Practical Secure Remote Computation

Amir Herzberg

Haya Shulman

Talk Outline

- Basic two-party computation building blocks:
  - Yao’s garbled circuit
  - Oblivious transfer
- Secure two-party computation with offline trusted third party against malicious adversaries
- Fair two-party computation

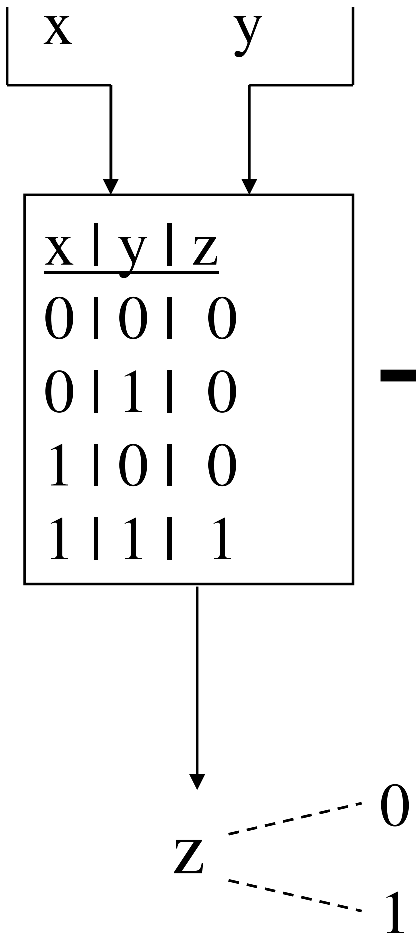
One-round secure computation and secure autonomous mobile agents

- Secure function evaluation (SFE):
  Alice has an input x and Bob has an input y.
  The goal is to compute $f(x,y)=(f_1(x,y),f_2(x,y))$ in one round computation, such that Alice obtains $f_1(x,y)$ and Bob obtains $f_2(x,y)$
- Any function computable by a poly sized circuit has a one-round secure computation scheme
  - Cryptographic tools:
    - Yao's garbled circuits
    - One-round oblivious transfer
- Security properties:
  - Privacy of the inputs supplied by local host
  - Integrity of the computation

Cryptographic Tools: Garbled Circuit

- Represent f(·,·) as a Boolean circuit
- Remote host "garbles" the circuit:
  - ∀ wire, assigns random strings representing 0/1
  - ∀ gate, constructs a "secure" garbled truth table
- Remote host sends to local the garbled tables and random strings corresponding to its input
- Local host uses (1-2)oblivious transfer to obtain garbled strings of its input
- Evaluates the garbled circuit, and obtains the result

Cryptographic Tools: Garbled Circuit Example: AND Gate Construction



x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

→

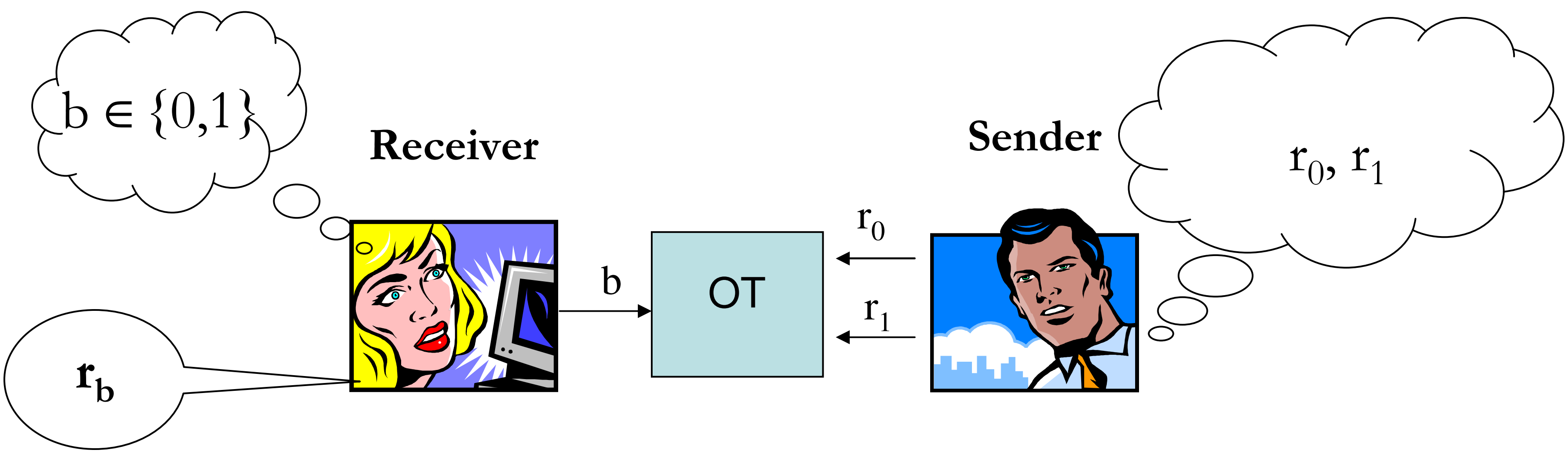
x	y	z
$k_x(0)$	$k_y(0)$	$k_z(0)$
$k_x(0)$	$k_y(1)$	$k_z(0)$
$k_x(1)$	$k_y(0)$	$k_z(0)$
$k_x(1)$	$k_y(1)$	$k_z(1)$

$$c_z(0,0)=E_{kx(0)}(E_{ky(0)}(k_z(0)))$$
$$c_z(0,1)=E_{kx(0)}(E_{ky(1)}(k_z(0)))$$
$$c_z(1,0)=E_{kx(1)}(E_{ky(0)}(k_z(0)))$$
$$c_z(1,1)=E_{kx(1)}(E_{ky(1)}(k_z(1)))$$

Permute rows

Cryptographic Tools: Oblivious Transfer

- Can be based on most public-key systems
- The sender has two inputs, and the receiver wants to learn one of them, at the end of the protocol:
  - the receiver learns this input and nothing else
  - the sender should not learn which input this was



Secure Two-Party Computation in Malicious Setting

- Secure Function Evaluating (SFE) based on garbled circuits is secure against semi-honest adversaries only
  - If Alice is malicious she can learn Bob's input by sending incorrect representation for one of Bob's input bits
- Theoretical solution: use a cut-and-choose protocol
  - Alice sends many circuits
  - Bob requests to expose all but one
    - If all constructed correctly Bob evaluates the garbled circuit on its secret input
  - Inefficient: cheating is detected with probability related to the number of garbled circuits

Secure Two-Party Computation with Offline *TTP* in Malicious Setting

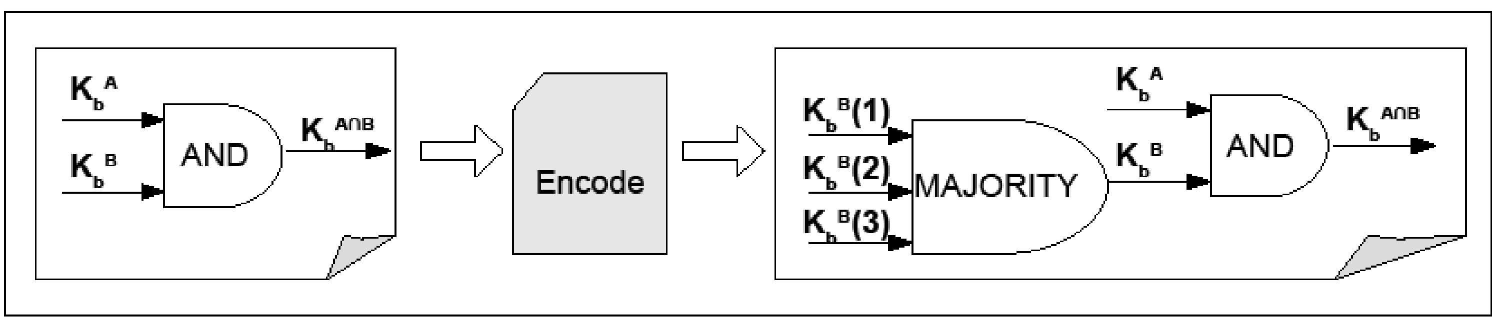
- Our solution: use *TTP* to construct and sign the garbled circuit computing a known function
- Offline generation phase, performed by *TTP*:
  - Constructs a circuit *C* computing a known function *f*, and garbles *C* to obtain *C'*
  - Generates a signature key-pair *(sk,vk)* and signs the garbled circuit
    - For each input of Bob: the signature is on the random string representing the input bit of Bob, the value of the bit and the index

Secure Two-Party Computation with Offline *TTP* in Malicious Setting

- Execution phase:
  - Alice sends Bob the garbled circuit
    - i.e., garbled tables, and strings for her inputs
    - Runs OT to transfer to Bob representation for his input
  - Bob evaluates the circuit on his input, and returns the result to Alice
- But, malicious Alice can cheat:
  - E.g., sends an incorrect representation of input 1 and correct for 0
  - Bob fails to evaluate if has input 1
  - Alice learns this input bit value
- Solution: *TTP* encodes circuit to ensure detection of malicious behaviour **without** exposing Bob's input

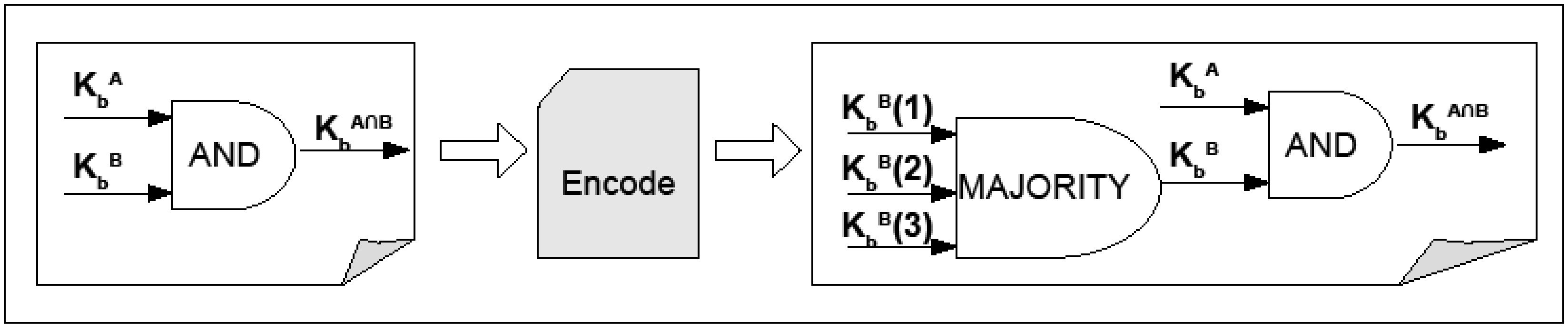
Secure Two-Party Computation with Offline *TTP* in Malicious

- Solution: *TTP* encodes circuit to ensure detection of malicious behaviour **without** exposing Bob's input
  - Each input wire of Bob at level 0 (input to the circuit) is replaced by a majority gate, e.g., 2 out of 3 (input values)
    - To read bit 0 Bob provides to OT two 0-es and one 1 (the correct majority values and complement for minority values)
  - Alice and Bob run OT for each input bit (to majority gate) of Bob



Encoded Circuit Construction

Each input wire at level zero is replaced by a majority gate



K_b^A	K_b^B	$K_b^A \wedge K_b^B = K_b^{B\wedge A}$
K_0^A	K_0^B	$K_0^{B\wedge A}$
K_0^A	K_1^B	$K_0^{B\wedge A}$
K_1^A	K_0^B	$K_0^{B\wedge A}$
K_1^A	K_1^B	$K_1^{B\wedge A}$

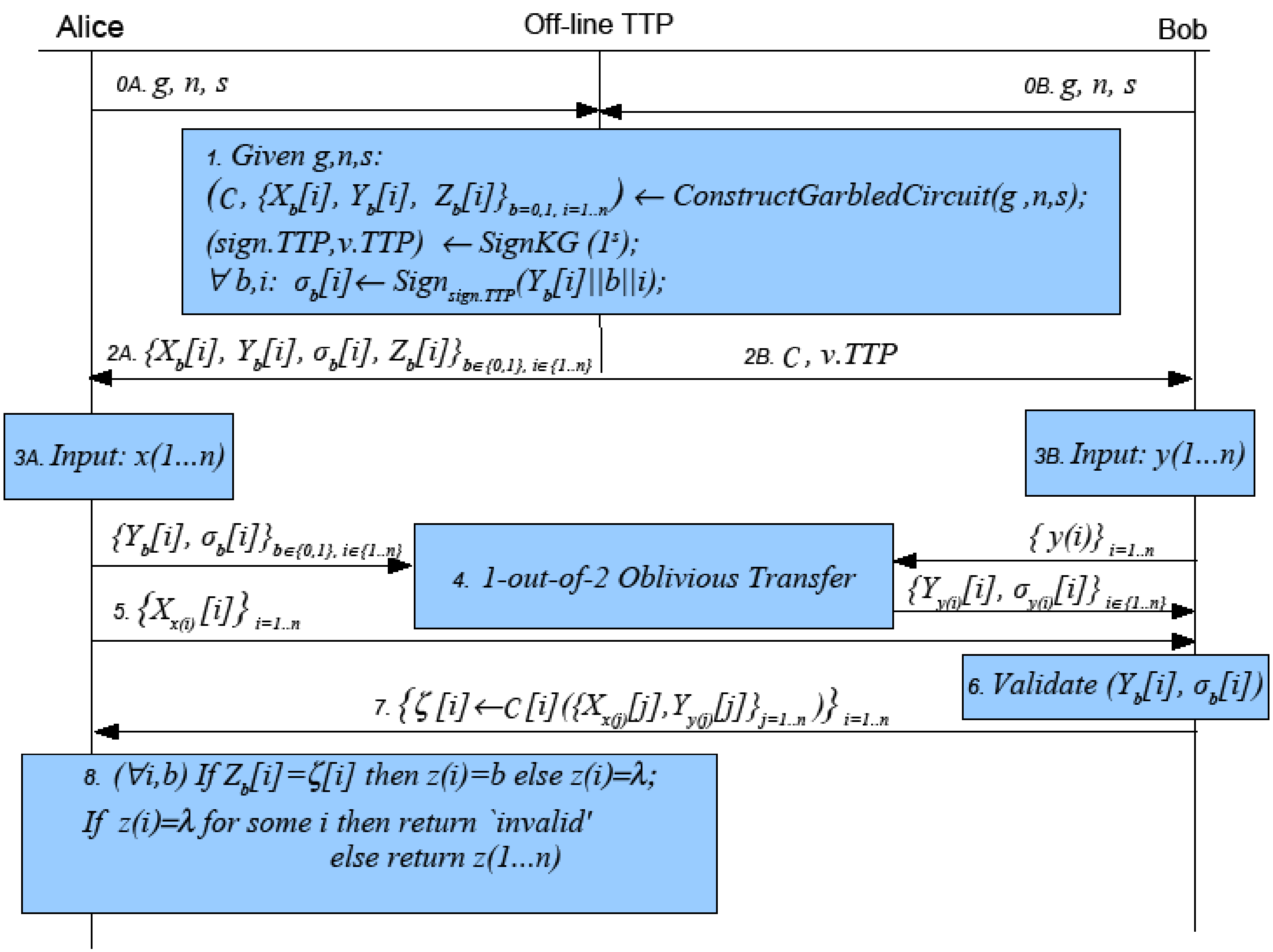
→

$K_b^B(1)$	$K_b^B(2)$	$K_b^B(3)$	K_b^B
$K_0^B(1)$	$K_0^B(2)$	$K_0^B(3)$	K_0^B
$K_0^B(1)$	$K_0^B(2)$	$K_1^B(3)$	K_0^B
$K_0^B(1)$	$K_1^B(2)$	$K_0^B(3)$	K_0^B
$K_0^B(1)$	$K_1^B(2)$	$K_1^B(3)$	K_1^B
$K_1^B(1)$	$K_0^B(2)$	$K_0^B(3)$	K_0^B
$K_1^B(1)$	$K_0^B(2)$	$K_1^B(3)$	K_1^B
$K_1^B(1)$	$K_1^B(2)$	$K_0^B(3)$	K_1^B
$K_1^B(1)$	$K_1^B(2)$	$K_1^B(3)$	K_1^B

Secure Two-Party Computation with Offline *TTP* in Malicious Setting

- Alice runs OT with Bob for each majority gate
  - E.g., three inputs to majority gate
  - Bob requests two bits that encode the value of its real input and requests one complement bit
- If Alice encoded incorrectly either bit this is detected, yet Bob's input is kept secret

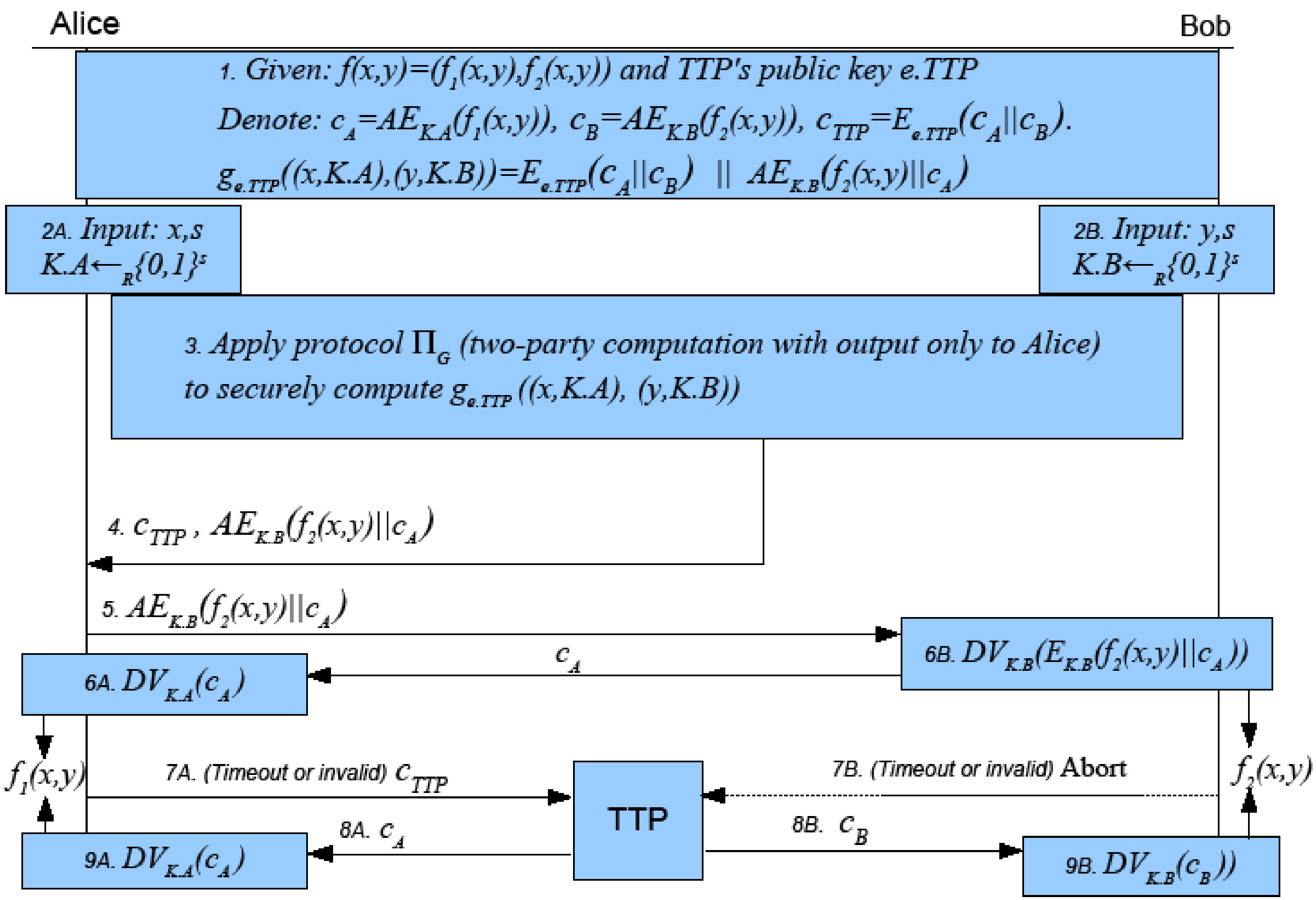
Secure Two Party Computation Against Malicious Adversaries



Fair Two Party Computation

- Alice has input x and Bob has input y and they wish to evaluate a common function f on their inputs, s.t.
  - Alice receives $f_1(x,y)$ and Bob receives $f_2(x,y)$
  - Or both receive failure
- Given a protocol Π_G to implement $g(x,y)=(z,\cdot)$ with output z at Alice only, construct a fair protocol Π_F to implement f using an offline trusted third party *TTP*

Fair Two Party Computation



Secure Fair Validated Computing

- Alice's input may be a program
- Validate the input of Alice to prevent execution of malicious program
  - e.g., exposing input of Bob
- Construct practical validated computing using the protocols presented before