Pietro Coronas 433043

Secure Computation tramite Garbled Circuits

Abstract

L'utilizzo congiunto di dati provenienti da fonti diverse porta con sé problematiche non indifferenti. La *"Secure Computation"* consente ai proprietari dei dati di mantenerli privati e al tempo stesso di essere utilizzati per i calcoli.

In questo caso consideriamo la "*Secure 2-party Computation*", in cui sono presenti 2 interlocutori che collaborano (un generatore e un valutatore) per calcolare l'output di una funzione f(a,b), mantenendo private le informazioni circa i loro ingressi a,b. Un possibile approccio alla *Secure 2-party Computation* sono i *Garbled Circuits di Yao*, che offrono un metodo per la computazione sicura dove più partecipanti devono calcolare una funzione senza che nessuno conosca l'input degli altri. Si basa su un modello semi-onesto: Si suppone che entrambe le parti eseguano il protocollo specificato, ma tentino al tempo stesso di imparare informazioni supplementari sugli ingressi privati della parte opposta.

In questa tesina verranno analizzati i principi generali del protocollo di *Yao*, verrà fatta un'analisi dei possibili attacchi e contromisure nel caso di *malicius model,* verranno valutati possibili metodi per migliorare il protocollo e infine un suo possibile utilizzo in sistemi cloud , sia nel caso del *semi-honest model* che nel *malicious model.*

- ## Introduzione

L'esempio dei due milionari chiarisce meglio il concetto che sta dietro la *Secure computation* : u1 e u2 vogliono determinare chi è più ricco, ma nessuno dei due vuole comunicare il suo esatto patrimonio. Grazie ai *Garbled Circuits* questo è possibile. Consideriamo una funzione f(x,y) tale che

- F(x,y)=1 se x<=y
- F(x,y)=0 altrimenti

L'obiettivo in questo caso, sia per u1 che per u2, è di imparare nulla di più di quello che si può dedurre dall' ingresso e dall'uscita di f .
Per poter comprendere a pieno il protocollo per *i Garbled Circuits* di Yao, dobbiamo introdurre alcuni concetti di crittografia.

- *Generazione di numeri pseudo-casuali*

Assumiamo che tutti i terminali coinvolti siano in grado di generare una sequenza con, approssimativamente, le stesse proprietà statistiche di una sequenza di numeri generata da un processo casuale.

- *Oblivious Transfer (OT)*

In un protocollo OT, un mittente A offre al ricevitore B una scelta di due messaggi da ricevere. B seleziona uno dei due messaggi, in modo che siano soddisfatte le seguenti condizioni:

  - B non conosce nulla del messaggio che non ha selezionato
  - A non sa quale messaggio sia stato selezionato da B

Quindi il nome *Oblivious Transfer ("trasferimento ignaro")* è appropiato, in quanto A non ha alcuna idea del messaggio ricevuto da B, e B non ha alcuna idea di quale fosse il messaggio che ha scartato.

• Yao's Garbled Circuit

Ora si hanno tutti gli strumenti per descrivere il protocollo. Sia n un parametro di sicurezza ed f(x_1,…, x_a; y_1,…,y_b) la funzione che P1 e P2 vogliono calcolare. x_i ed $y_i$sono i rispettivi ingressi di P1 e P2. P1 è il *generatore* mentre p2 il *valutatore*.

- *Circuit Diagram*

La prima operazione che P1 fa su f è effettuare *l'hard coding* dei suoi ingressi A_1,…, A_a in essa, creando quindi la funzione: g(y_1,…,y_b) =f(A_1,…, A_a; y_1,…,y_b) Dobbiamo garantire che P2 non intuisca ciò che questa funzione fa.

Poiché g può essere scritta come un programma eseguibile su un calcolatore, possiamo descrivere un *Circuit Diagram* per g costituito da connessioni e porte logiche. Assumiamo che ogni porta logica abbia in ingresso due connessione e in uscita una solamente. Se una connessione ha più diramazioni per fornire ingresso a più porte, viene considerata ancora come singola connessione.

- *Garbling the circuit*

Il passo successivo è creare la versione "*garbled* "del circuito. Inizialmente P1 genera i valori della porta normalmente ottenendo una tabella di verità. Successivamente, per ogni connessione W_I e per ogni possibile input 0 o 1, P1 crea due chiavi casuali K_I^0 E K_I^1 ciascuna di lunghezza n. L'idea è che la chiave li K_I^0 corrisponderà alla connessione Wì a cui è assegnato il valore di 0 e viceversa K_I^1corrisponderà alla connessione W_I a cui è assegnato il valore di 1. In questo caso P1 genera 6 chiavi, una per ciascuno dei due possibili valori booleani presenti nelle 3 connessioni della porta.

Successivamente P1 cripta ogni entry della tabella per la connessione di uscita tramite le chiavi utilizzate per i corrispondenti input. Inoltre cambia l'ordine delle righe della tabella oscurando ulteriormente i valori. Dato che l'output di ogni operazione di cifratura si assume essere random (criptata con hash), rimuove ogni correlazione tra i reali valori della tabella di verità e i risultanti valori criptati.

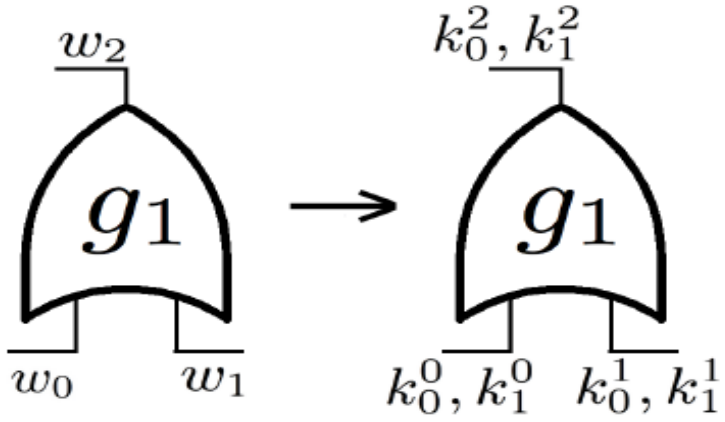


Figure 1: Garbling a single gate

w_0	w_1	w_2
0	0	0
0	1	1
1	0	1
1	1	1

(a) Original Values

w_0	w_1	w_2	garbled value
k_0^0	k_1^0	k_2^0	$H(k_0^0\|\|k_1^0\|\|g_1) \oplus k_2^0$
k_0^0	k_1^1	k_2^1	$H(k_0^0\|\|k_1^1\|\|g_1) \oplus k_2^1$
k_0^1	k_1^0	k_2^1	$H(k_0^1\|\|k_1^0\|\|g_1) \oplus k_2^1$
k_0^1	k_1^1	k_2^1	$H(k_0^1\|\|k_1^1\|\|g_1) \oplus k_2^1$

(b) Garbled Values

Figure 2: Computation table for g_1^{OR}

w_3	w_4	w_5
0	0	0
0	1	0
1	0	0
1	1	1

(a) Original Values

w_3	w_4	w_5	garbled value
k_3^0	k_4^0	k_5^0	$H(k_3^0\|\|k_4^0\|\|g_2) \oplus k_5^0$
k_3^0	k_4^1	k_5^0	$H(k_3^0\|\|k_4^1\|\|g_2) \oplus k_5^0$
k_3^1	k_4^0	k_5^0	$H(k_3^1\|\|k_4^0\|\|g_2) \oplus k_5^0$
k_3^1	k_4^1	k_5^1	$H(k_3^1\|\|k_4^1\|\|g_2) \oplus k_5^1$

(b) Garbled Values

Figure 4: Computation table for g_2^{AND}

w_2	w_5	w_6
0	0	0
0	1	1
1	0	1
1	1	0

(a) Original Values

w_2	w_5	w_6	garbled value
k_2^0	k_5^0	k_6^0	$H(k_2^0\|\|k_5^0\|\|g_3) \oplus k_6^0$
k_2^0	k_5^1	k_6^1	$H(k_2^0\|\|k_5^1\|\|g_3) \oplus k_6^1$
k_2^1	k_5^0	k_6^1	$H(k_2^1\|\|k_5^0\|\|g_3) \oplus k_6^1$
k_2^1	k_5^1	k_6^0	$H(k_2^1\|\|k_5^1\|\|g_3) \oplus k_6^0$

(b) Garbled Values

Figure 5: Computation table for g_3^{XOR}

- *Invio a P2*

Una volta che P1 ha finito di generare i GC, ha bisogno di criptare il suo input per la funzione, creando una mappatura tra i_{p1} e i suoi equivalenti cifrati. Il processo inizia sostituendo il primo bit dell'input con la chiave corrispondente alla connessione di input nel circuito corrispondente. Vediamo un esempio:

- Il primo bit di input di P1 e' nella connessione w_0 e il valore di $i_{p1(0)}$ è 1.
- P1 vuole selezionale k_0^1 come primo valore del suo input per il GC
- P1 ripete l'operazione per i restanti bit, ottenendo un "*garble input*"
- P1 invia il circuito *cg* e il suo *garble* input a P2

- *Ricezione dell'input di P2 tramite OT*

P2 riceve quindi *cg* e il *garble input* di P1, ma ha ancora bisogno della rappresentazione "*garbled*" del suo input per poter operare sul circuito (f(a,b), in questo caso P2 ha solamente f ed a). P2 quindi instaura una connessione OT con P1.In questo caso P1 è il mittente e il suo input sono i valori k_0^1 e k_1^1 (gli *input garbled* associati a quella connessione)mentre P2 è il ricevitore e il suo input è 0 o 1. Successivamente, P2 effettuerà altre connessioni OT con P1 per ogni suo valore di input.

- *Computazione del GC*

A questo punto P2 ha tutto il necessario per calcolare il circuito: i suoi *garbled input*, i *garbled inputs* di P1 e il *garbled circuit*. P2 inizia il calcolo: Per ogni porta di input, P2 inserisce i suoi input e gli input di p1 nelle rispettive connessioni e li usa come chiave per decriptare il valore di output dalla tabella di verità della porta.

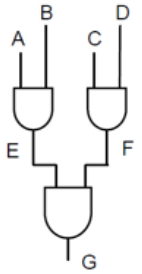
P2 però non sa a quali output corrispondano le chiavi di input e quindi tenterà di decifrare tutti e 4 gli output. Se il protocollo è stato eseguito correttamente, una sola chiave sarà decifrata correttamente e diventerà una chiave di input per la porta successiva.

P2 continuerà questa procedura finchè non raggiunge le connessioni di uscita del circuito. Ogni connessione di output ha un singolo bit in chiaro. P2 successivamente ri-assembla i bit di uscita ed ha la soluzione corretta per la funzione codificata dal *gc*. P2 completa il protocollo inviando l'output del circuito a p1

- *Riassunto*

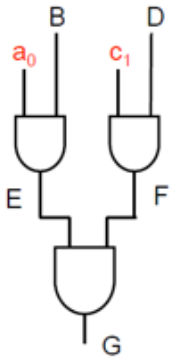
Chiariamo quindi le fasi principali del protocollo:

Fase 1



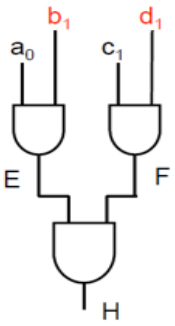
- P1 genera i garbled circuits.
- Genera una coppia di chiavi per ogni connessione
- Cripta l'output di ogni porta con queste chiavi
- Invia il GC a P2

Fase 2



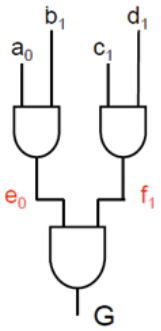
- P1 invia a p2 il suo input criptato

Fase 3



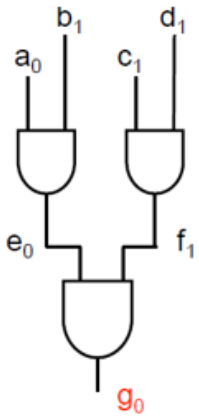
- Protocollo OT tra P1-p2
- P2 riceve il suo input criptato

Fase 4



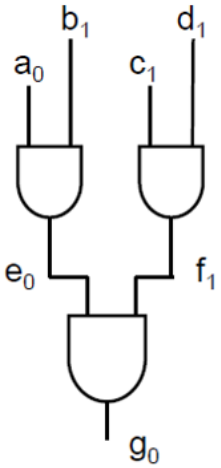
- P2 usa gli input per decriptare il primo strato del circuito

Fase 5



- P2 usa la stringa ottenuta precedentemente per decriptare lo strato successivo

Fase 6



- P2 valuta la funzione ed invia il risultato a P1

- ## Miglioramenti al protocollo

Il protocollo di *yao* è molto oneroso, in particolare i punti critici sono: La valutazione di tutta la tabella di verità per poter identificare il giusto valore, la creazione/valutazione/trasmissione di porte XOR e le dimensioni delle tabelle di verità. Molte tecniche sono state sviluppate per migliorare il protocollo, le principali sono:

- *Point and permute*: Permette al valutatore del circuito di individuare il giusto valore da decriptare nella tabella di verità criptata, risparmiandogli la decriptazione di tutta la tabella.
  Ogni valore della tabella consiste, oltre alla chiave, ad un bit di permutazione che è utilizzato dal valutatore per selezionare il corretto valore da decriptare.

- *Free XOR*: La creazione/trasferimento/valutazione di porte XOR in un *garbled circuit* è una delle operazioni più onerose. La tecnica quindi consiste nello generare porte XOR senza bisogno di criptazione.
  L'idea di base è assegnare un bit random R tale che per ogni connessione w_0(che rappresenta 0) sia campionata casualmente mentre la connessione w_1 (che rappresenta 1) sia settata semplicemente come w_0 XOR R. Per ogni porta XOR con due connessioni di input w_i, w_j e una di output w_k, lo 0 sulla connessione di uscita si possa derivare effettuando $w_k^0 = w_i^0$ XOR w_j^0, mentre
  $w_k^1 = w_i^0$ XOR w_j^0 XOR R.

- *Garbled Row Reduction*: Tecnica utilizzata per ridurre le dimensioni delle tabelle di verità per tutte le porta non XOR. Invece di definire 2 valori *garbled* distinti sulle connessioni di output, si definisce uno di essi come una funzione dipendente dai valori *garbled* in input. La tabella di verità risultante non conterrà quindi i valori per tutte le possibili combinazioni dei due ingressi. Se il valutatore ha il valore *garbled* della coppia di ingresso, potrà calcolare la corrispondente uscita direttamente senza consultare la tabella di verità.

- ## Vulnerabilità dei garbled circuit

L'implementazione base dei *Garbled Circuits*, come già detto prevede modello semi-onesto: Si suppone che entrambe le parti eseguano il protocollo specificato, ma tentino al tempo stesso di imparare informazioni supplementari sugli ingressi privati della parte opposta.

Purtroppo però ci si trova spesso in una situazione diversa, in cui uno dei partecipanti può deviare dal protocollo concordato in modi arbitrari. Questa è la situazione studiata in un modello Malevolo (*malicious model*). In questo caso l'avversario può compromettere la privacy dei dati degli altri partecipanti o manomettere la correttezza del risultato.

- *Malicious circuits generation*: Un generatore potrebbe costruire un circuito difettoso che svela l'ingresso privato degli altri partecipanti
- *Selective failure*: Durante l'OT, il generatore invia due valori di cui uno corrotto. La capacità del valutatore di completare il circuito rivelerà al generatore quale input è stato scelto.
- *Input inconsistency*: Attacco utilizzato durante la tecnica *cut-and-choose*: Durante la valutazione dei circuiti per verificare che la maggioranza sia corretta, è possibile che i partecipanti inviino input diversi per circuiti diversi, in modo da rivelare informazioni riguardo l'input dell'atro partecipante.

- ## Contromisure

Sono stati proposti molti approcci per rendere sicuri i *Garbled Circuit* in modelli malevoli.

1. *Cut and choose*

L'idea di base è di preparare più copie casuali del *garbled circuit.*

- Invece di inviare un solo GC, P1 invia n Garbled Circuits a P2.
- P2 chiede ad P1 di aprirne una parte.
- P1 a questo punto invierà a P2 la casualità che ha utilizzato nella creazione di tali circuiti. In questo modo P2 può controllare che i circuiti aperti siano stati creati correttamente (questi circuiti si chiamano *circuiti di controllo*).
- Se qualche circuito non è compatibile, viene interrotta la comunicazione.

- Se invece il controllo va a buon fine, P2 valuta i restanti circuiti (chiamati *circuiti di valutazione*) e prende come output finale l'output che compare nella maggioranza dei circuiti

2. *Commit and prove*

Prevede la costruzione di un singolo *Garbled Circuit* e delle pesanti operazioni di crittografia per ogni porta. Solitamente costoso in termini prestazionali, è stata creata una variante chiamata *Authenticated Bits*: prevede che solo i partecipanti che seguono il protocollo rimangono autenticati. In altre parole, se un partecipante malintenzionato devia dal protocollo concordato, l'altro partecipante se ne accorge ed abortisce la comunicazione. *L'Authenticated Bits* può essere implementato come un'estensione del protocollo OT.

3. *Committing OT*

Nozione più forte del normale protocollo OT contro i *failure attack*: Durante l'OT, il generatore invia due valori di cui uno corrotto per conoscere l'input del partecipante. Questo protocollo (COT) prevede il controllo di entrambi i valori prima del loro utilizzo.

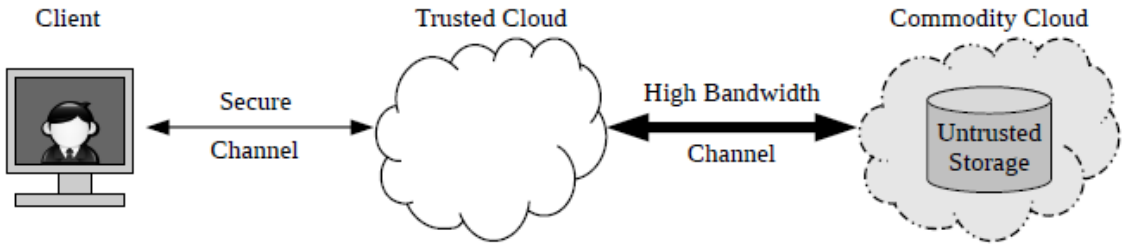
4. *Malleable claw free collection*

Per evitare la generazione di input diversi per diversi circuiti di valutazione (*input inconsistency*) viene utilizzata la tecnica *Malleable claw free collection*: e' una composizione di 4 algoritmi che permette di verificare che ogni input sia stato generato per una singola funzione (quindi per un solo circuito).

• Garbled Circuits e Cloud

- *Semi-honest model*

La motivazione principale del *cloud* consiste nello spostare il carico computazione da un client ad una entità esterna chiamata appunto *cloud.* Considerando che le risorse del *cloud* vengono condivise da più utenti, i problemi di sicurezza diventano molto rilevanti. Un possibile metodo per garantire la sicurezza nel sistemi con *cloud* è utilizzare appunto i *Garbled Circuits*. Abbiamo il seguente scenario:



- *Client*: Si avvale dei servizi offerti dal *cloud*
- *Trusted cloud*: *Cloud* sicuro, tipicamente privato. Offre alto livello di sicurezza.
- *Secure channel*: Canale sicuro tra Client e *cloud* sicuro. Tipicamente a banda stretta.
- *Comodity cloud*: Cloud non sicuro, offre altre prestazioni computazionali e storage.
- *Channel*: Canale tra *cloud* sicuro e *comodity cloud*. Non sicuro e a banda larga.

Solitamente il *Trusted cloud* offre alti requisiti di sicurezza al client, ma è molto costoso e non offre ne elevate prestazioni ne elevata capacità di storage. In questo modello funge da *proxy* tra il client e il *comodity cloud*. Il *comodity cloud* è esattamente l'opposto. Fornisce sia alte prestazioni che elevata capacità di storage dei dati, ma non è considerato sicuro.

Lo scenario è molto semplice: Il client invia dei dati D e un programma P da calcolare. Successivamente effettua una *query* e si aspetta un risultato r in tempi relativamente brevi. Come rendere il procedimento sicuro? E' qui che entrano in gioco i Garbled Circuit.

- Il client fornisce il programma P ed i dati D al *Trusted cloud* in chiaro. Per il canale si utilizza un protocollo di sicurezza (es. SSL/TSL)
- Il *Trusted cloud* cripta D e P tramite *Garbled Circuits* e li salva nel *Comodity cloud.*
- Il client invia una *query* q in chiaro al server sicuro, che la cripta tramite *Garbled Circuit* e la invia al *comodity cloud*.
- Il *comodity cloud* esegue la computazione e invia il risultato criptato al *Trusted cloud*.
- Il *Trusted cloud* verifica la correttezza, decripta i dati e invia il risultato al client.

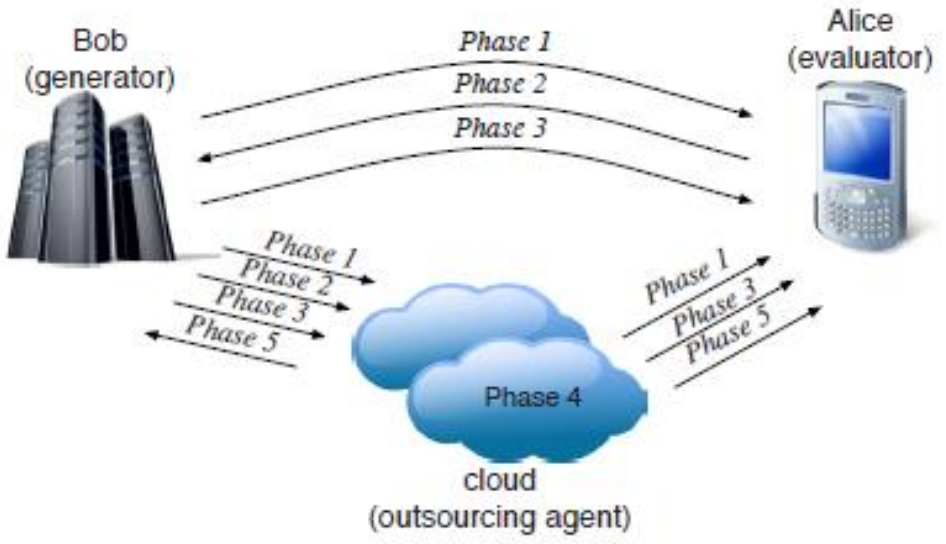
In pratica il *Trusted cloud* e il *comodity cloud* hanno rispettivamente i ruoli di P1 e P2 descritte in precedenza nel protocollo.

- *Malicious model*

Il protocollo precedentemente descritto è sicuro nel caso di un modello semi-onesto. In caso di modello malevolo, sono necessarie delle operazioni aggiuntive per garantire la sicurezza. Uno scenario tipico di questo modello è rappresentato da 3 agenti:

- A: tipicamente un dispositivo mobile
- B: Tipicamente un web server o un *application server* adibito alla generazione dei GC.
- Proxy: Il cloud .

Il protocollo in questo caso può essere diviso in 5 fasi principali



- *Fase 1*: B genera un numero di *Garbled Circuit* e li cripta tramite Hash. A e B scelgono casualmente quali circuiti andranno valutati e quali controllati. Per i circuiti controllati, B invia il principio di casualità con cui li ha generati al *cloud* e l'hash di ciascun circuito ad A. Questo controllo è necessario per verificare che B non abbia generato circuiti corrotti o che deviano dal protocollo concordato (*cut-and-choose*).
- *Fase 2*: A invia i suoi input a B tramite OT. B invia i corrispondenti input *garbled* al *cloud*. In questo modo ne B (che li riceve tramite OT) nè il c*loud* (che li riceve *garbled*) conoscono nulla circa i reali input di A.
- *Fase 3*: B invia i suoi *input garbled* al Cloud che verifica siano consistenti per ciascun circuito di valutazione. Questo previene che B generi input corrotti.
- *Fase 4*: Il *cloud* valuta il circuito di alice e gli ingressi di B.
- *Fase 5:* Il *cloud* invia i valori di output ad A e B .

Come si può notare, in questo caso vengono prese contromisure su tutti i partecipanti alla comunicazione: A e B comunicano tramite protocollo OT, Sul generatore di circuiti B vengono fatti controlli sui circuiti e sull'input che genera, il *cloud* non riceve mai nessun valore in chiaro.

Questo tipo di organizzazione offre le seguenti contromisure:

- *Cut-and-choose*: Contro la generazione di un circuito malevolo (malicius circuit) viene utilizzata la tecnica *cut-and-choose* nella fase 1.
- *Input Encoding*: Contro la generazione di input corrotti (selective failure), viene utilizzata la tecnica *dell'input encoding*: Nella fase 2, A codifica i suoi input in modo tale che una possibile corruzione del circuito non riveli nulla riguardo il suo reale input.
- *Malleable claw free collection*: Utilizzata durante la tecnica *cut-and-choose* per evitare attacchi di tipo input inconsistency.

• Bibliografia

1. **Efficient Secure Computation with Garbled Circuits** *(Yan Huang1, Chih-hao Shen, David Evans, Jonathan Katz, and abhi shelat)*

2. **Improved Garbled Circuit: Free XOR Gates and Applications** *(Vladimir Kolesnikov and Thomas Schneider)*

3. **Secure Outsourced Garbled Circuit Evaluation for Mobile Devices** *(Henry Carter, Georgia Institute of Technology; Benjamin Mood, University of Oregon; Patrick Traynor, Georgia Institute of Technology; Kevin Butler, University of Oregon)*

4. **Twin Clouds: An Architecture for Secure Cloud Computing** *(Sven Bugiel, Stefan Nurnberger, Ahmad-Reza Sadeghi, Thomas Schneider)*

5. **Yao's Garbled Circuits: Recent Directions and Implementations** *(Peter Snyder,University of Illinois at Chicago)*

6. **Yao's Protocol** *(Vitaly Shmatikov)*