Privacy-Preserving Machine Learning in the Cloud

An Evaluation of Garbled Circuits for Secure Multi-Party Computation

18.10.2024

Fakultät für Ingenieurwissenschaften
Kristin Dahnken
www.hs-wismar.de

HOCHSCHULE WISMAR
UNIVERSITY OF APPLIED SCIENCES TECHNOLOGY, BUSINESS AND DESIGN

Inhalt

1 Motivation

2 Garbled Circuits

3 The Linear Regression Model

4 Privacy-Preserving Machine Learning with Garbled Circuits

5 Proof of Concept

6 Conclusion

7 Outlook on Garbled Circuits in Machine Learning

8 Discussion

Motivation

Motivation

- Machine Learning has become an important tool in research across various domains, from medicine to cybersecurity

Motivation

- Machine Learning has become an important tool in research across various domains, from medicine to cybersecurity
- However, the use of cloud services for machine learning poses challenges when dealing with sensitive data, as control over data privacy is in the hands of the cloud provider

Motivation

- Machine Learning has become an important tool in research across various domains, from medicine to cybersecurity
- However, the use of cloud services for machine learning poses challenges when dealing with sensitive data, as control over data privacy is in the hands of the cloud provider
- Additionally, not all parties involved in developing or training a model should have access to the full training data

Motivation

- To address these problems, this thesis presents a proof of concept that incorporates garbled circuits into the machine learning process

Motivation

- To address these problems, this thesis presents a proof of concept that incorporates garbled circuits into the machine learning process
- The aim is to demonstrate that the usage of *garbled circuits* is both feasible and secure by detailing an exemplary implementation of a simple linear regression model.

Garbled Circuits

Garbled Circuits

- Introduced by Andrew Yao in 1982, garbled circuits are a foundational SMPC protocol, enabling two or more parties to jointly compute a function without revealing individual inputs

Garbled Circuits

- Introduced by Andrew Yao in 1982, garbled circuits are a foundational SMPC protocol, enabling two or more parties to jointly compute a function without revealing individual inputs
- Transforms a Boolean circuit into a *garbled* version, where each gate's true inputs are replaced by randomly generated labels

Garbled Circuits

- Introduced by Andrew Yao in 1982, garbled circuits are a foundational SMPC protocol, enabling two or more parties to jointly compute a function without revealing individual inputs
- Transforms a Boolean circuit into a *garbled* version, where each gate's true inputs are replaced by randomly generated labels

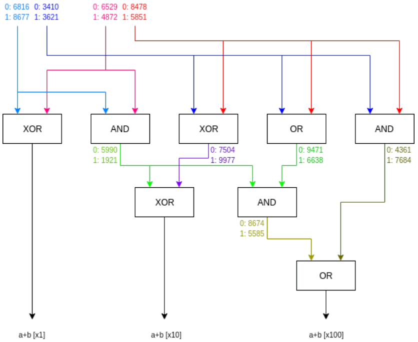


Figure 1: A two-bit-adder represented as a *garbled circuit*

Garbled Circuits

Garbling

- The garbler encrypts the input bits of each gate by creating random labels in their place. To obfuscate the truth table, it is then permuted

Garbled Circuits

Evaluation

- The evaluator receives both the garbled table and input labels. Using *1-2 oblivious transfer*, they acquire the garbled inputs and decrypt the output

Garbled Circuits

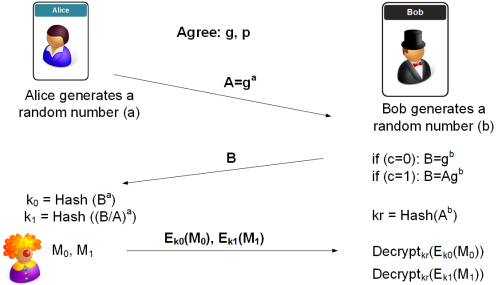
Evaluation

- The evaluator receives both the garbled table and input labels. Using *1-2 oblivious transfer*, they acquire the garbled inputs and decrypt the output
- OT ensures the secure acquisition of the input labels, enabling the garbler to remain unaware of the evaluator's actual inputs

Garbled Circuits

Evaluation

- The evaluator receives both the garbled table and input labels. Using *1-2 oblivious transfer*, they acquire the garbled inputs and decrypt the output
- OT ensures the secure acquisition of the input labels, enabling the garbler to remain unaware of the evaluator’s actual inputs



Garbled Circuits

- Garbled circuits provide strong privacy guarantees by ensuring that only the function output is revealed while the input values remain confidential

Garbled Circuits

- Garbled circuits provide strong privacy guarantees by ensuring that only the function output is revealed while the input values remain confidential
- The protocol is particularly effective in semi-honest settings where parties follow the rules but may try to learn extra information; however, vulnerabilities can arise in malicious environments where adversaries may actively deviate from the protocol

Garbled Circuits

- Garbled circuits provide strong privacy guarantees by ensuring that only the function output is revealed while the input values remain confidential
- The protocol is particularly effective in semi-honest settings where parties follow the rules but may try to learn extra information; however, vulnerabilities can arise in malicious environments where adversaries may actively deviate from the protocol
- Despite their security strengths, garbled circuits face challenges with computational and communication overhead, especially when dealing with large or complex circuits

Garbled Circuits

- Bob picks a random n-bit integer, and computes privately the value of $E_a(x)$; call the result k

Garbled Circuits

- Bob picks a random n-bit integer, and computes privately the value of $E_a(x)$; call the result k
- Bob sends Alice the number $k - j + 1$

Garbled Circuits

- Bob picks a random n-bit integer, and computes privately the value of $E_a(x)$; call the result k
- Bob sends Alice the number $k - j + 1$
- Alice computes privately the values of $y_u = D_a(k - j + u)$ for $u = 1, 2, \ldots, 10$

Garbled Circuits

- Bob picks a random n-bit integer, and computes privately the value of $E_a(x)$; call the result k
- Bob sends Alice the number $k - j + 1$
- Alice computes privately the values of $y_u = D_a(k - j + u)$ for $u = 1, 2, \ldots, 10$
- Alice generates a random prime p of $N/2$ bits, and computes the values $z_u = y_u \pmod{p}$ for all u; if all z_u differ by at least 2 in the mod$p$ sense, stop; otherwise generates another random prime and repeats the process until all z_u differ by at least 2; let p, z_u denote this final set of numbers

Garbled Circuits

- Alice sends the prime p and the following 10 numbers to $B: z_1, z_2, \dots, z_i$ followed by $z_i + 1, z_{i+1} + 1, \dots, z_{10} + 1$; the above numbers should be interpreted in the $\pmod p$ sense

Garbled Circuits

- Alice sends the prime p and the following 10 numbers to $B: z_1, z_2, \ldots, z_i$ followed by $z_i + 1, z_{i+1} + 1, \ldots, z_{10} + 1$; the above numbers should be interpreted in the $\pmod p$ sense
- Bob looks at the j-th number (not counting p) sent from Alice, and decides that $i \geq j$ if it is equal to $x \bmod p$, and $i < j$ otherwise

Garbled Circuits

- Alice sends the prime p and the following 10 numbers to $B: z_1, z_2, \ldots, z_i$ followed by $z_i + 1, z_{i+1} + 1, \ldots, z_{10} + 1$; the above numbers should be interpreted in the $\pmod{p}$ sense
- Bob looks at the j-th number (not counting p) sent from Alice, and decides that $i \geq j$ if it is equal to $x \bmod p$, and $i < j$ otherwise
- Bob tells Alice what the conclusion is.

The Linear Regression Model

The Linear Regression Model

- Describes the dynamics between a dependent variable y_i and one (or multiple) independent variables x_i

The Linear Regression Model

- Describes the dynamics between a dependent variable y_i and one (or multiple) independent variables x_i
- Used to make predictions on these sets of data through estimating the coefficients of the underlying linear equation

The Linear Regression Model

- Describes the dynamics between a dependent variable y_i and one (or multiple) independent variables x_i
- Used to make predictions on these sets of data through estimating the coefficients of the underlying linear equation
- Simple linear regression calculations use the *mean squared error* function to find the best for a given set of data

The Linear Regression Model

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

- $\hat{y}_i$ is the predicted output of the dependent variable
- β_0 is the intercept or constant
- β_1 is the slope or regression coefficient
- x_i is the independent variable

Privacy-Preserving Machine Learning with Garbled Circuits

Privacy-Preserving Machine Learning with Garbled Circuits

What needs to be done to achieve this?

Privacy-Preserving Machine Learning with Garbled Circuits

What needs to be done to achieve this?

- A breakdown of all necessary computations into smaller binary operations that can be executed without revealing sensitive data

Privacy-Preserving Machine Learning with Garbled Circuits

What needs to be done to achieve this?

- A breakdown of all necessary computations into smaller binary operations that can be executed without revealing sensitive data
- Conversion of training data into binary format, garbling and evaluating the circuits

Privacy-Preserving Machine Learning with Garbled Circuits

What needs to be done to achieve this?

- A breakdown of all necessary computations into smaller binary operations that can be executed without revealing sensitive data
- Conversion of training data into binary format, garbling and evaluating the circuits
- Usage of non-interactive garbled circuits to maintain confidentiality during the training phase

Privacy-Preserving Machine Learning with Garbled Circuits

Binary Operations

Privacy-Preserving Machine Learning with Garbled Circuits

Binary Operations

- All *basic arithmetic operations* are translated into binary circuits constructed from logic gates like *AND*, *OR* and *XOR*

Privacy-Preserving Machine Learning with Garbled Circuits

Binary Operations

- All *basic arithmetic operations* are translated into binary circuits constructed from logic gates like *AND, OR* and *XOR*

- Addition and subtraction utilize *half-adders* and *full-adders* to handle carry bits, enabling the addition and subtraction of numbers of arbitrary bit lengths through ripple-carry adders

Privacy-Preserving Machine Learning with Garbled Circuits

Binary Operations

- All *basic arithmetic operations* are translated into binary circuits constructed from logic gates like *AND, OR* and *XOR*

- Addition and subtraction utilize *half-adders* and *full-adders* to handle carry bits, enabling the addition and subtraction of numbers of arbitrary bit lengths through ripple-carry adders

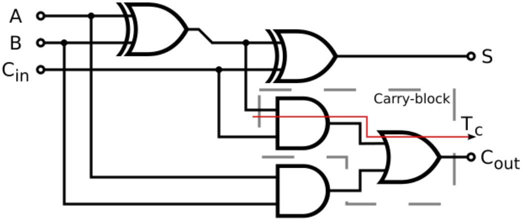


Figure 3: Full-adder schematic

Privacy-Preserving Machine Learning with Garbled Circuits

- Multiplication and division are achieved through techniques like *successive addition* and *repeated subtraction* to simulate these operations in a binary format

Privacy-Preserving Machine Learning with Garbled Circuits

- Multiplication and division are achieved through techniques like *successive addition* and *repeated subtraction* to simulate these operations in a binary format
- Additionally, auxiliary functions like the *Two's Complement* are implemented

Proof of Concept

Proof of Concept

Binary Operations

Proof of Concept

Binary Operations

- In the Data Preparation Layer, all training data is converted from decimal to *signed binary*, with a predetermined bit length

Proof of Concept

Binary Operations

- In the Data Preparation Layer, all training data is converted from decimal to *signed binary*, with a predetermined bit length
- All binary operations are implemented, from the *half-adder* function to the *n-bit Division*

Proof of Concept

Garbled Circuits

Proof of Concept

Garbled Circuits

- `Gabes` library in python serves as blueprint

Proof of Concept

Garbled Circuits

- `Gabes` library in python serves as blueprint

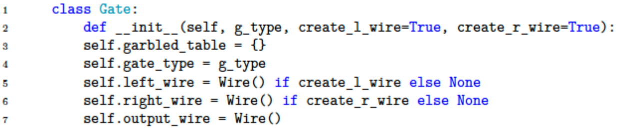
- Includes all components of a garbled circuit, with garbling, evaluation and decryption methods

Proof of Concept

Garbled Circuits

- `Gabes` library in python serves as blueprint

- Includes all components of a garbled circuit, with garbling, evaluation and decryption methods



```python
class Gate:
    def __init__(self, g_type, create_l_wire=True, create_r_wire=True):
        self.garbled_table = {}
        self.gate_type = g_type
        self.left_wire = Wire() if create_l_wire else None
        self.right_wire = Wire() if create_r_wire else None
        self.output_wire = Wire()
```

Figure 4: Gate class

Proof of Concept

Templates

Proof of Concept

Templates

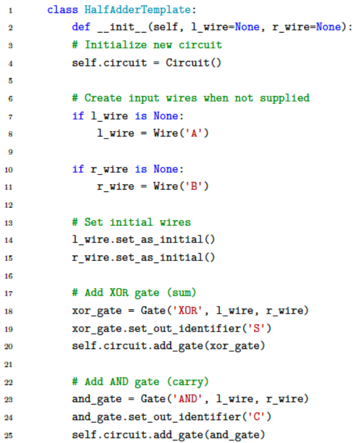
- Implementation features reusable templates with a similar structure for all arithmetic circuit operations

Proof of Concept

Templates

- Implementation features reusable templates with a similar structure for all arithmetic circuit operations
- Each template provides a method for garbling and evaluation, including subcircuit evaluation when necessary

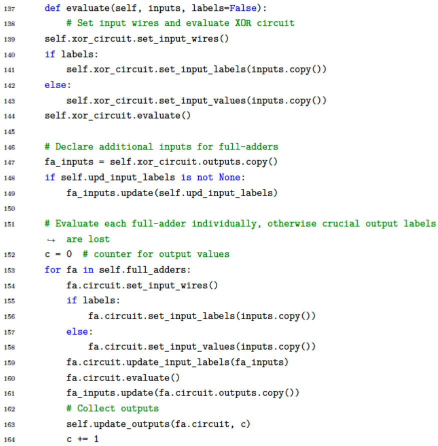
Proof of Concept



```
class HalfAdderTemplate:
    def __init__(self, l_wire=None, r_wire=None):
    # Initialize new circuit
    self.circuit = Circuit()

    # Create input wires when not supplied
    if l_wire is None:
        l_wire = Wire('A')

    if r_wire is None:
        r_wire = Wire('B')

    # Set initial wires
    l_wire.set_as_initial()
    r_wire.set_as_initial()

    # Add XOR gate (sum)
    xor_gate = Gate('XOR', l_wire, r_wire)
    xor_gate.set_out_identifier('S')
    self.circuit.add_gate(xor_gate)

    # Add AND gate (carry)
    and_gate = Gate('AND', l_wire, r_wire)
    and_gate.set_out_identifier('C')
    self.circuit.add_gate(and_gate)
```

Figure 5: The general template structure, applied to the half-adder

Proof of Concept



```
def evaluate(self, inputs, labels=False):
    # Set input wires and evaluate XOR circuit
self.xor_circuit.set_input_wires()
if labels:
    self.xor_circuit.set_input_labels(inputs.copy())
else:
    self.xor_circuit.set_input_values(inputs.copy())
self.xor_circuit.evaluate()

# Declare additional inputs for full-adders
fa_inputs = self.xor_circuit.outputs.copy()
if self.upd_input_labels is not None:
    fa_inputs.update(self.upd_input_labels)

# Evaluate each full-adder individually, otherwise crucial output labels
↪   are lost
c = 0  # counter for output values
for fa in self.full_adders:
    fa.circuit.set_input_wires()
    if labels:
        fa.circuit.set_input_labels(inputs.copy())
    else:
        fa.circuit.set_input_values(inputs.copy())
    fa.circuit.update_input_labels(fa_inputs)
    fa.circuit.evaluate()
    fa_inputs.update(fa.circuit.outputs.copy())
    # Collect outputs
    self.update_outputs(fa.circuit, c)
    c += 1
```

Figure 6: Evaluating an adder-subtractor circuit

Proof of Concept

Training

Proof of Concept

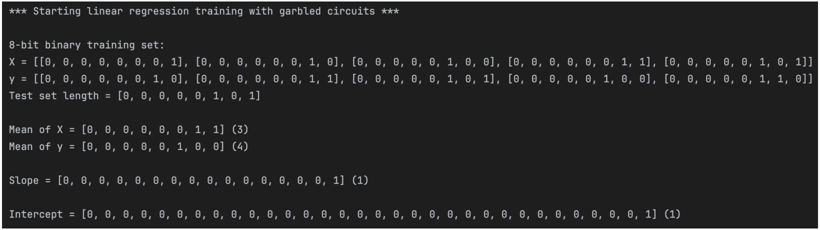
Training

- All training steps are executed with both binary and garbled circuits

Proof of Concept

Training

- All training steps are executed with both binary and garbled circuits
- Utilizes a simple training set that can easily be verified



```
*** Starting linear regression training with garbled circuits ***

8-bit binary training set:
X = [[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
y = [[0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0]]
Test set length = [0, 0, 0, 0, 0, 1, 0, 1]

Mean of X = [0, 0, 0, 0, 0, 0, 1, 1] (3)
Mean of y = [0, 0, 0, 0, 0, 1, 0, 0] (4)

Slope = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)

Intercept = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)
```

Figure 7: Result of the garbled circuit intercept calculation

Conclusion

Conclusion

- The research demonstrates that garbled circuits can effectively protect sensitive data during machine learning processes

Conclusion

- The research demonstrates that garbled circuits can effectively protect sensitive data during machine learning processes
- Unfortunately, the computational demand is still high, although overall execution takes less than a second

Conclusion

- The research demonstrates that garbled circuits can effectively protect sensitive data during machine learning processes
- Unfortunately, the computational demand is still high, although overall execution takes less than a second

Run	Binary Circuits	Garbled Circuits
1	0.0016210079193115234	0.3389451503753662
2	0.0010771751403808594	0.3446238040924072
3	0.0011820793151855469	0.33850693702697754
4	0.0005500316619873047	0.379697322845459
5	0.0006239414215087891	0.3488502502441406

Tabelle 1: Execution duration of Linear Regression training in seconds, using binary and garbled circuits

Conclusion

- Increasing execution times are also an indicator of higher memory consumption

Conclusion

- Increasing execution times are also an indicator of higher memory consumption
- Despite linear regression being one of the simplest machine learning models, combining it with garbled circuits took a comparatively high implementation effort

Conclusion

- Protocol remains inherently secure, so the proof of concept can be considered secure as well

Conclusion

- Protocol remains inherently secure, so the proof of concept can be considered secure as well
- Potential vulnerabilities stem from the implementation itself

Conclusion

- Protocol remains inherently secure, so the proof of concept can be considered secure as well
- Potential vulnerabilities stem from the implementation itself
- Disclosure of intermediate values necessary but not enough for an adversary to derive additional information

Conclusion

- Templates offer adaptablility for varying training sets

Conclusion

- Templates offer adaptablility for varying training sets
- Although circuits as whole cannot be reused, the template offers reusable components like gates or wires

Conclusion

- Templates offer adaptablility for varying training sets
- Although circuits as whole cannot be reused, the template offers reusable components like gates or wires
- No floating point arithmetic

Conclusion

- Templates offer adaptablility for varying training sets
- Although circuits as whole cannot be reused, the template offers reusable components like gates or wires
- No floating point arithmetic
- Only simplest form of garbling available

Outlook on Garbled Circuits in Machine Learning

Outlook on Garbled Circuits in Machine Learning

- Future work should incorporate optimizations such as the *point-permute technique* or *free XOR* to reduce the amount of ciphertexts per gate and improve overall computational efficiency

Outlook on Garbled Circuits in Machine Learning

- Future work should incorporate optimizations such as the *point-permute technique* or *free XOR* to reduce the amount of ciphertexts per gate and improve overall computational efficiency
- Enabling support for *floating point arithmetic* in training models to allow for more complex datasets and algorithms

Outlook on Garbled Circuits in Machine Learning

- Future work should incorporate optimizations such as the *point-permute technique* or *free XOR* to reduce the amount of ciphertexts per gate and improve overall computational efficiency
- Enabling support for *floating point arithmetic* in training models to allow for more complex datasets and algorithms
- Allowing multiple parties to contribute their training data would facilitate joint model training, further validating the practical utility of garbled circuits in real-world scenarios

Outlook on Garbled Circuits in Machine Learning

- Garbled circuits can not only be utilized for training but also for secure model inference, allowing clients to obtain predictions from a trained model without revealing their inputs or the model's structure to the server

Outlook on Garbled Circuits in Machine Learning

- Garbled circuits can not only be utilized for training but also for secure model inference, allowing clients to obtain predictions from a trained model without revealing their inputs or the model's structure to the server
- While the proof of concept focused on linear regression, it can be extended to other algorithms, such as *logistic regression* and *neural networks*

Discussion