CompGC: Efficient Offline/Online Semi-honest Two-party Computation

Alex Ledger
Arkady Yerukhimovich, Adam Groce, Alex Malozemoff

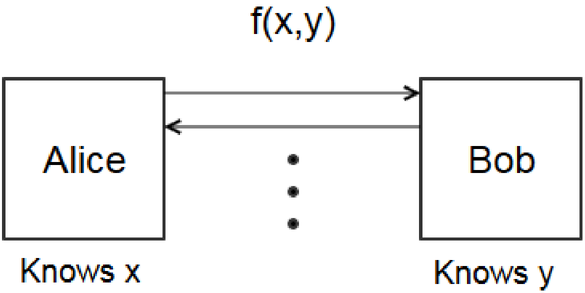
Overview

1. Two Party Secure Computation (2PC)
2. Garbled Circuits
3. Component-based Garbled Circuits
4. Experiments and Results

The Millionaire Problem

- Alice and Bob want to determine who is wealthier [6].
- They do not want to disclose their wealth to each other.
  - Alice has \$$x$, Bob has \$$y$.
  - Alice should not learn anything about y.
  - Bob should not learn anything about x.

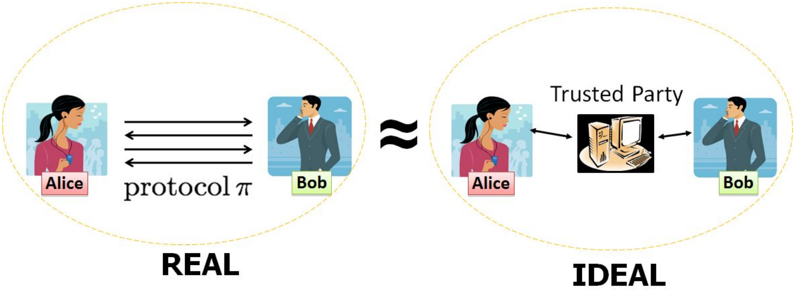
$$f(x,y) = \begin{cases} Alice, & y \leq x; \\ Bob, & y > x. \end{cases} \tag{1}$$



Security Properties

- Confidentiality of Inputs
  - Alice and Bob do not learn anything about the other's input.
  - Except for info that is inferable from their input and output.
  - Bob should not learn that $1,000,000 < x \leq 2,000,000$.
  - But if $y < x$ and $y = 2,000$, then he learns $x < 2,000$.
- Correctness
  - Alice and Bob receive $f(x, y)$.
- Semi-honest
  - We assume that each party obeys the protocol, but attemps to learn extra information from its interactions.

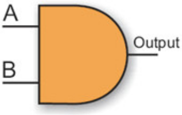
Security



A secure computation protocol is secure if Alice and Bob learn the same information in the real world as they would in ideal world.

Boolean Circuits

- We encode a function f into a circuit C.
- Circuit C is made of AND, XOR and NOT gates.
- AND and XOR gates have two input wires and a single output wire
- A NOT gate has one input wire and one output wire

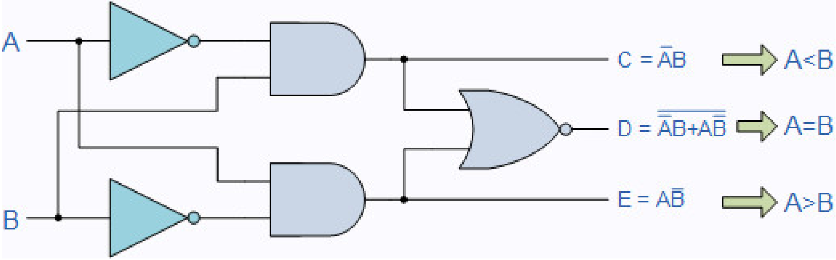


A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

Boolean Circuits

- Any function can be encoded into a circuit.
- Here is the less than circuit.

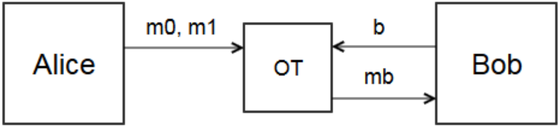


Dual-key Encryption

- Our protocol will use symmetric dual-key encryption.
  - Let $ct = \mathsf{Enc}_{k_0,k_1}(pt)$.
  - And $pt = \mathsf{Dec}_{k_0,k_1}(ct)$.
  - Can be instantiated with $\mathsf{Enc}_{k_0,k_1}(pt) = \mathsf{Enc}_{k_0}(\mathsf{Enc}_{k_1}(pt))$.
- We also assume that you can tell if a decryption *succeeds* or *fails*.

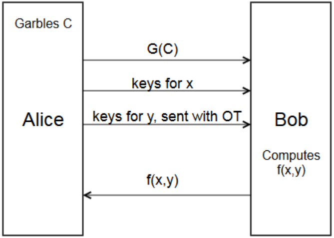
Oblivious Transfer (OT)

- Alice potentially sends either m_0 or m_1 to Bob.
- Bob, without seeing m_0 or m_1, decides that he wants m_b.
- Bob receives m_b.
- Property 1: Alice does not know which message Bob recieved.
- Property 2: Bob doesn't anything about m_{1-b}.



Roadmap of Garbled Circuits

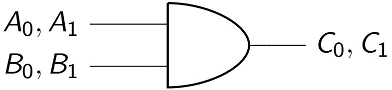
1. Alice (generator) *garbles* the circuit.
2. Alice sends the *garbled tables* of each gate and some keys to Bob.
3. Bob (evaluator) *evaluates* the gate.



Garbling a Gate 1

Step 1. Alice assigns *wire labels* to each wire.

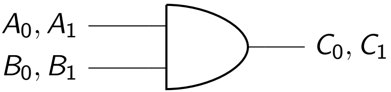
- For each wire in the circuit, assign two random labels to each wire
- Wire A has two wire labels A_0 and A_1.
- We say A_0 *semantically represents* 0, and A_1 *semantically represents* 1.
- And A_0 and A_1 are sampled uniformly at random from $\{0,1\}^k$.



Garbling a Gate 2

Step 2. Alice constructs garbled table.

- Encrypt wire labels of C, C_0 and C_1, using the wire labels of A and B.
- Randomly permute table



A	B	Encryption
A_0	B_0	$\mathrm{Enc}_{A_0,B_0}(C_0)$
A_1	B_0	$\mathrm{Enc}_{A_1,B_0}(C_0)$
A_0	B_1	$\mathrm{Enc}_{A_0,B_1}(C_0)$
A_1	B_1	$\mathrm{Enc}_{A_1,B_1}(C_1)$

Garbling a Gate 3

Step 3. Send garbled table to Bob.

$$\boxed{\begin{array}{c} \mathsf{Enc}_{A_0,B_0}(C_0) \\ \mathsf{Enc}_{A_1,B_0}(C_0) \\ \mathsf{Enc}_{A_0,B_1}(C_0) \\ \mathsf{Enc}_{A_1,B_1}(C_1) \end{array}}$$

Garbling a Gate 4

Step 4. Alice sends input wire labels to Bob.

- Suppose $x \in \{0, 1\}$ is Alice's input, and $y \in \{0, 1\}$ is Bob's input.
- Alice sends A_x to Bob.
- Alice sends B_y to Bob via Oblivious Transfer
  - The wire labels corresponding to her inputs.
- Bob has:

Garbled Table
$\text{Enc}_{A_0,B_0}(C_0)$
$\text{Enc}_{A_1,B_0}(C_0)$
$\text{Enc}_{A_0,B_1}(C_0)$
$\text{Enc}_{A_1,B_1}(C_1)$

Input Labels
A_x
B_y

Garbling a Gate 5

Step 5. Bob evaluates the circuit

- Bob has the garbled table, A_x and B_y.
- Bob trial decrypts each row of the garbled table, until an encrytion succeeds.
- Bob acquires $C_{x \wedge y}$.
- Bob has:

Garbled Table
$\mathrm{Enc}_{A_0,B_0}(C_0)$
$\mathrm{Enc}_{A_1,B_0}(C_0)$
$\mathrm{Enc}_{A_0,B_1}(C_0)$
$\mathrm{Enc}_{A_1,B_1}(C_1)$

Input Labels
A_x
B_y

Output Label
$C_{x \wedge y}$

Garbling a Gate 6

Step 6. Bob gets a final answer.

- Alice sends $\mathsf{Enc}_{C_0}(0)$ and $\mathsf{Enc}_{C_1}(1)$ to Bob.
- Bob trial decrypts these with $C_{x \wedge y}$.
- One will succeed, and Bob will acquire $z = x \wedge y$.
- So Bob knows z, but not x!

Garbled Table
$\mathsf{Enc}_{A_0,B_0}(C_0)$
$\mathsf{Enc}_{A_1,B_0}(C_0)$
$\mathsf{Enc}_{A_0,B_1}(C_0)$
$\mathsf{Enc}_{A_1,B_1}(C_1)$

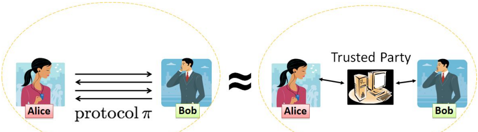
Input Labels
A_x
B_y

Output Label
$C_{x \wedge y}$

Output Map
$\mathsf{Enc}_{C_0}(0)$
$\mathsf{Enc}_{C_1}(1)$

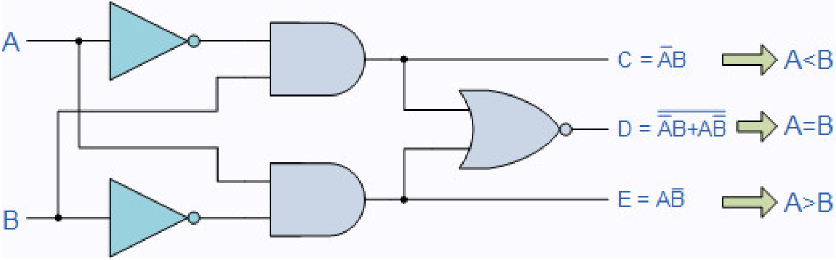
Security Considerations

- Think about what Alice acquires:
  - Alice generates objects and sends them to Bob
  - So she doesn't have many opportunities to learn about Bob's input.
  - The only place she can learn anything is during OT.
- Think about what Bob acquires:
  1. Garbled table
  2. Input wire labels: A_a and B_b
  3. Encryptions of output: $\text{Enc}_{C_0}(0)$ and $\text{Enc}_{C_1}(1)$
- Can Bob learn anything about x?
  - If he could decrypt another row of the garbled table, then we would learn something.
  - But he can't, because he doesn't have the keys.
  - So he doesn't learn anything because everything is encrypted and he doesn't have keys to decrypt anything else.

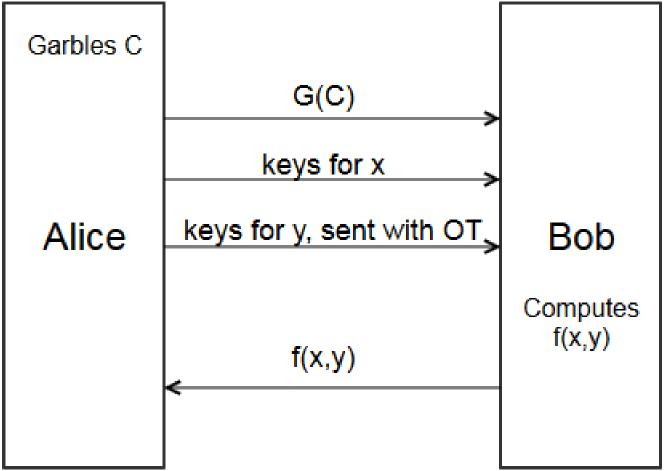


Extending a garbled gate into a garbled circuit

- To operate on a more complex function, the operation is recursed.
- The output wires of the first gates are used as inputs to subsequent gates.
- Alice only sends output maps for the final gates.

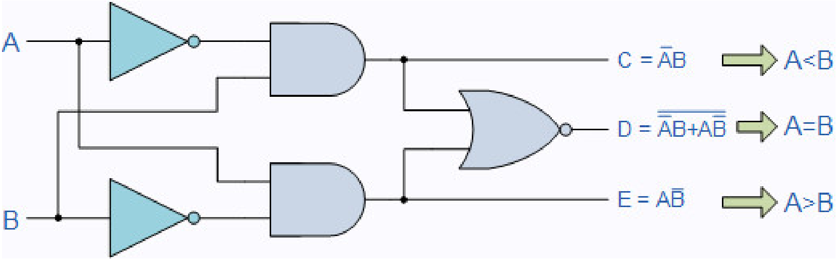


The Garbled Circuit Protocol



The cost of garbled circuits

- Alice sends 4 ciphertexts per gate, since the garbled table has 4 rows, to Bob.
- Based on empirical work, bandwidth is the biggest bottleneck in garbled circuits.
- So reducing the size of the garbled table is of utmost priority.



Reducing the size of the garbled table with Free XOR

- Let $\Delta \leftarrow \{0,1\}^n$ be fixed globally in a circuit [5].
- For each input wire A, sample a single ciphertext; call it A.
  - The *zeroith* wire label is A.
  - The *first* wire label is $A \oplus \Delta$.
- Set output wire C of a gate to be $A \oplus B$ (the xor of its input wires).
- Bob *evaluates* an XOR gate by XORing the two input labels.

$$(A \oplus a\Delta) \oplus (B \oplus b\Delta) = A \oplus B \oplus (a \oplus b)\Delta$$

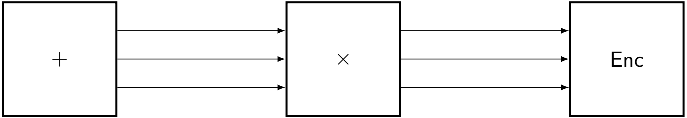
- So XOR gates do not require a garbled table, aka they're free.

Offline/Online

- Imagine two banks use secure computation during their daily operations.
- At night - the offline phase - they exchange garbled circuits (the garbled tables)
- During the day - the online phase - they exchange input wire labels and evaluate the pre-exchanged garbled circuits.
- The computation is fast!
- Problems:
    - Functions are decided at night - no room for flexibility
    - Input size is fixed at night

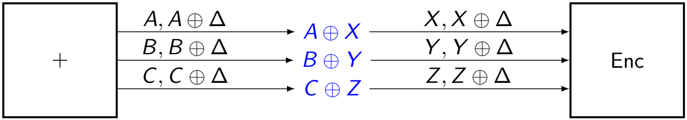
Component-based Garbled Circuits

- Goal: Add flexibility to offline/online garbled circuits [3].
- Key observation: many useful functions in the real world are composed of small, standard components.
  - E.g. addition, subtraction, matrix operations
  - Leveshtein distance algorithm - a dynamic algorithm
  - Encryption is a common component
- Idea: chain garbled circuits together
  - Take the output of one garbled circuit and plug it into another garbled circuit

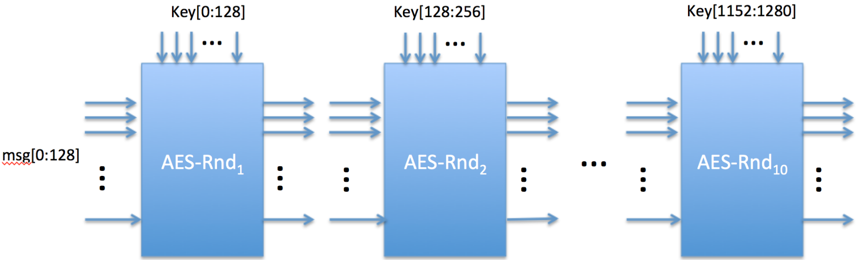


How to chain garbled circuits

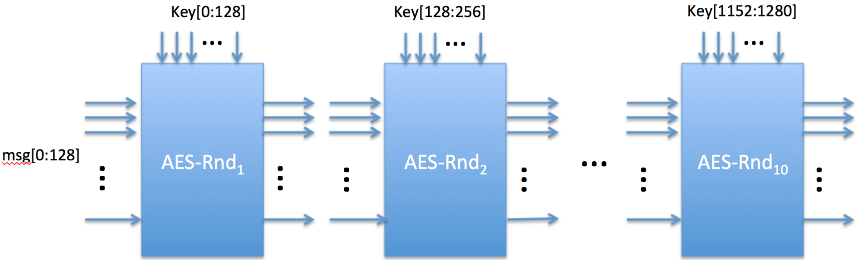
- Suppose that we are chaining a garbled circuit with output wire A to garbled circuit with input wire X.
- We want $A \to X$ and $A \oplus \Delta \to X \oplus \Delta$.
- Straightforward:
  - Alice sends Bob $L_{AX} = A \oplus X$
  - Bob sets $X_* \leftarrow A_* \oplus L_{AX}$



AES with 10 Components



AES with 10 Components



Are component-based garbled circuits secure?

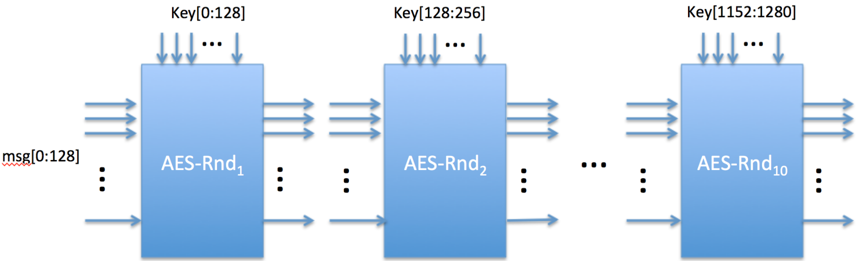
Setup for Our Experiments

- Offline phase:
  1. Generator generates, garbles and sends component-circuits to evaluator.
  2. Generator and evaluator complete OT preprocessing
- Online phase:
  1. Generator sends wire labels for their inputs
  2. Generator and garbler complete OT
  3. Generator sends link labels
  4. Evaluator links and evaluates components

General Experiments

- AES: 10 AES Rounds
- CBC: 100 AES Rounds and 10 XOR for 10-block CBC
- Levenshtein 30: 900 Levenshtein Components
- Levenshtein 60: 3600 Levenshtein Components

AES with 10 Components



Leveshtein Component

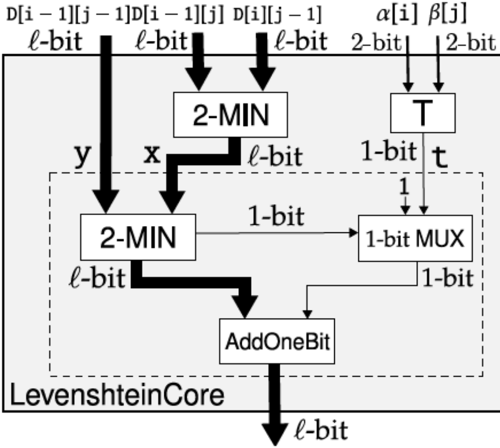


Figure from [4]

General Results

	Time (simulated)		Comm.	
	Naive	CompGC	Naive	CompGC
AES	62.7 ± 0.1	40.8 ± 0.1	1.9	0.4
CBC mode	414.7 ± 0.6	127.2 ± 0.6	18.9	4.3
Leven. (30)	827.7 ± 0.8	114.8 ± 0.7	39.1	3.6
Leven. (60)	3903.0 ± 2.2	408.4 ± 4.1	191.2	15.7

- Times in milliseconds. Communication in Megabits.
- Naive is normal garbled circuits with optimizations and preprocessed OT.
- Times are over network with 50Mbit/s bandwidth and 20ms latency
- All timings are online time of evaluator averaged over 100 trials.
- Experiments done on a laptop.

Machine Learning Experiments

- Private classification: Suppose that a machine learning model exists, and a party wants to privately classify their data with the model.
- We implement functions to query ML models using basic components [2]:
  - Less than
  - Inner product
  - Argmax
  - Addition
  - Select
- ML classifications:
  - Decision Tree
  - Naive Bayes
  - Linear (Hyperplane) Classification
- We use real data UCI Machine Learning repository [1].

Decision Tree Node

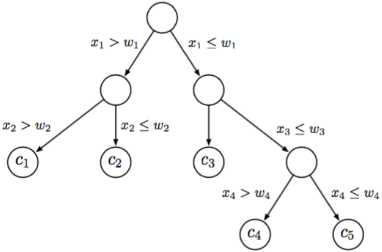


Figure 2: Decision tree

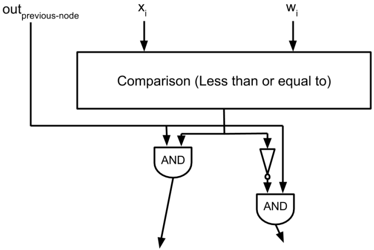
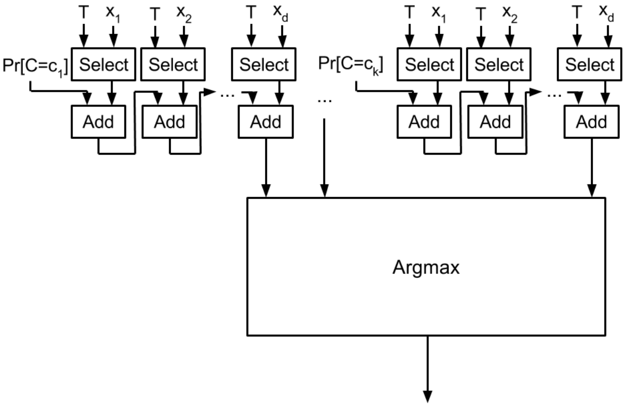


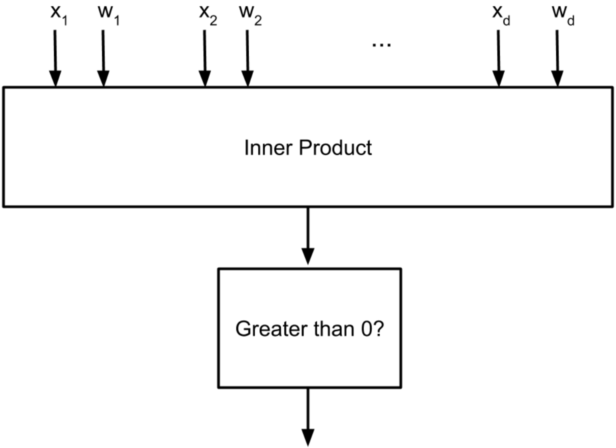
Figure on left from [2]

Naive Bayes

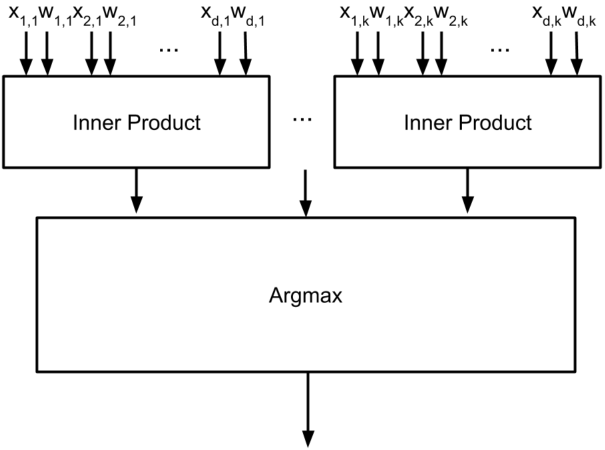


- Select$(arr, idx) \rightarrow arr[idx]$.
- Add$(x, y) \rightarrow x + y$.
- Argmax$(a, b, c, \ldots) \rightarrow$ index of largest of input.

Linear Classifier



Bigger Linear Classifier



Results

- Times in milliseconds. Communication in Megabits.
- Naive is normal garbled circuits with optimizations and preprocessed OT.
- Times in parentheses are over network with bounded bandwidth: 50Mbit/s bandwidth and 20ms latency
- All timings are online time of evaluator averaged over 100 trials.
- Rnds is the number of roundtrips between parties.

Results

		Naive		CompGC		Bost et al. [?]		
Data Set	Size	Time	Comm.	Time	Comm.	Time	Comm.	Rnds
Cancer	30	172 (987)	47	56 (56)	0.7	204	0.3	3.5
Credit	47	256 (1,679)	82	65 (72)	1.1	217	0.3	3.5

(a) Hyperplane decision classifier. "Size" is the length of the model vector w.

	Specs.		Naive		CompGC		Bost et al. [?]		
Data Set	C	F	Time	Comm.	Time	Comm.	Time	Comm.	Rnds
Cancer	2	9	215 (961)	46	97 (485)	22	479	0.6	7
Nursery	5	9	391 (2,832)	138	169 (1,444)	68	1415	1.2	21

(b) Naive Bayes classifier. "C" is the number of classes and "F" is the number of features.

	Specs.		Naive		CompGC		Bost et al. [?]		
Data Set	N	D	Time	Comm.	Time	Comm.	Time	Comm.	Rnds
Nursery	4	4	40 (40)	0.2	40 (40)	0.0	2085	21.6	15
ECG	6	4	40 (40)	0.4	40 (41)	0.1	8816	29.1	22

(c) Decision tree classifier. "N" is the number of internal nodes in the tree and "D" is its depth.

Conclusion

- Communicating the garbled circuit is the bottleneck the garbled circuit protocol.
- Many functions can be constructed from a small set of components.
- Component-based garbled circuits substantially reduce bandwidth and improve online running time over standard garbled circuits.

Bibliography

- K. Bache and M. Lichman. "UCI machine learning repository". Available at `http://archive.ics.uci.edu/ml/`. 2013.
- Raphael Bost et al. "Machine Learning Classification over Encrypted Data." In: *NDSS*. 2015.
- Adam Groce et al. "CompGC: Efficient Offline/Online Semi-honest Two-party Computation." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 458.
- Yan Huang et al. "Faster Secure Two-Party Computation Using Garbled Circuits." In: *USENIX Security Symposium*. Vol. 201. 1. 2011.
- Vladimir Kolesnikov and Thomas Schneider. "Improved garbled circuit: Free XOR gates and applications". In: *Automata, Languages and Programming*. Springer, 2008, pp. 486–498.
- Andrew Yao. "How to generate and exchange secrets". In: *Foundations of Computer Science 1986*. IEEE. 1986, pp. 162–167.