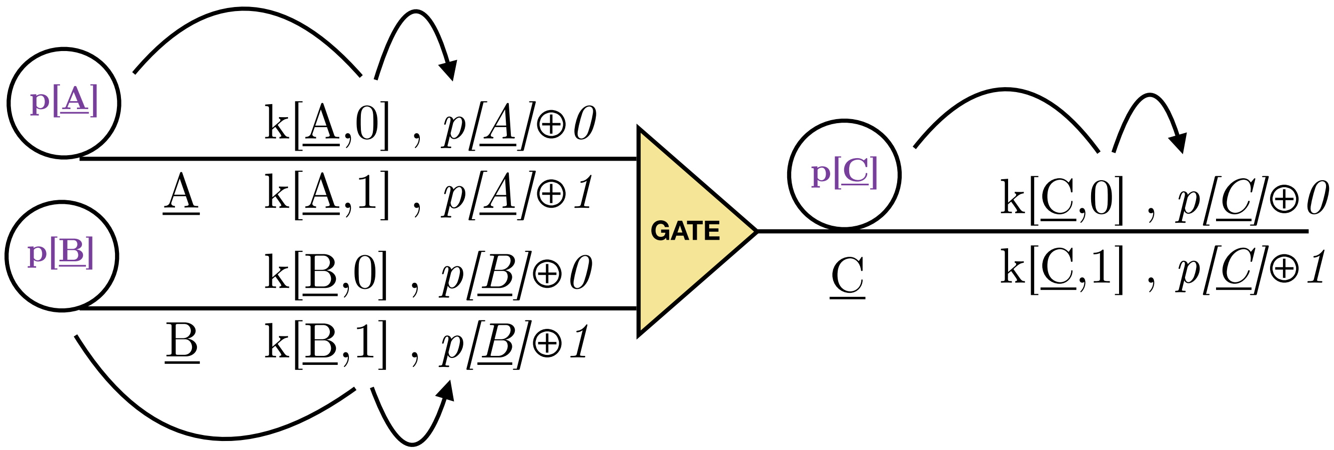
  k[w,0] for encryption of a 0 value
  k[w,1] for encryption of a 1 value

- The two input keys to a gate will be used to recover the output key which can be used as an input key to the next gate.



p-bits

- Each wire w also has a **random 0 or 1** value (key) **p[w]** that is used to **encrypt** the actual values v[w] using XOR i.e. **p[w]** ⊕ **v[w]** and to permutate garbled tables. Sometimes called a *colouring-bit*.
- For a gate with input wires A, B and output wire C we have:

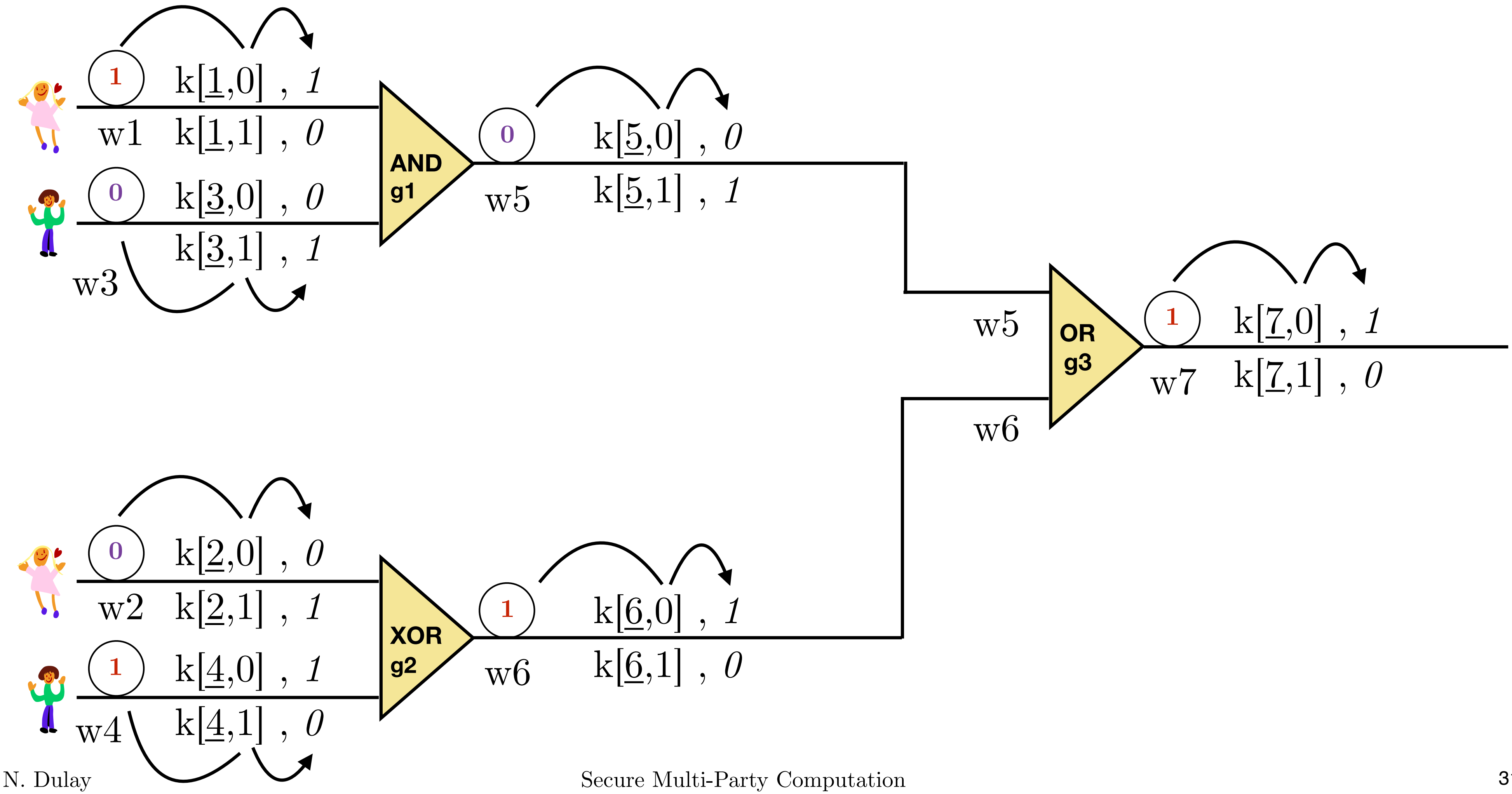


p-bits Example

- For example, the circuit with the following p values(keys) is shown below.

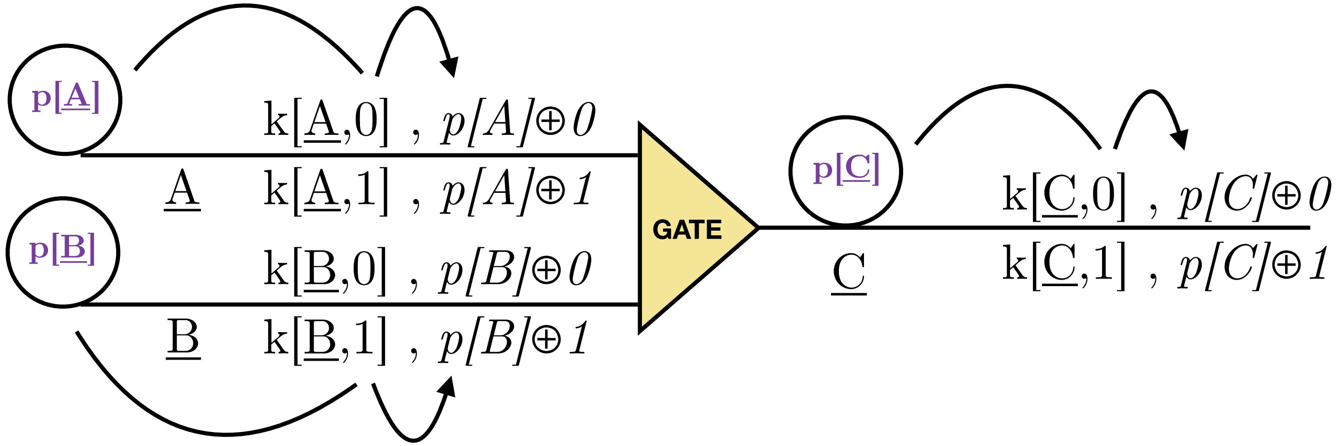
  p[1]=p[4]=p[6]=p[7]= 1 and p[2]=p[3]=p[5]= 0

- Recall that xor-ing with 0 passes the value through, while 1 inverts it.



Garbled Tables for Gates

- For each gate we compute a garbled (encrypted) and permutated table for all 4 combinations of encrypted wire values. For a gate with input wires A, B and output wire C we have:



		Encrypted A,B values		Encrypted C value
$[0,0] = E_{k[A,x],k[B,y]}(k[C,z],\ t)$	where	$x=0 \oplus p[A]$, $y=0 \oplus p[B]$,	$z=GATE(x,\ y)$,	$t=z \oplus p[C]$
$[0,1] = E_{k[A,x],k[B,y]}(k[C,z],\ t)$	where	$x=0 \oplus p[A]$, $y=1 \oplus p[B]$,	$z=GATE(x,\ y)$,	$t=z \oplus p[C]$
$[1,0] = E_{k[A,x],k[B,y]}(k[C,z],\ t)$	where	$x=1 \oplus p[A]$, $y=0 \oplus p[B]$,	$z=GATE(x,\ y)$,	$t=z \oplus p[C]$
$[1,1] = E_{k[A,x],k[B,y]}(k[C,z],\ t)$	where	$x=1 \oplus p[A]$, $y=1 \oplus p[B]$,	$z=GATE(x,\ y)$,	$t=z \oplus p[C]$

Exercise: Garbled Tables for Gates 1, 2 and 3 ?

Compute the garbled tables for

- Gate g1 (AND) with p[$\underline{1}$]=1, p[$\underline{3}$]=0, p[$\underline{5}$]=0
- Gate g2 (XOR) with p[$\underline{2}$]=0, p[$\underline{4}$]=1, p[$\underline{6}$]=1
- Gate g3 (OR) with p[$\underline{5}$]=0, p[$\underline{6}$]=1, p[$\underline{7}$]=1

Garbled Table for Gate 1

- For each gate we compute a garbled table for all 4 combinations of encrypted wire values. For gate g1 we have p[$\underline{1}$]=1, p[$\underline{3}$]=0, p[$\underline{5}$]=0

$$E_{k[\underline{1},x],k[\underline{3},y]}(k[\underline{5},z],\ t) \quad \text{where } x=\mathit{0}\oplus p[\underline{1}],\ y=\mathit{0}\oplus p[\underline{3}],\ z=\mathrm{AND}(x,\ y),\ t=z\oplus p[\underline{5}]$$
$$E_{k[\underline{1},x],k[\underline{3},y]}(k[\underline{5},z],\ t) \quad \text{where } x=\mathit{0}\oplus p[\underline{1}],\ y=\mathit{1}\oplus p[\underline{3}],\ z=\mathrm{AND}(x,\ y),\ t=z\oplus p[\underline{5}]$$
$$E_{k[\underline{1},x],k[\underline{3},y]}(k[\underline{5},z],\ t) \quad \text{where } x=\mathit{1}\oplus p[\underline{1}],\ y=\mathit{0}\oplus p[\underline{3}],\ z=\mathrm{AND}(x,\ y),\ t=z\oplus p[\underline{5}]$$
$$E_{k[\underline{1},x],k[\underline{3},y]}(k[\underline{5},z],\ t) \quad \text{where } x=\mathit{1}\oplus p[\underline{1}],\ y=\mathit{1}\oplus p[\underline{3}],\ z=\mathrm{AND}(x,\ y),\ t=z\oplus p[\underline{5}]$$

$$E_{k[\underline{1},x],k[\underline{3},y]}(k[\underline{5},z],\ t) \quad \text{where } x=\mathit{0}\oplus 1=1,\ y=\mathit{0}\oplus 0=0,\ z=\mathrm{AND}(1,\ 0)=0,\ t=0\oplus 0=0$$
$$E_{k[\underline{1},x],k[\underline{3},y]}(k[\underline{5},z],\ t) \quad \text{where } x=\mathit{0}\oplus 1=1,\ y=\mathit{1}\oplus 0=1,\ z=\mathrm{AND}(1,\ 1)=1,\ t=1\oplus 0=1$$
$$E_{k[\underline{1},x],k[\underline{3},y]}(k[\underline{5},z],\ t) \quad \text{where } x=\mathit{1}\oplus 1=0,\ y=\mathit{0}\oplus 0=0,\ z=\mathrm{AND}(0,\ 0)=0,\ t=0\oplus 0=0$$
$$E_{k[\underline{1},x],k[\underline{3},y]}(k[\underline{5},z],\ t) \quad \text{where } x=\mathit{1}\oplus 1=0,\ y=\mathit{1}\oplus 0=1,\ z=\mathrm{AND}(0,\ 1)=0,\ t=0\oplus 0=0$$

$$E_{k[\underline{1},1],k[\underline{3},0]}(k[\underline{5},0],\ 0) \quad \text{where } x=1,\ y=0,\ z=0,\ t=0$$
$$E_{k[\underline{1},1],k[\underline{3},1]}(k[\underline{5},1],\ 1) \quad \text{where } x=1,\ y=1,\ z=1,\ t=1$$
$$E_{k[\underline{1},0],k[\underline{3},0]}(k[\underline{5},0],\ 0) \quad \text{where } x=0,\ y=0,\ z=0,\ t=0$$
$$E_{k[\underline{1},0],k[\underline{3},1]}(k[\underline{5},0],\ 0) \quad \text{where } x=0,\ y=1,\ z=0,\ t=0$$

Garbled Table for Gate 2

- For each gate we compute a garbled table for all 4 combinations of encrypted wire values. For gate g2 we have p[2]=0, p[4]=1, p[6]=1

$E_{k[2,x],k[4,y]}(k[6,z],\ t)$ where $x=\mathit{0}\oplus p[2]$, $y=\mathit{0}\oplus p[4]$, $z=XOR(x,\ y)$, $t=z\oplus p[6]$
$E_{k[2,x],k[4,y]}(k[6,z],\ t)$ where $x=\mathit{0}\oplus p[2]$, $y=\mathit{1}\oplus p[4]$, $z=XOR(x,\ y)$, $t=z\oplus p[6]$
$E_{k[2,x],k[4,y]}(k[6,z],\ t)$ where $x=\mathit{1}\oplus p[2]$, $y=\mathit{0}\oplus p[4]$, $z=XOR(x,\ y)$, $t=z\oplus p[6]$
$E_{k[2,x],k[4,y]}(k[6,z],\ t)$ where $x=\mathit{1}\oplus p[2]$, $y=\mathit{1}\oplus p[4]$, $z=XOR(x,\ y)$, $t=z\oplus p[6]$

$E_{k[2,x],k[4,y]}(k[6,z],\ t)$ where $x=\mathit{0}\oplus 0=0$, $y=\mathit{0}\oplus 1=1$, $z=XOR(0,\ 1)=1$, $t=1\oplus 1=0$
$E_{k[2,x],k[4,y]}(k[6,z],\ t)$ where $x=\mathit{0}\oplus 0=0$, $y=\mathit{1}\oplus 1=0$, $z=XOR(0,\ 0)=0$, $t=0\oplus 1=1$
$E_{k[2,x],k[4,y]}(k[6,z],\ t)$ where $x=\mathit{1}\oplus 0=1$, $y=\mathit{0}\oplus 1=1$, $z=XOR(1,\ 1)=0$, $t=0\oplus 1=1$
$E_{k[2,x],k[4,y]}(k[6,z],\ t)$ where $x=\mathit{1}\oplus 0=1$, $y=\mathit{1}\oplus 1=0$, $z=XOR(1,\ 0)=1$, $t=1\oplus 1=0$

$E_{k[2,0],k[4,1]}(k[6,1],\ 0)$ where $x=0$, $y=1$, $z=1$, $t=0$
$E_{k[2,0],k[4,0]}(k[6,0],\ 1)$ where $x=0$, $y=0$, $z=0$, $t=1$
$E_{k[2,1],k[4,1]}(k[6,0],\ 1)$ where $x=1$, $y=1$, $z=0$, $t=1$
$E_{k[2,1],k[4,0]}(k[6,1],\ 0)$ where $x=1$, $y=0$, $z=1$, $t=0$

> There is an optimisation that can be applied which avoids the need for a garbled table for XOR gates

Garbled Table for Gate 3

- For each gate we compute a garbled table for all 4 combinations of encrypted wire values. For gate g3 we have p[5]=0, p[6]=1, p[7]=1

$E_{k[5,x],k[6,y]}(k[7,z],\ t)$ where $x=\mathit{0}\oplus p[5]$, $y=\mathit{0}\oplus p[6]$, $z=OR(x,\ y)$, $t=z\oplus p[7]$
$E_{k[5,x],k[6,y]}(k[7,z],\ t)$ where $x=\mathit{0}\oplus p[5]$, $y=\mathit{1}\oplus p[6]$, $z=OR(x,\ y)$, $t=z\oplus p[7]$
$E_{k[5,x],k[6,y]}(k[7,z],\ t)$ where $x=\mathit{1}\oplus p[5]$, $y=\mathit{0}\oplus p[6]$, $z=OR(x,\ y)$, $t=z\oplus p[7]$
$E_{k[5,x],k[6,y]}(k[7,z],\ t)$ where $x=\mathit{1}\oplus p[5]$, $y=\mathit{1}\oplus p[6]$, $z=OR(x,\ y)$, $t=z\oplus p[7]$

$E_{k[5,x],k[6,y]}(k[7,z],\ t)$ where $x=\mathit{0}\oplus 0=0$, $y=\mathit{0}\oplus 1=1$, $z=OR(0,\ 1)=1$, $t=1\oplus 1=0$
$E_{k[5,x],k[6,y]}(k[7,z],\ t)$ where $x=\mathit{0}\oplus 0=0$, $y=\mathit{1}\oplus 1=0$, $z=OR(0,\ 0)=0$, $t=0\oplus 1=1$
$E_{k[5,x],k[6,y]}(k[7,z],\ t)$ where $x=\mathit{1}\oplus 0=1$, $y=\mathit{0}\oplus 1=1$, $z=OR(1,\ 1)=1$, $t=1\oplus 1=0$
$E_{k[5,x],k[6,y]}(k[7,z],\ t)$ where $x=\mathit{1}\oplus 0=1$, $y=\mathit{1}\oplus 1=0$, $z=OR(1,\ 0)=1$, $t=1\oplus 1=0$

$E_{k[5,0],k[6,1]}(k[7,1],\ 0)$ where $x=0$, $y=1$, $z=1$, $t=0$
$E_{k[5,0],k[6,0]}(k[7,0],\ 1)$ where $x=0$, $y=0$, $z=0$, $t=1$
$E_{k[5,1],k[6,1]}(k[7,1],\ 0)$ where $x=1$, $y=1$, $z=1$, $t=0$
$E_{k[5,1],k[6,0]}(k[7,1],\ 0)$ where $x=1$, $y=0$, $z=1$, $t=0$

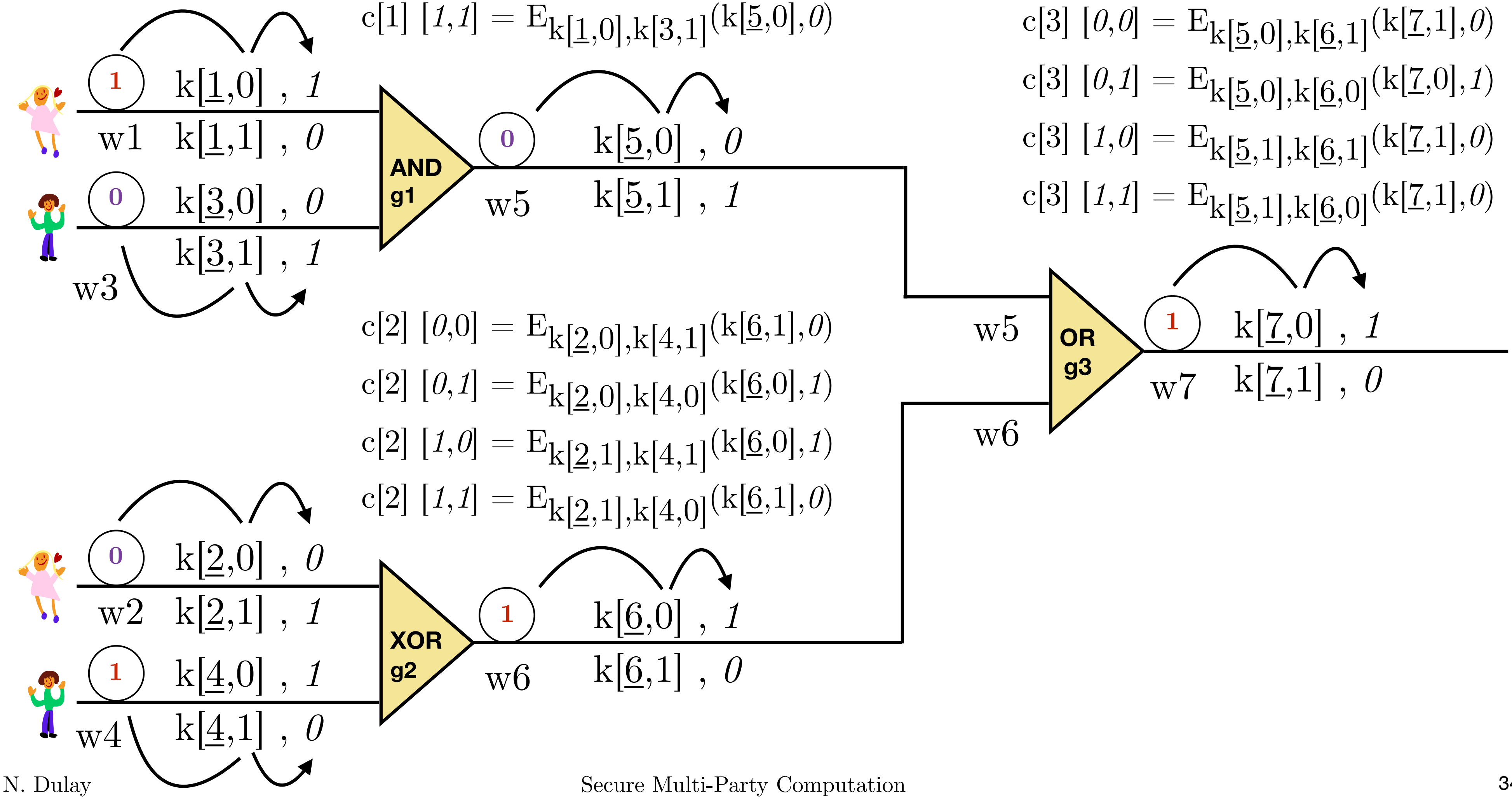
Garbled Circuit for example

- Garbled circuit created by Alice

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

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

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



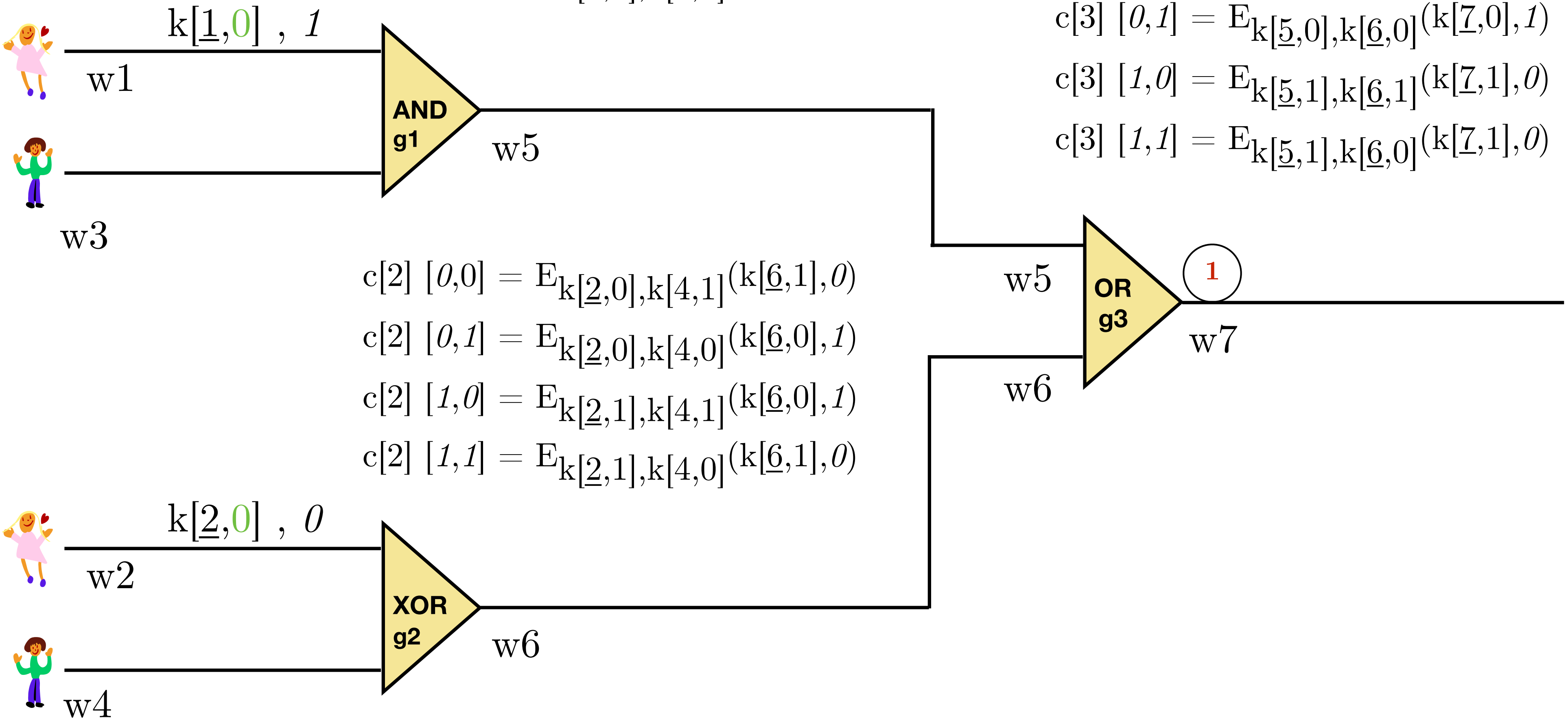
Alice to Bob

- Alice sends to Bob the garbled tables plus the wire keys and encrypted bits for its input bits v[1]=0 and v[2]=0 and the decryption bit for the output wire i.e $p[7]=1$

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

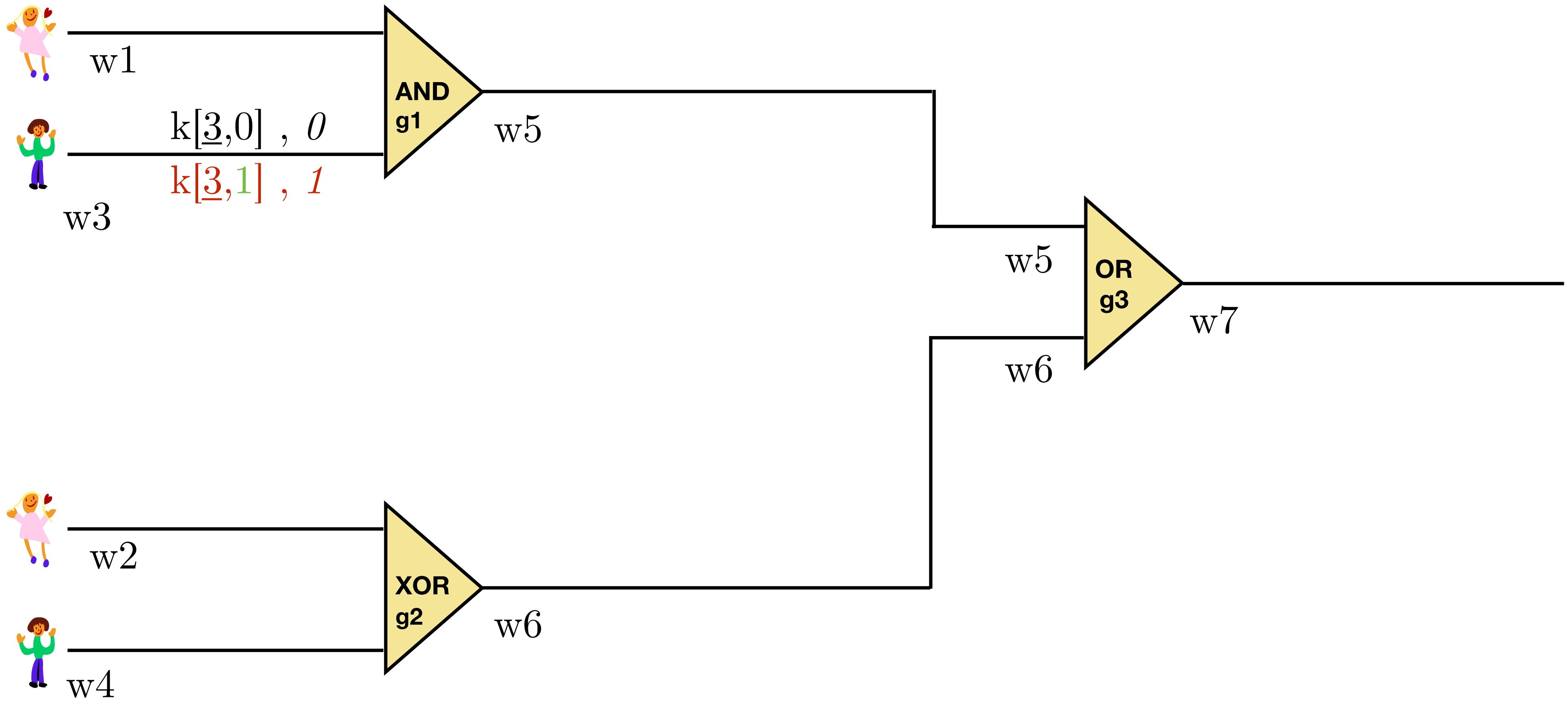
$$c[3]\ [\mathit{0},\mathit{0}] = E_{k[\underline{5},0],k[\underline{6},1]}(k[\underline{7},1],\mathit{0})$$
$$c[3]\ [\mathit{0},\mathit{1}] = E_{k[\underline{5},0],k[\underline{6},0]}(k[\underline{7},0],\mathit{1})$$
$$c[3]\ [\mathit{1},\mathit{0}] = E_{k[\underline{5},1],k[\underline{6},1]}(k[\underline{7},1],\mathit{0})$$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

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



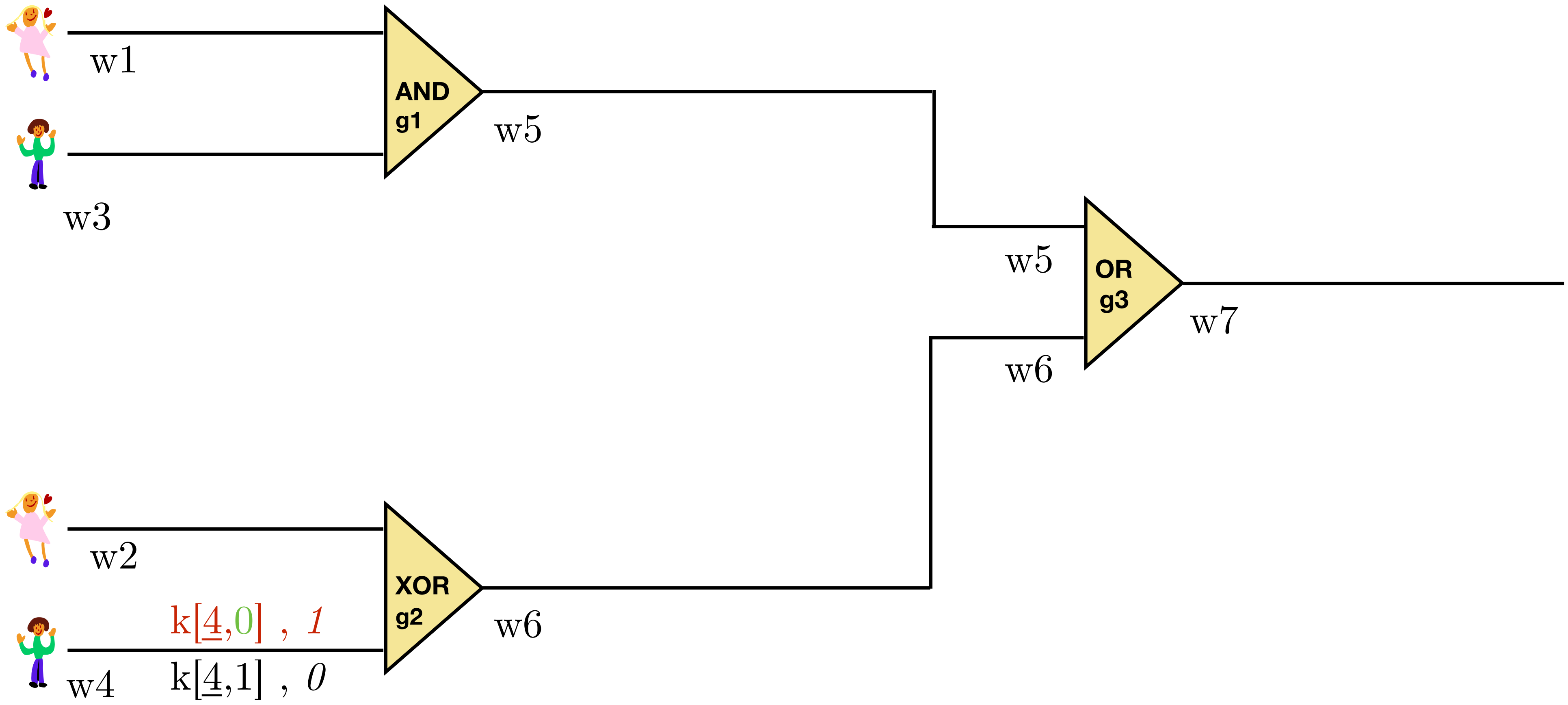
Bob and Alice engage in OT for v[3]=1

- Bob engages in an oblivious transfer with Alice using input v[3]=1 to privately select one Alice's keys/bits for wire3 i.e. from k[3,0], *0* and k[3,1], *1*
- Bob learns k[3,1], *1*



Bob and Alice engage in a 2nd OT for v[4]=0

- Bob engages in a 2nd oblivious transfer with Alice using input v[4]=0 to privately selects one of Alice's keys/bits or for wire3 i.e. from k[4,0], 1 and k[4,1], 0
- Bob will learn k[4,0], *1*



Garbled Circuit (known by Bob)

- On completion of the Oblivious Transfers Bob knows:

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

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

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

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

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

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

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

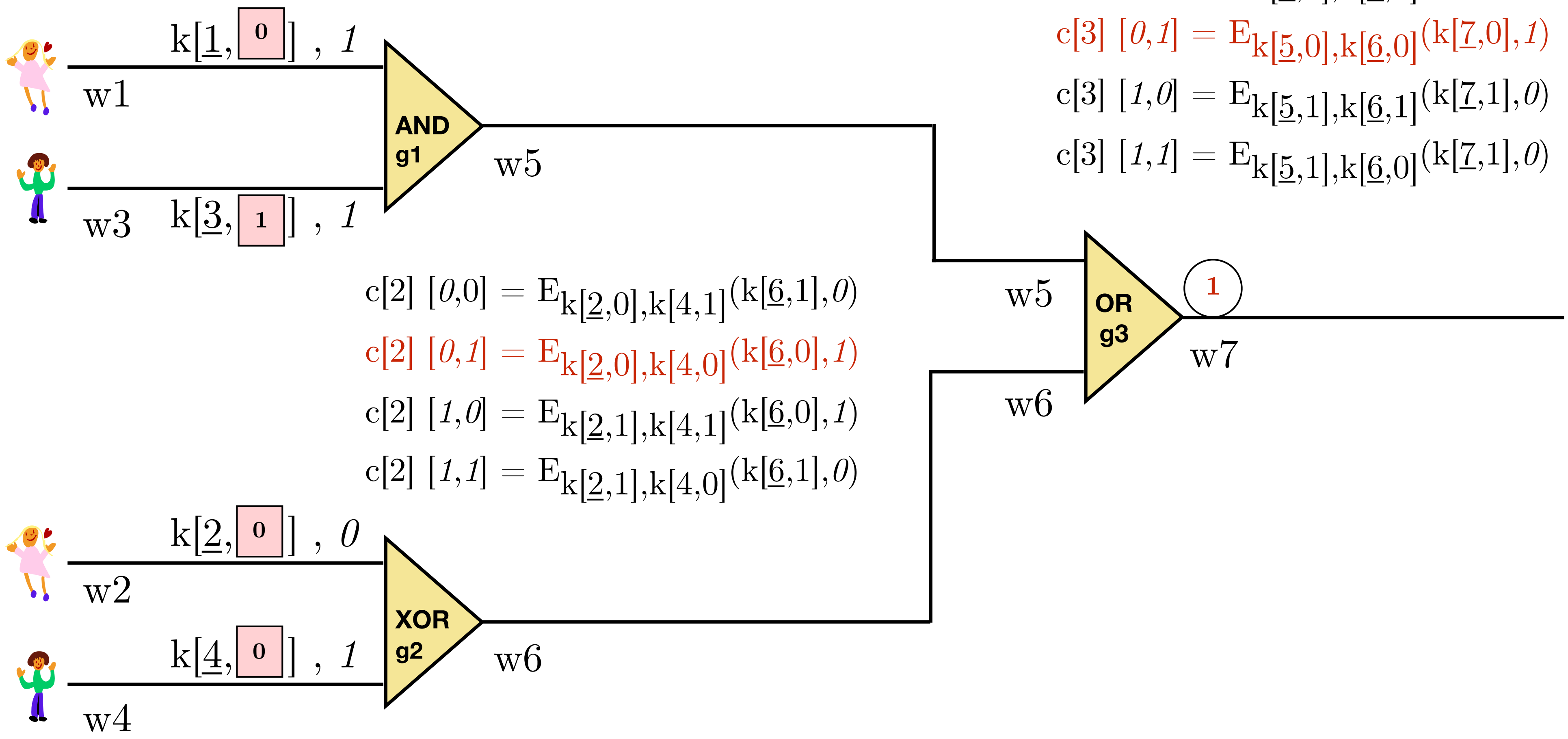
$$c[3]\ [\mathit{1},\mathit{1}] = E_{k[\underline{5},1],k[\underline{6},0]}(k[\underline{7},1],\mathit{0})$$

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

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

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

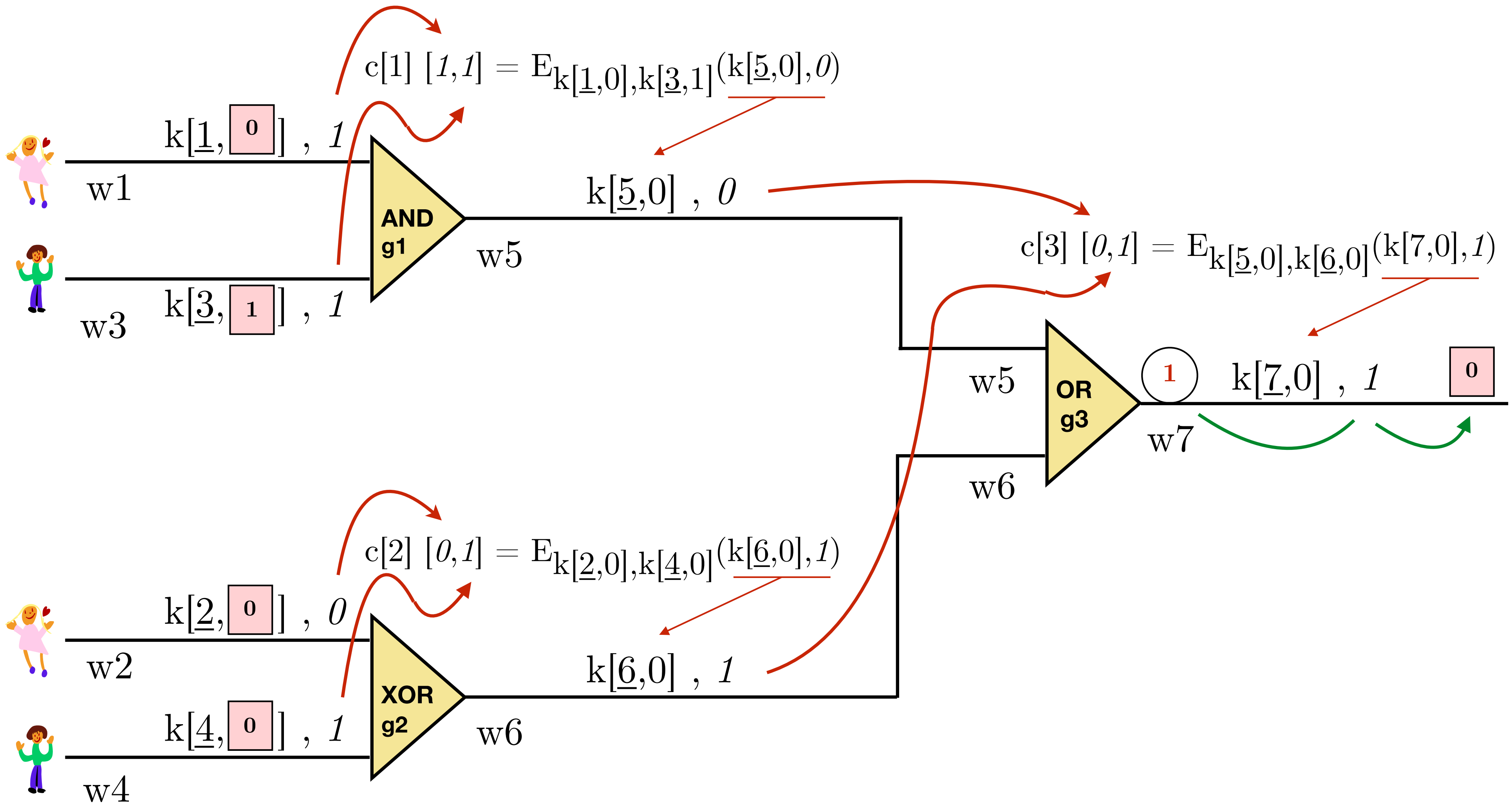
$$c[2]\ [\mathit{1},\mathit{1}] = E_{k[\underline{2},1],k[4,0]}(k[\underline{6},1],\mathit{0})$$



Garbled Circuit Evaluation by Bob

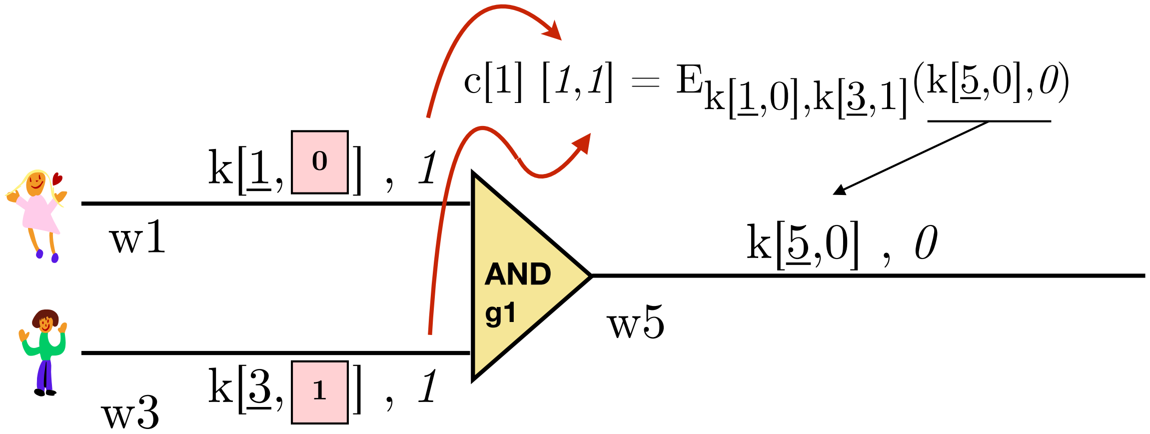
- For our example: **Alice's** inputs are v[1]= 0 , v[2]= 0

  Bob's inputs are v[3]= 1, v[4]= 0

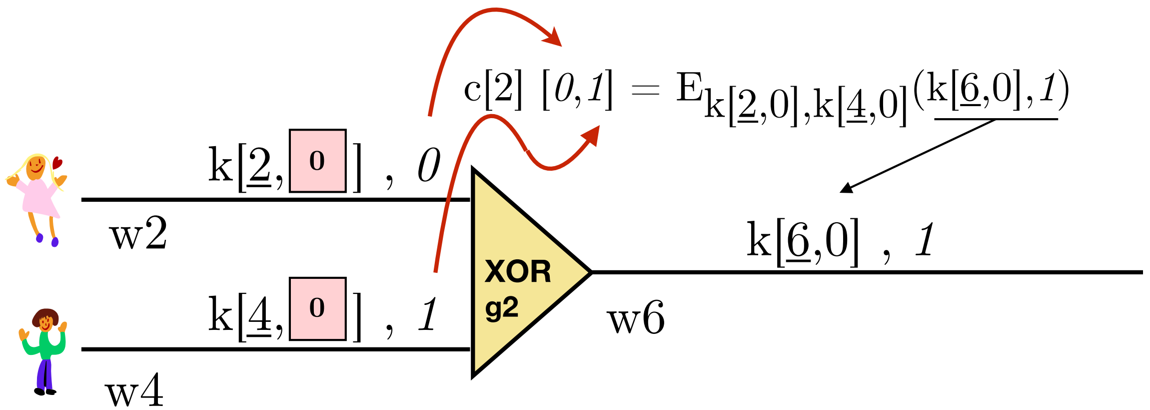


Garbled Circuit Evaluation 2

- For the AND gate Bob sees the encrypted value of w1 is *1*, 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 decrypts this using $k[\underline{1},0]$ and $k[\underline{3},1]$ to get the pair $k[\underline{5},0],\mathit{0}$



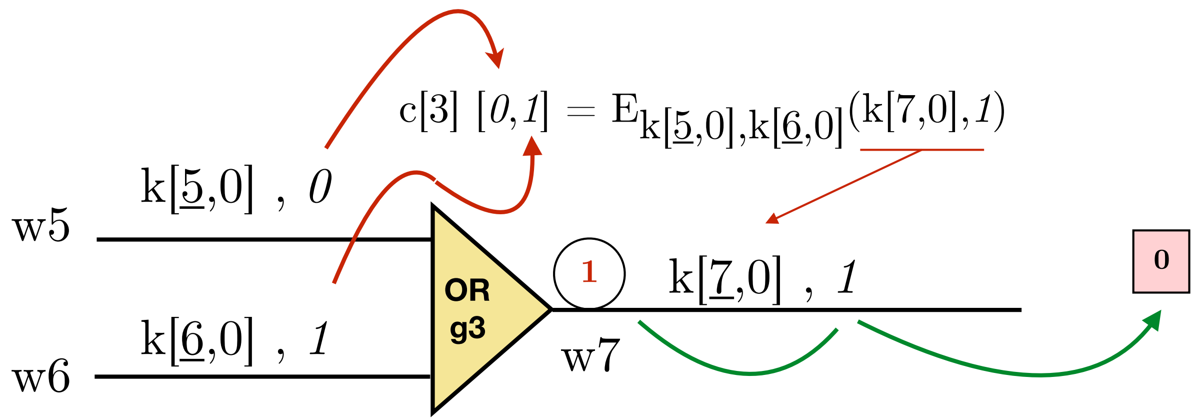
- For the XOR gate Bob sees the encrypted value of w2 is *0*, w4 is *1*.
  Bob uses these to index the gate's garbled table to get $E_{k[\underline{2},0],k[\underline{4},0]}(k[\underline{6},0],\mathit{1})$
  Bob decrypts this using $k[\underline{2},0]$ and $k[\underline{4},0]$ to get the pair $k[\underline{6},0],\mathit{1}$



Garbled Circuit Evaluation 3

- For the OR gate Bob sees the encrypted value of w5 is *0*, 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 decrypts this using k[5,0] and k[6,0] to get the pair k[7,0], *1*
  Bob can decrypt *1* using p[7] i.e. p[7] ⊕ *1* = 1 ⊕ *1* = 0

- So result of the circuit is 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

Yao's Protocol Summary

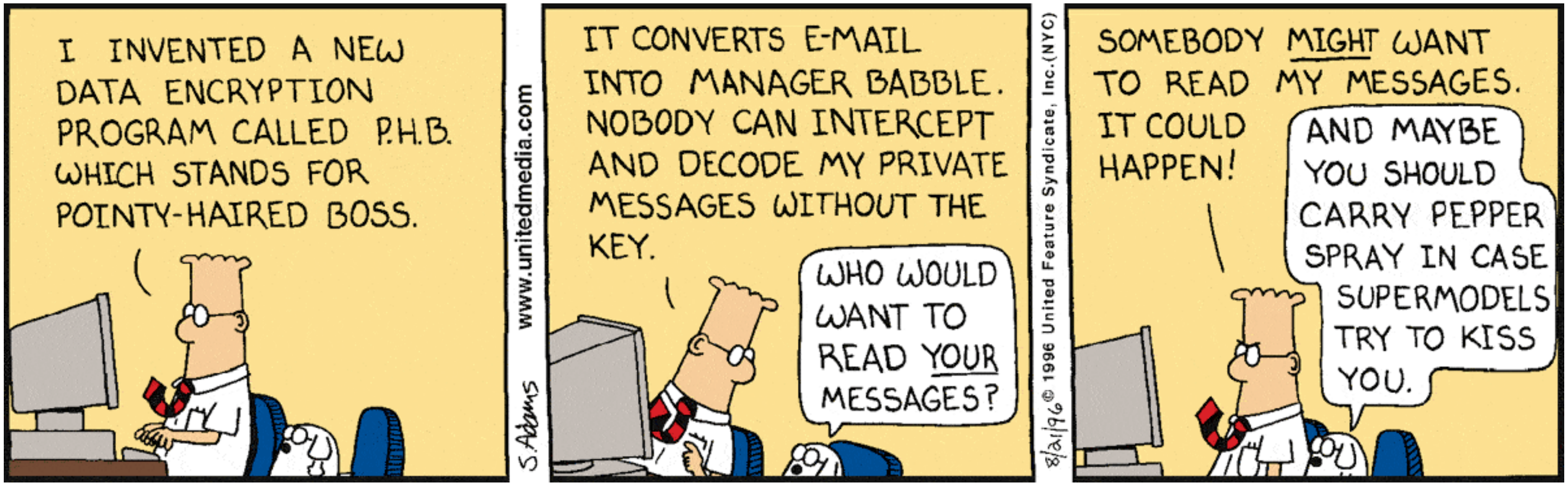
- If there are N inputs and G gates, we need 4xG encryption entries and N oblivious transfers.
- Both Alice and Bob could cheat at various stages. There are techniques to prevent cheating, e.g. zero-knowledge proofs at each stage or cut-and-choose, but those often have have high overheads.
- Circuits can be generated using compilers like Fairplay's SFDL and garbled tables optimised for memory in various ways.

Example performance circa 2009 (*Need to update to state-of-the-art implementations which are significantly better*)

- **Alice's input**: 128-bit AES key
  Bob's input: 128-bit message
  Output: Output of one round of AES.
- **Semi-honest adversary**:
  Time: 7 secs, **Gates**: 33K, 34% of gates required garbled table, rest XOR gates
- **Malicious adversary**:
  Time: 1114 seconds!, Gates: 45K, 25% of gates required garbled table.

Secure Two-Party Computation is Practical, Pinkas et al, AsiaCrypt 2009.

Babbling



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.