3. MPC - Garbled Circuits

Naranker Dulay

n.dulay@imperial.ac.uk
https://www.doc.ic.ac.uk/~nd/peng

Secret Sharing vs Garbled Circuits

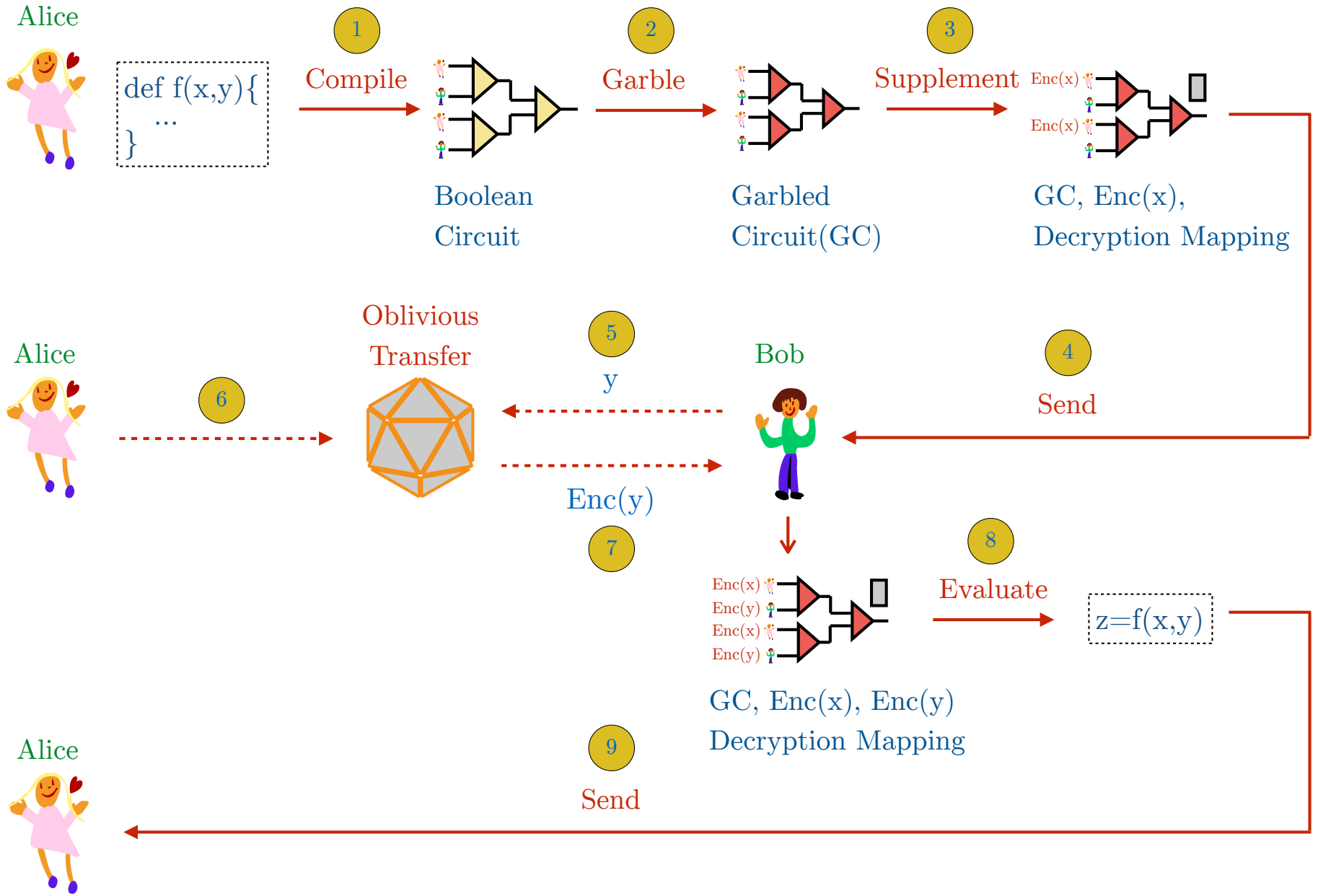
Secret Sharing (SS)

- 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

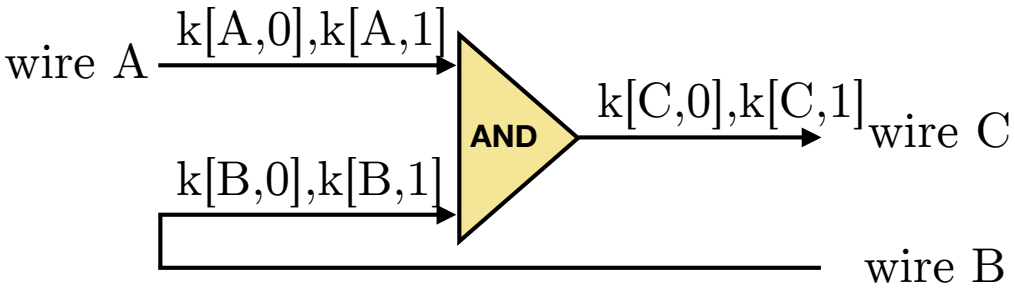
- Express the function to be computed as an "encrypted" boolean circuit, e.g. with ANDs, ORs, XORs, etc
- Garbled circuits are more efficient than for 2-party joint processing setups. They get too complex for more parties.
- We will look Yao's Garbled Circuits protocol for 2-parties.

Two-party Garbled Circuit Overview



One Gate Garbled Circuit (Alice)

Alice (Garbler)



Bob (Evaluator)

Garbled (Encrypted) Truth Table for AND

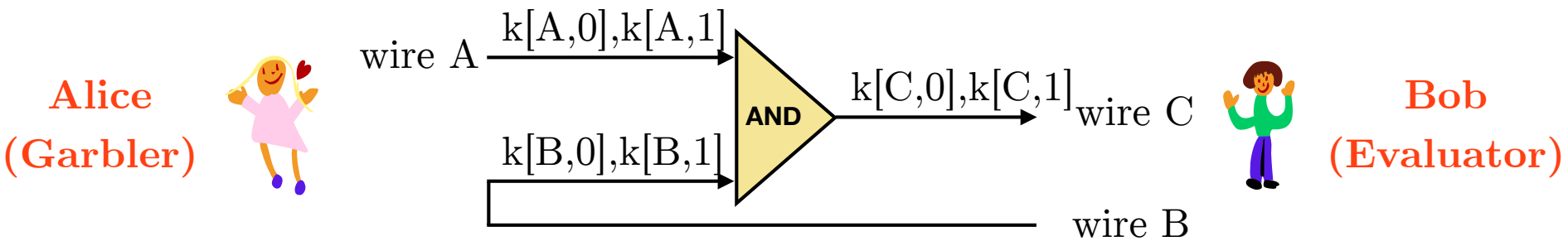
$$GT\ [0,0] = E_{k[A,0],k[B,0]}\ (\ k[C,AND(0,0)]\ )$$

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

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

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

Randomly Permutate GT (Alice)



Garbled (Encrypted) Truth Table for AND → Randomly permutated Garbled Table for AND

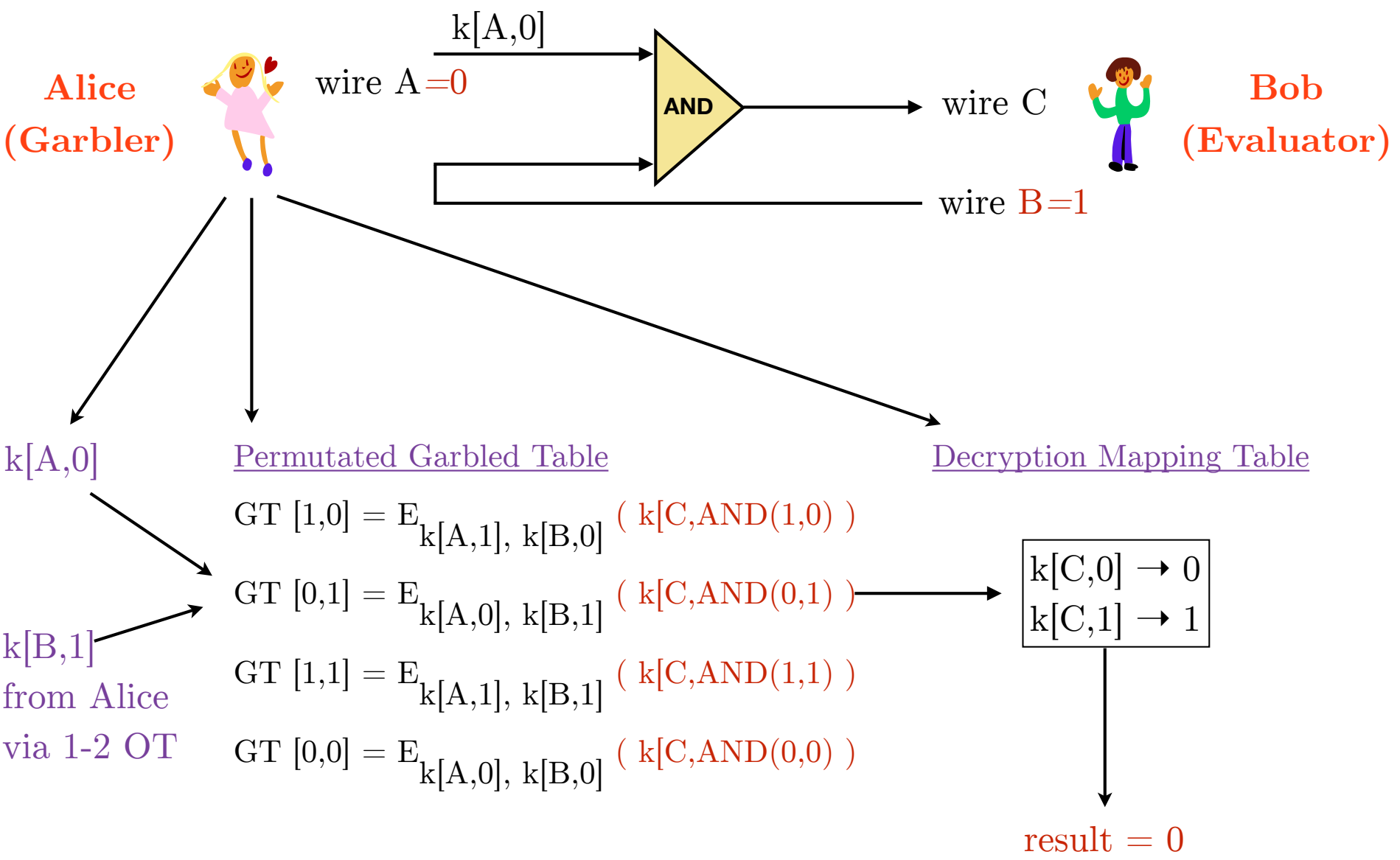
$$GT\ [0,0] = E_{k[A,0],k[B,0]}\ (\ k[C,AND(0,0)\ ) \qquad GT\ [1,0] = E_{k[A,1],k[B,0]}\ (\ k[C,AND(1,0)\ )$$

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

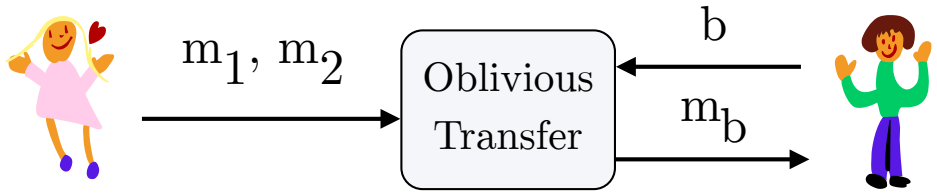
$$GT\ [1,0] = E_{k[A,1],k[B,0]}\ (\ k[C,AND(1,0)\ ) \qquad GT\ [1,1] = E_{k[A,1],k[B,1]}\ (\ k[C,AND(1,1)\ )$$

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

1-gate GC: Evaluation (Bob)



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

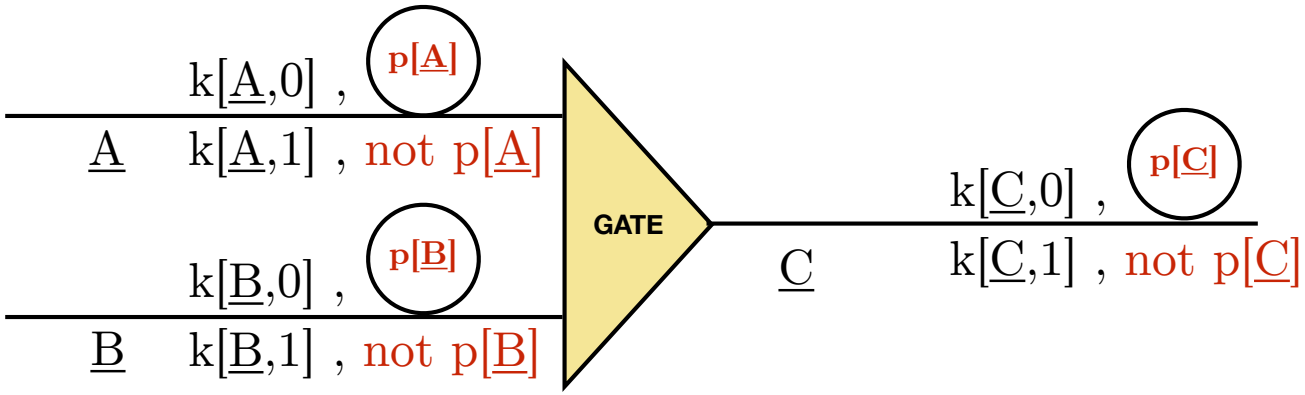
Bob → Alice: $c=E_{Pub1}(k)$ — Alice does $D_{Priv1}(c)= k$ Bob's key
and $D_{Priv2}(c)=u$ some random value

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

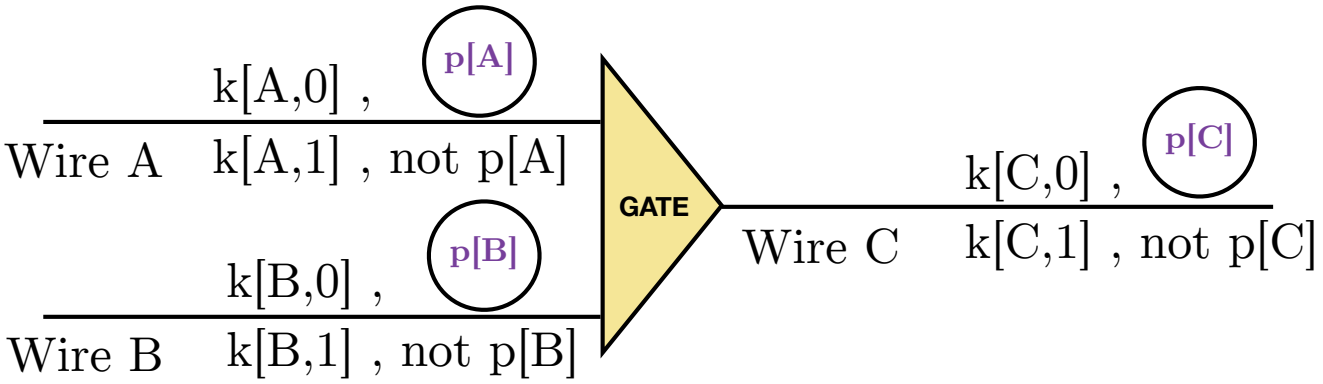
Point-and-Permutate p-bit

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:



Permutated Garbled Table Construction



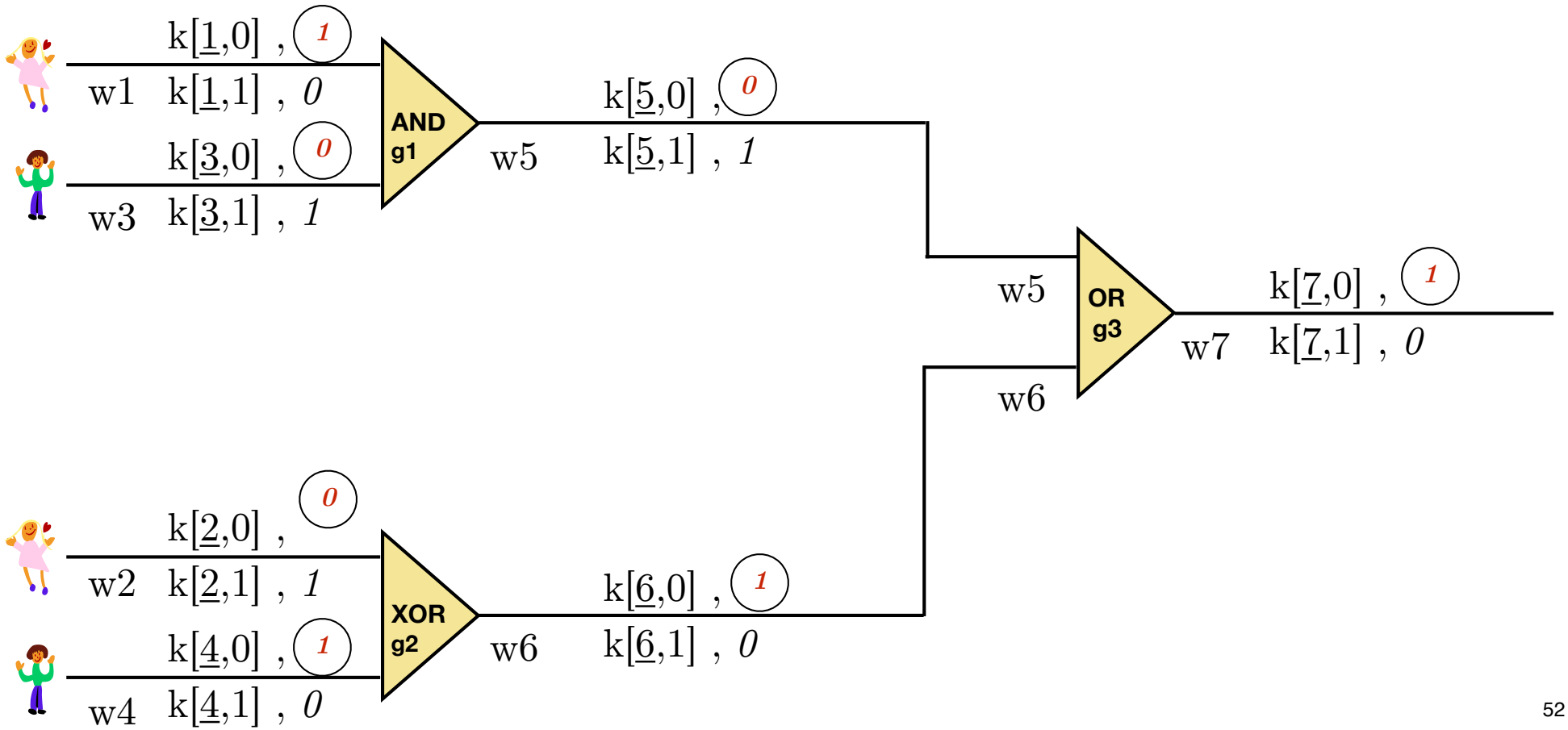
[a,b]	x=a ⊕ p[A], y=b ⊕ p[B]		
[0,0]	x=0 ⊕ p[A], y=0 ⊕ p[B]	x=a ⊕ p[A] y=b ⊕ p[B] z=GATE(x, y) z'=z ⊕ p[C]	$E_{k[A,x],k[B,y]}(k[C,z], z')$
[0,1]	x=0 ⊕ p[A], y=1 ⊕ p[B]		
[1,0]	x=1 ⊕ p[A], y=0 ⊕ p[B]		
[1,1]	x=1 ⊕ p[A], y=1 ⊕ p[B]		

Example: Point-and-Permutate p-bit

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

p[1]=p[4]=p[6]=p[7]= (1) and p[2]=p[3]=p[5]= (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]).



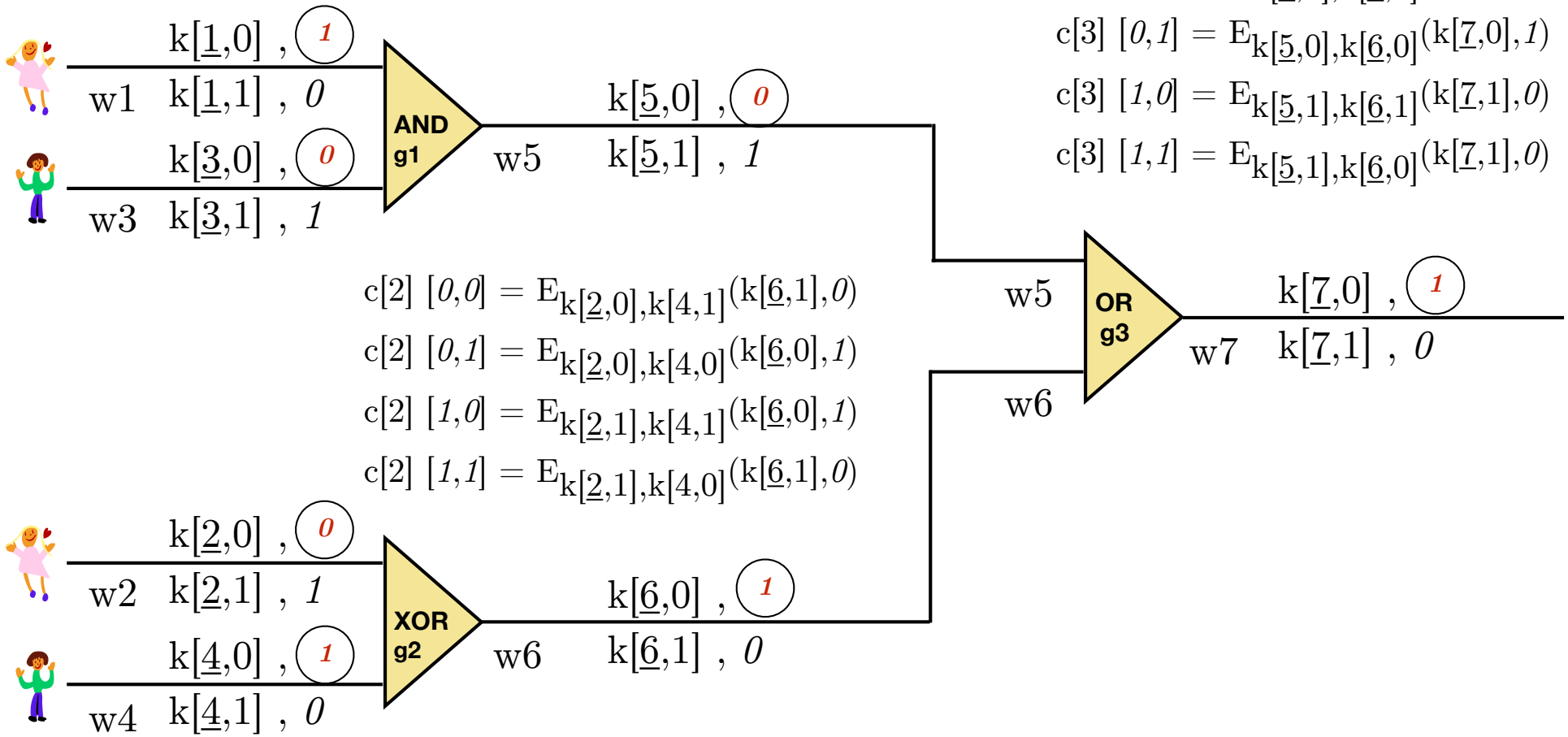
Example: Permutated Garbled Tables

Garbled tables for p[1]=p[4]=p[6]=p[7]= (1) and p[2]=p[3]=p[5]= (0)

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

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

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



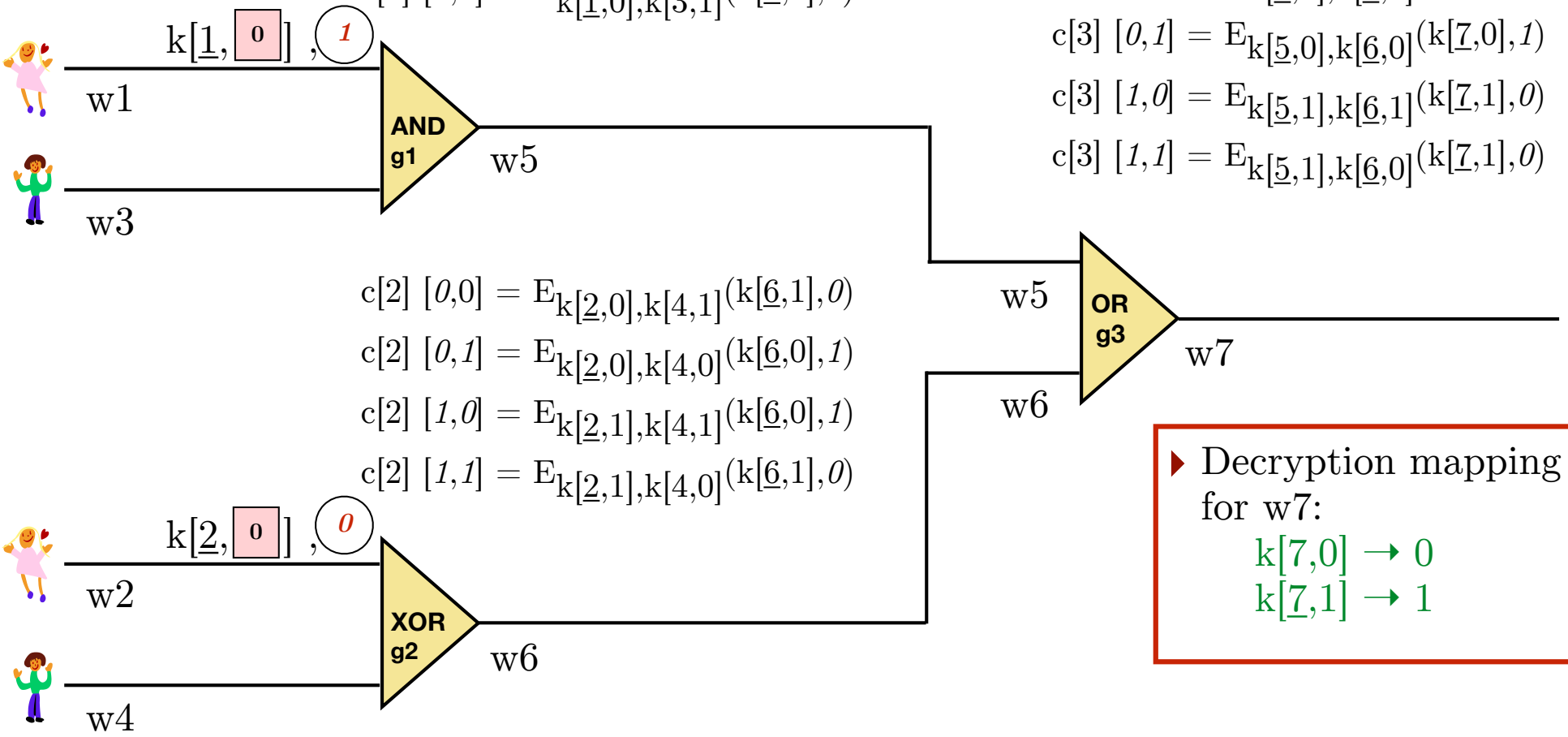
Circuit Transfer (Alice to Bob)

- For plaintext inputs **Alice:** v[1]= 0 , v[2]= 0 **Bob:** v[3]= 1 , v[4]= 0

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

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

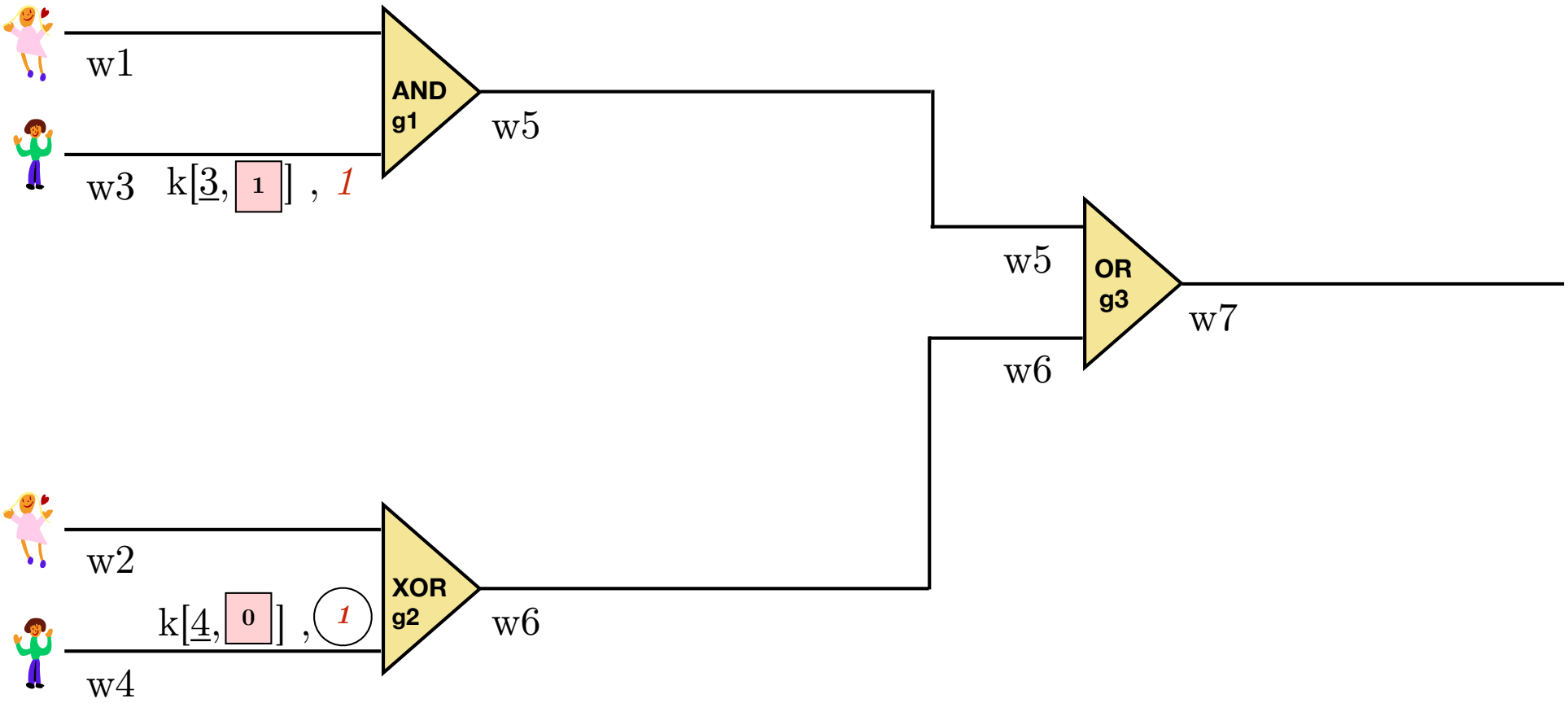
$$c[2]\ [0,0] = E_{k[2,0],k[4,1]}(k[6,1],0)$$
$$c[2]\ [0,1] = E_{k[2,0],k[4,0]}(k[6,0],1)$$
$$c[2]\ [1,0] = E_{k[2,1],k[4,1]}(k[6,0],1)$$
$$c[2]\ [1,1] = E_{k[2,1],k[4,0]}(k[6,1],0)$$



- Decryption mapping for w7:
  - k[7,0] → 0
  - k[7,1] → 1

Oblivious Transfers (Alice and Bob)

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

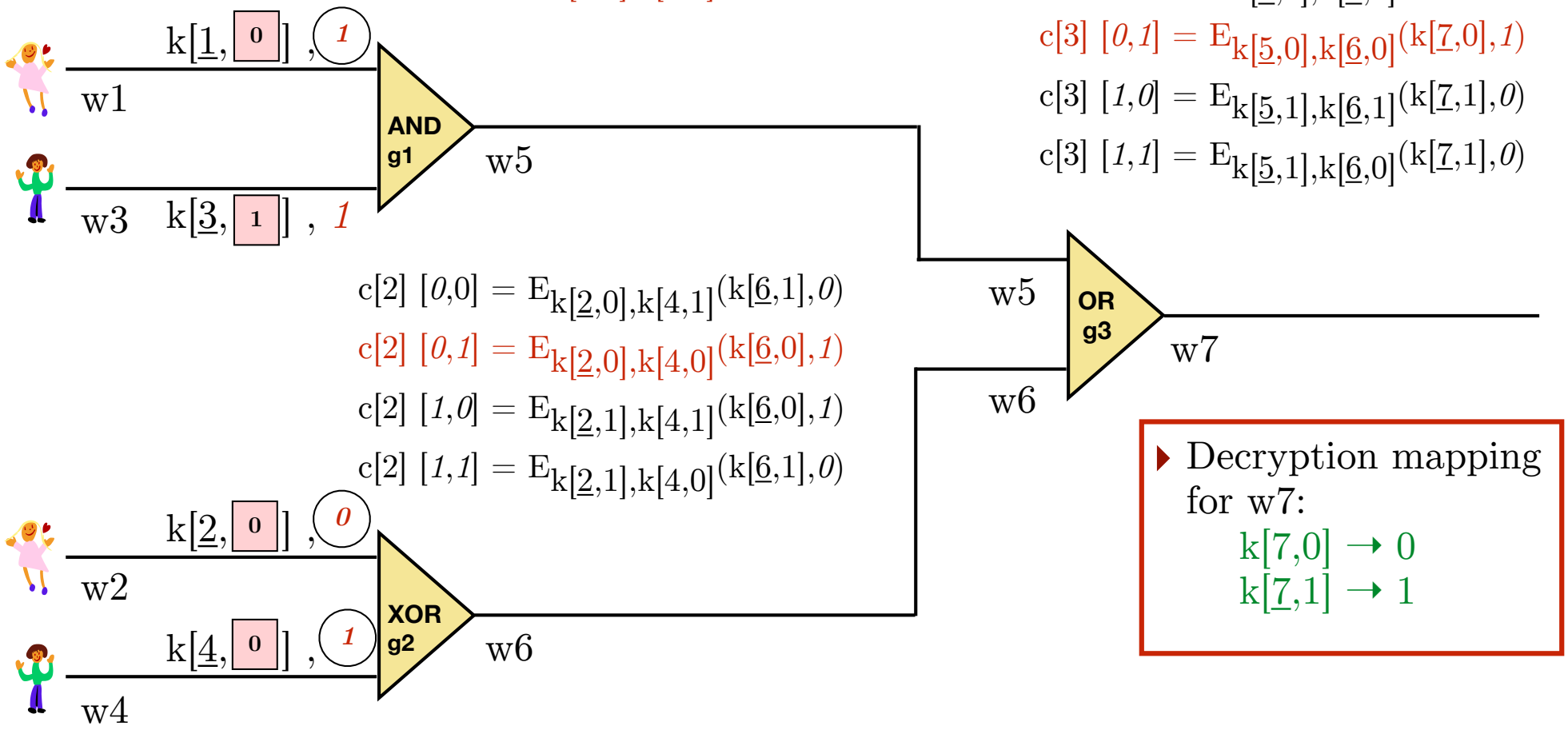


Garbled Circuit (known by Bob)

- On completion of the Oblivious Transfers Bob knows:

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

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

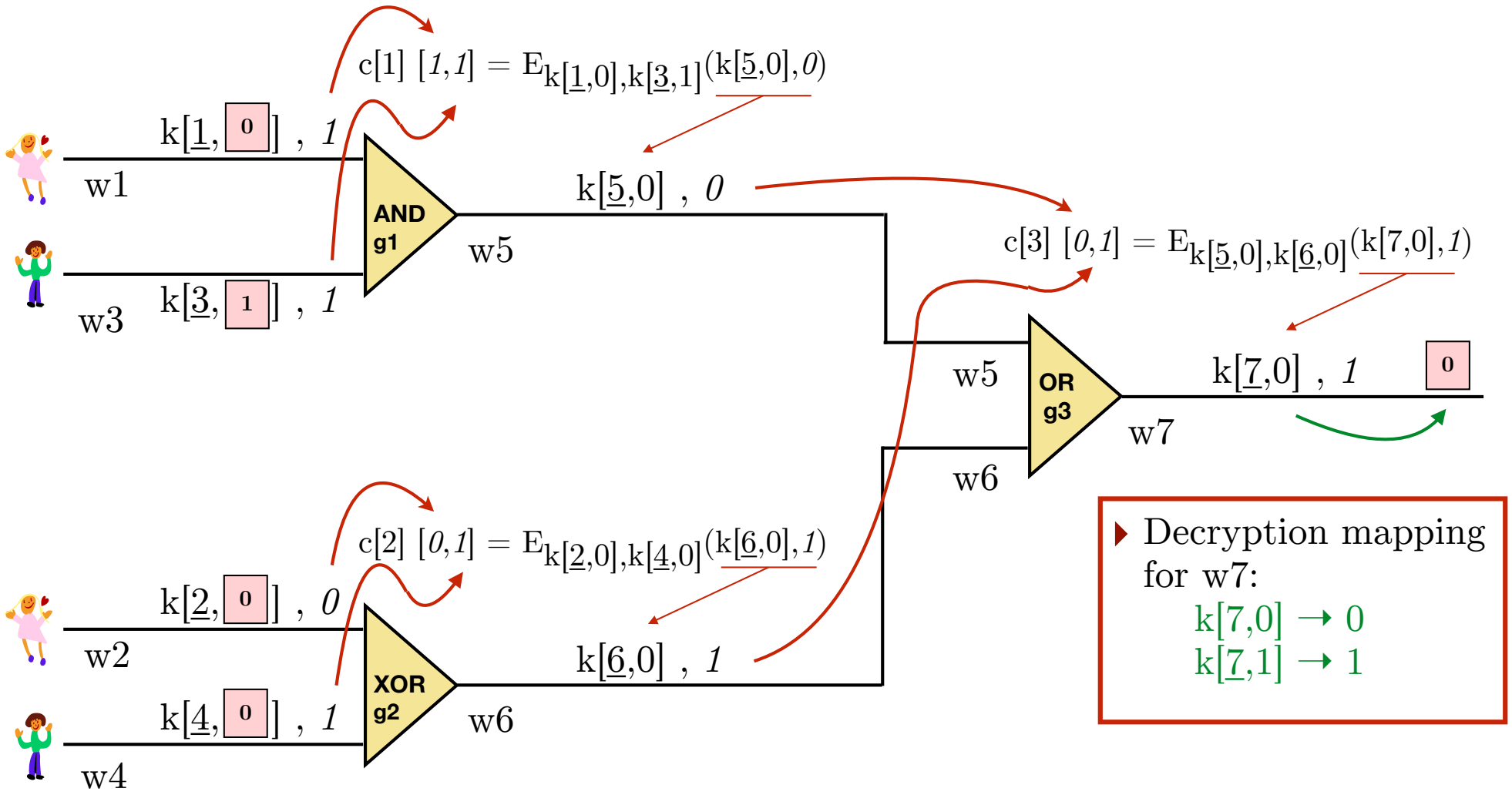


$$c[2]\ [0,0] = E_{k[2,0],k[4,1]}(k[6,1],0)$$
$$c[2]\ [0,1] = E_{k[2,0],k[4,0]}(k[6,0],1)$$
$$c[2]\ [1,0] = E_{k[2,1],k[4,1]}(k[6,0],1)$$
$$c[2]\ [1,1] = E_{k[2,1],k[4,0]}(k[6,1],0)$$

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

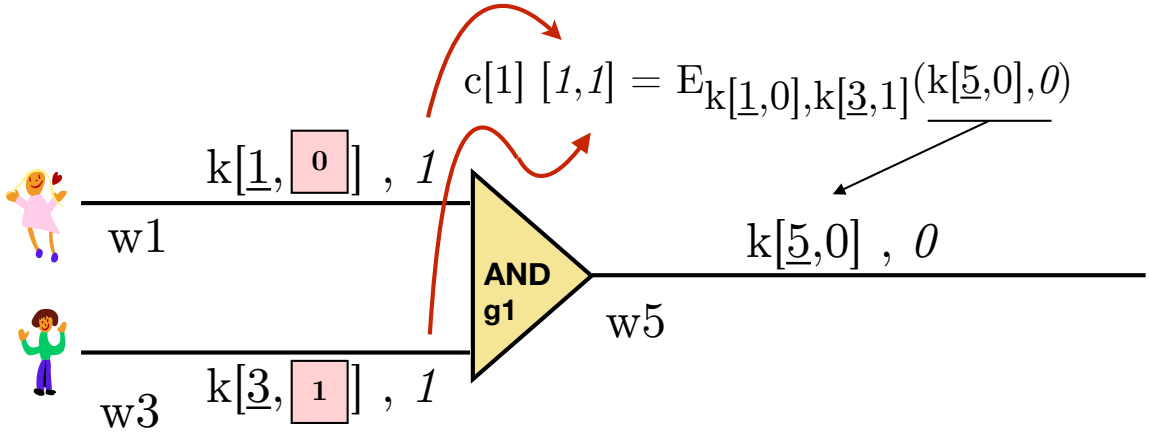
Garbled Circuit Evaluation by Bob

- Plaintext bits **Alice:** v[1]= 0 , v[2]= 0 **Bob:** inputs v[3]= 1 , v[4]= 0

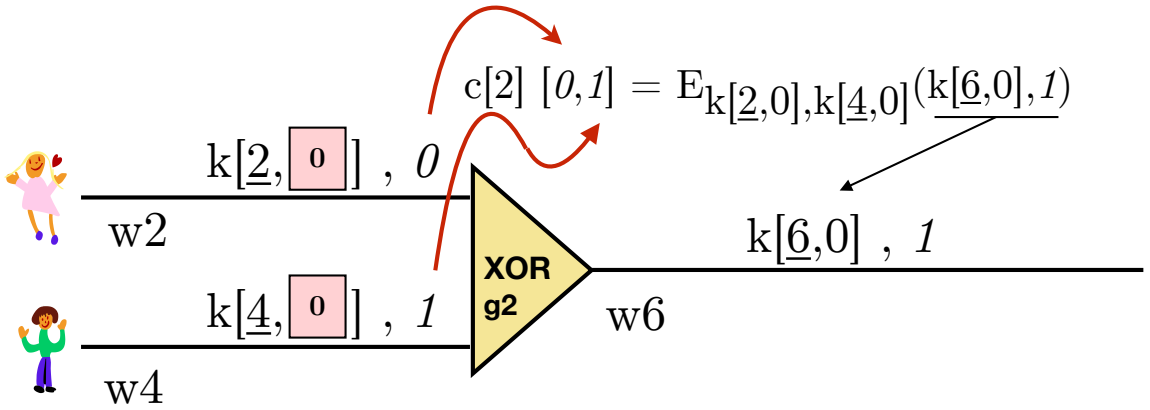


Garbled Circuit Evaluation 2

- 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 $E_{k[1,0],k[3,1]}(k[5,0],0)$
  Bob then decrypts this using k[1,0] and k[3,1] to get the pair k[5,0],*0*

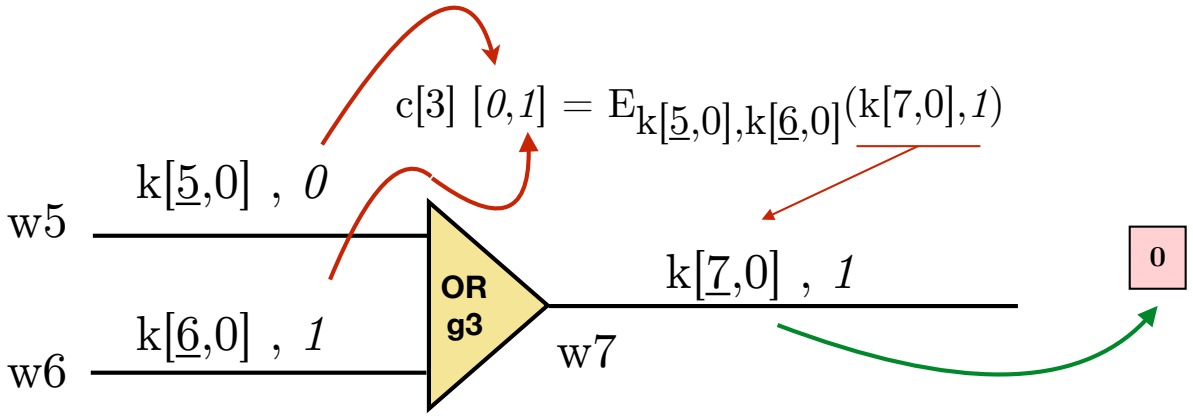


- For the XOR gate Bob sees the random p-bit element for w2 is *0*, and for w4 is *1*. Bob uses these to index the gate's garbled table to get $E_{k[2,0],k[4,0]}(k[6,0],1)$
  Bob then decrypts this using k[2,0] and k[4,0] to get the pair k[6,0],*1*



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 $E_{k[\underline{5},0],k[\underline{6},0]}(k[7,0],1)$
  Bob then decrypts this using k[5,0] and k[6,0] to get the pair k[7,0],*1*
- Finally, Bob uses the decryption mapping to determine that k[7,0] corresponds to 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

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.

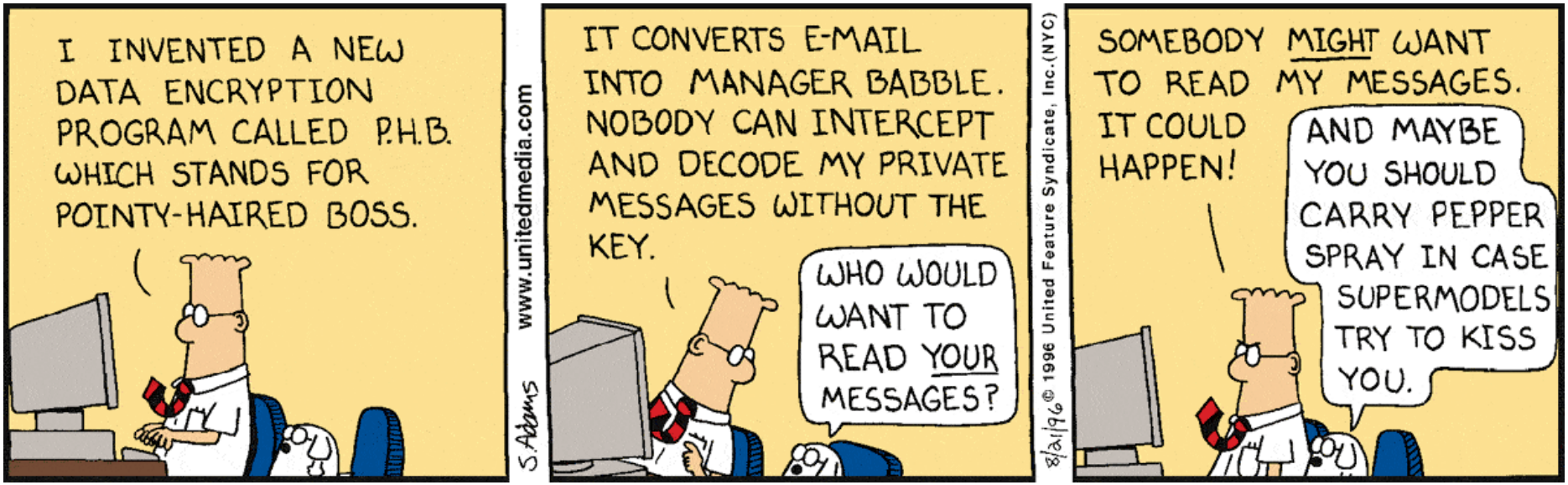
Compilation

- 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



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.***

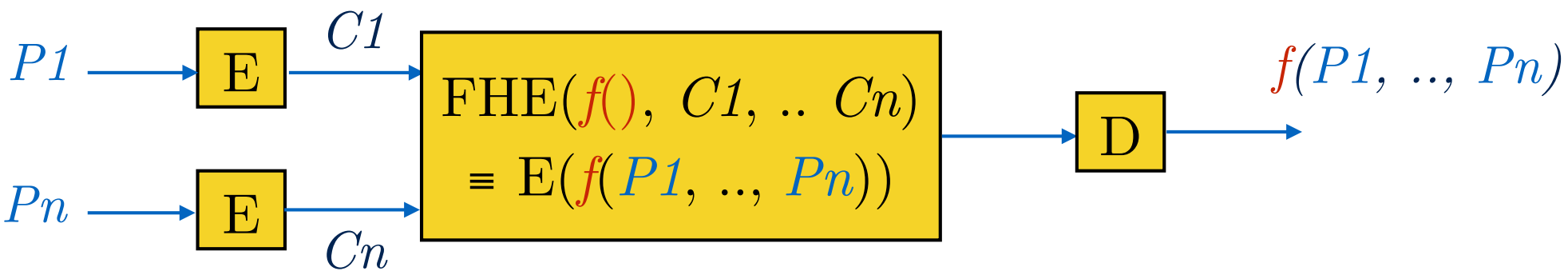
FHE allows anyone (e.g. untrusted cloud providers), not just the key holder, to output a ciphertext with the encrypted value of ***f(P1,.. Pn)*** for **ANY FUNCTION *f*.**

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: $z{=}x{\times}y$
  2. take their logs (E): $\log(x),\ \log(y)$
  3. add their logs (FHE): $\log(x){+}\log(y){=}\log(z)$
  4. take the anti-log (D): $z{=}\text{antilog}(\log(z))$

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

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

- 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
@___?        ?____@                    ?________________________ @
```

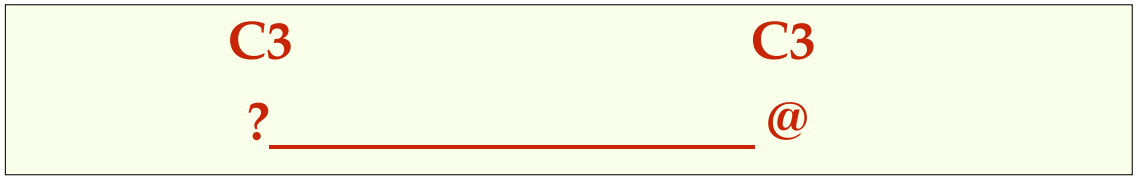
- **As more operations are performed, noise (i.e. errors) will grow leading to wrong results**

Gentry's FHE: A very informal explanation 2

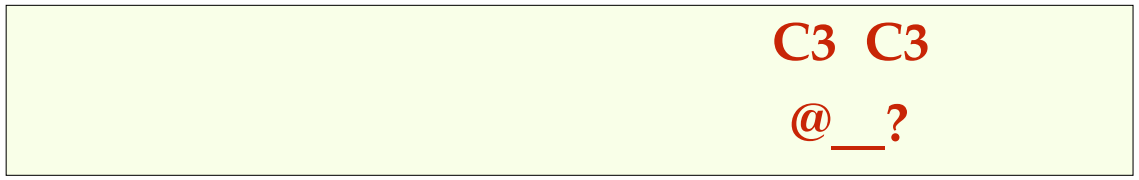
In order to control the noise, Gentry resets ciphertext values if they get too noisy (reach a noise threshold). **But how can the ciphertext be reset?**

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



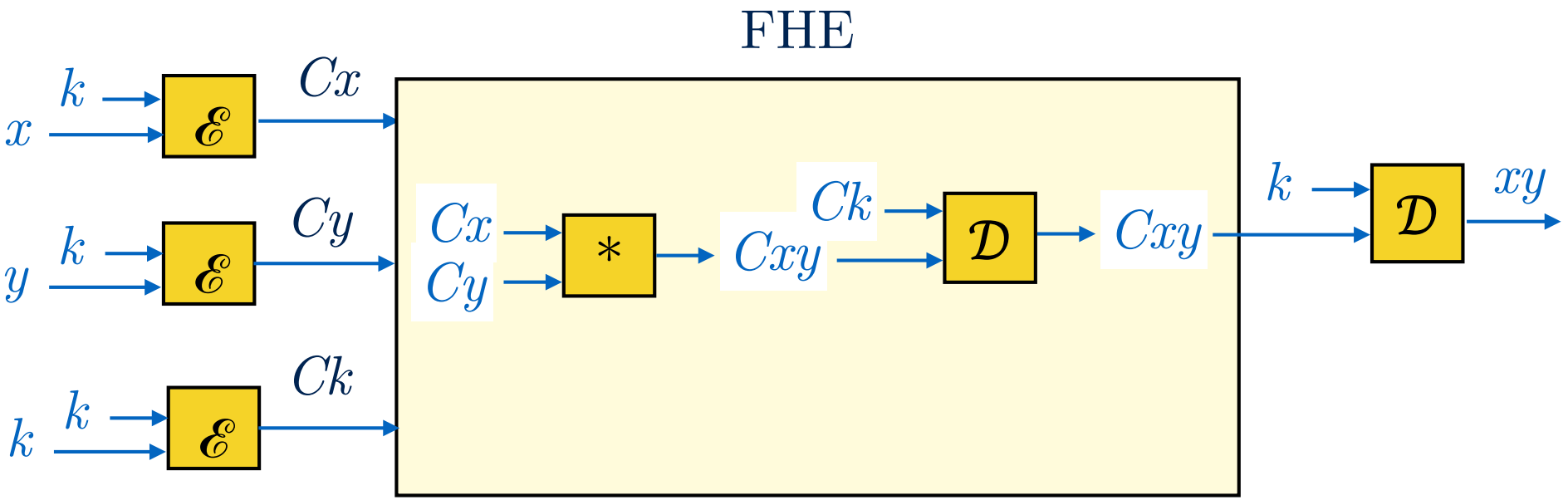
decrypting then re-encrypting C3 would become



- **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:



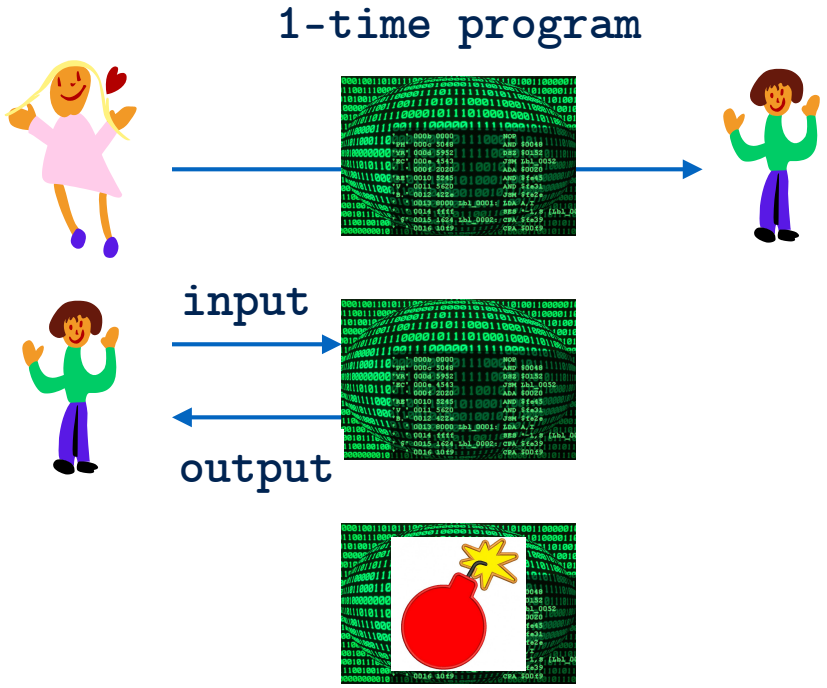
MPC vs FHE

	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?

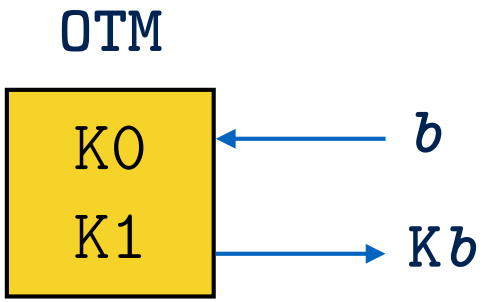
One-Time Programs

- 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)

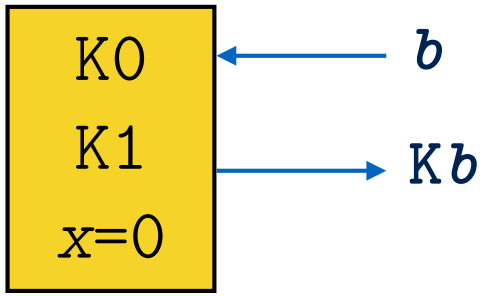
- Does not perform any computation.
- Tamperproof and can withstand ***side-channel attacks***.
- (1) Locations that are never accessed are never leaked by a side-channel
  (2) Locations that are accessed are immediately leaked
  (3) Also includes a single tamper-proof bit
- More specifically, GKR proposed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

Version 1



- 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.

Version 2

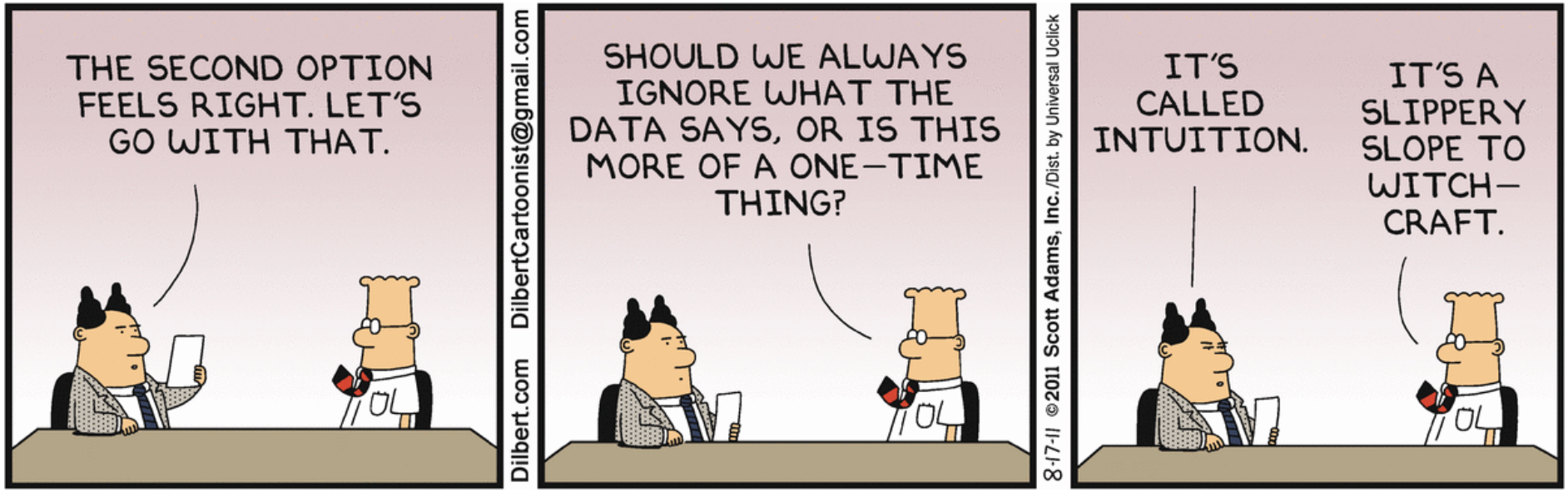


- Initialise OTM with two keys K0, K1 and a tamperproof bit x set to 0.
  if x=0 **then** {set x to 1, accept b, output Kb}
  elif x=1 **then** outputs error
  K$_{1\text{-}b}$ is never accessed in this version.

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