Secure Decision Forest Evaluation

Slim Bettaieb, Loic Bidoux, Olivier Blazy, Baptiste Cottier, David Pointcheval

Secure Decision Forest Evaluation

Slim Bettaieb[1], Loic Bidoux[1,2], Olivier Blazy[3], Baptiste Cottier[1,4,5], and David Pointcheval[4,5]

[1]Worldline, France
[2]Cryptography Research Centre, United Arab Emirates
[3]XLIM, University of Limoges, France
[4]DIENS, Ecole Normale Superieure de Paris, France
[5]INRIA, France

August 19, 2021

Abstract

Decision forests are classical models to efficiently make decision on complex inputs with multiple features. While the global structure of the trees or forests is public, sensitive information have to be protected during the evaluation of some client inputs with respect to some server model. Indeed, the comparison thresholds on the server side may have economical value while the client inputs might be critical personal data. In addition, soundness is also important for the receiver. In our case, we will consider the server to be interested in the outcome of the model evaluation so that the client should not be able to bias it. In this paper, we propose a new offline/online protocol between a client and a server with a constant number of rounds in the online phase, with both privacy and soundness against malicious clients.

1 Introduction

Over the past years, companies have tremendously increased the amount of data they collect from their users. These data are often feed to machine learning algorithms in order to turn them into valuable business insights that are used to develop innovative services. Applications include user authentication, fraud detection in banking systems, recommendation services as well as spam detection. However, collecting and processing these data raises privacy concerns since they generally contain sensitive information regarding users. Besides, the models used to evaluate these data may contain critical business information that also need to be protected. In this work, we focus on the secure evaluation of decision forests which are a commonly used class of machine learning algorithms. We consider the case where a client who holds sensitive data interacts with a

server who holds a decision forest model in order to jointly evaluate the client inputs with respect to the server model. Our goal is to ensure that the privacy of both the client and the server is guaranteed on their respective inputs. We also investigate the scenario where the client is malicious and intends to bias the outcome of the protocol.

In order to motivate the design rationale of the proposed protocols, we consider continuous user authentication based on decision forests as an application. In continuous authentication, users are authenticated using a set of features that strengthen usual authentication credentials such as passwords or security tokens. When the user's identity needs to be validated after some time interval or after some inactivity, continuous authentication offers a user-friendly experience as it avoids interrupting legitimate users and reduces the number of times they have to authenticate explicitly. The usual scenario of continuous authentication consists of a server authenticating users based on behavioural biometrics. A variety of features such as keystroke patterns, swiping gestures and scrolling duration are collected on the user's device and sent to the server. The server then evaluates the client inputs with respect to its model in order to make an authentication decision. The model is usually generated during a training step using a dedicated training dataset.

1.1 Related Work

Several approaches have been proposed in order to securely evaluate decision trees [BPSW07, BFK+09, BPTG15, WFNL16, TMZC17, DDH+16, TKK19]. However, most of these constructions are either only secure in the honest-but-curious setting or tailored for decision tree evaluation rather than decision forest evaluation. As such, with the exception of [WFNL16] and [TMZC17] which we consider hereafter, the aforementioned works can't be compared to our protocol meaningfully. Similarly to our protocol, both [WFNL16] and [TMZC17] rely on Additive Homomorphic Encryption (AHE) and Oblivious Transfers (OT). In addition, our protocol also uses Garbled Circuits (GC) in the malicious setting.

A major difference between our protocol and existing constructions is that we rely on the server sending its encrypted model to the client rather than the client sending its encrypted data to the server. This strongly impacts the design of our protocol and allows us to introduce a preprocessing phase that can be performed offline and leveraged later during the online phrase. This is advantageous for several use-cases such as the continuous authentication one as it offers a trade-off between the number of rounds required to execute the protocol and the required bandwidth.

All existing constructions leak some information with respect to the model structure (*i.e.* on the server side) whether it be its total number of nodes M, the total number of comparison nodes m or the maximal depth δ of the trees. In our protocol, the client may also learn which features will be used to evaluate the trees. We consider this as a privacy leak, but as our protocols will have a complexity independent of the depth of the trees (or more precisely, the length of the paths down to the leaves), we will be able to add dummy comparisons

Table 1: Comparison to state-of-the-art

Scheme	Rounds	Tools	Bandwidth
Honest-but-Curious model			
[WFNL16]	6	AHE+OT	$\mathcal{O}(m)$
[TMZC17]	4	AHE+OT	$\mathcal{O}(m)$
Section 3 (Offline)	0.5	AHE	$\mathcal{O}(m \cdot \delta \cdot 2^{\nu})$
			$\mathcal{O}(m \cdot \chi \cdot 2^{\nu})$
Section 3 (Online)	0.5		$\mathcal{O}(P)$
Malicious model			
[WFNL16]	2	AHE+OT	$\mathcal{O}(M)$
[TMZC17]	4	AHE+OT	$\mathcal{O}(m)$
Section 4 (Offline)	0.5	AHE	$\mathcal{O}(m \cdot \delta \cdot 2^{\nu})$
			$\mathcal{O}(m \cdot \chi \cdot 2^{\nu})$
Section 4 (Online)	2.5	AHE+GC+OT	$\mathcal{O}(P)$

with dummy features, which will completely hide which are the actually used features. We thus propose two variants of our protocols in Table 1, without or with dummy comparisons. They only differ during the offline step as the number of features in the comparisons impacts the communication and the storage: δ is the number of real features per path, whereas χ is the number of real and dummy features. We stress that this modification does not impact the online step of our protocols: communication only depends on the number P of paths, and not their length. This thus allows us to use as many dummy features as we want to hide the trees and the forest structure. This will be a crucial property for the privacy of the model. In addition, our constructions also feature some leakage on the client side. When used to evaluate decision forests, our protocol leaks the number of successful paths within the forest which constitutes a tolerated leakage in our targeted application. Indeed, it corresponds to the number of accepting trees which allows to compute a confidence level associated to the result.

One can note half rounds in our constructions, which mean one-way flows, from the sender to the recipient. Indeed, most of our schemes are actually non-interactive. Since our goal is to provide the result to the server, in the malicious setting, we get a 5-flow protocol.

1.2 Contributions

In this paper, we propose several constructions to securely evaluate decision forests with binary output classes. In our setting, a server evaluates a model $\mathcal{M}$ with respect to some client inputs $\boldsymbol{x}$ and accept or reject the client according to the evaluation outcome. We consider the server to be honest-but-curious and describe two protocols that are respectively tailored for honest-but-curious and malicious clients. As we target applications where the interactions between the client and the server should be as low as possible, we design two-step protocols in which some part of the computation can be performed offline, and the communication performed before knowing the inputs of the client.

In the honest-but-curious setting, our protocol only requires one flow from the client to the server to be executed during the online step which outperforms previous results from the literature. In the malicious setting, our protocol can be seen as a trade-off between existing constructions with respect to the number of rounds required and the bandwidth cost. However, our protocols leak the number of trees successfully evaluated within the forest to the server. We consider this as a feature rather than a drawback as we want the server to learn the outcome of the evaluation in our setting. Indeed, this additional information is generally used as a confidence score with respect to the evaluation outcome and is expected to be known in some use-cases such as the continuous authentication one.

Another contribution is the fact that in our context, we use garbled circuit in a malicious setting, without any additional techniques compared to the honest-but-curious setting. Thereby, we do not need to use solutions such as *Cut & Choose*. Even if this solution has been well studied and optimized [MF06, LP07, Lin13, LR14, AMPR14, WMK17], this is still an expensive solution requiring ℓ garbled circuits for statistical security $2^{-\ell}$. In a survey, Dupin et al. [DPB18] showed that a malicious generator can corrupt a garbled circuit only by adding NOT-gates or by failure attacks, but we can protect against these attacks in our context. Indeed, in case of failure attacks, the adversary will likely be rejected, without learning any information about the thresholds. In the case of wrong circuit (with additional NOT-gates), we have introduced random inversions of the outcomes of the paths, and so an attack will reduce to guess all (or most of) the inversions, which will again likely lead to a reject, without leaking any information.

1.3 Paper Organization

We present the main tools that will be used to design our secure decision forest evaluation protocols in Section 2, Then, we describe and analyze our construction in the honest-but-curious setting in Section 3. Eventually, we move to the malicious setting in Section 4. Performances and results for several applications are discussed in Section 5.

2 Preliminaries

2.1 Decision Tree Learning

Decision tree learning is a discipline used to solve multi-criteria problems that can be modelled using *decision trees*. In this paper, we focus on binary classification trees, which are decision trees whose leaves can take two values (the two output classes). In order to improve the accuracy of the model, one often considers *decision forests* (sets of decision trees) where each tree of the forest is evaluated separately and then aggregated for the final decision.

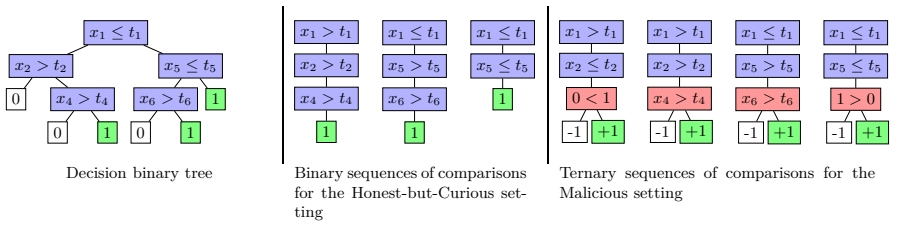

Figure 1: Decision tree, Sequences of comparisons, Binary sequences

A decision forest is thus a list of binary decision trees, as one is shown on Figure 1, on the left part. They can each be converted into a list of comparison sequences down to accepting leaves (in the center of Figure 1). Each comparison is between a feature value x_i and the threshold value t_i from the model: we thus denote the model $\mathcal{M} = (P, \delta, \nu, \tau, (t_{i,j}, v_{i,j})_{i\in[P],j\in[\delta]})$ to represent a binary decision forest and $\boldsymbol{x} = (x_{i,j})_{i\in[P],j\in[\delta]}$ the inputs of size ν to be evaluated. The model $\mathcal{M}$ represents P paths of maximal depth δ that can be evaluated to compute a score in order to determine the output of the evaluation with respect to some threshold τ on the number of accepted paths (which is equal to the number of accepting trees). Each path is a series of comparisons, where the (i, j)-th comparison denotes the comparison of depth j in the i-th path: namely, the comparison of input $x_{i,j}$ and threshold $t_{i,j}$. In addition, the boolean values $v_{i,j}$ are used to determine the comparison operator. *Lower or equal* ($\leq$) whenever $v_{i,j} = 1$ or *Strictly greater* ($>$) whenever $v_{i,j} = 0$. A path is considered to be accepting if all its comparisons yields to `TRUE`, otherwise it is rejecting. And then a tree is accepting if one path is accepting. One can note that in the sequences extracted from a tree, at most one is accepting. Given some input $\boldsymbol{x} = (x_{i,j})_{i\in[P],j\in[\delta]}$, the number of accepting paths with respect to model $\mathcal{M}$ is denoted by $\mathcal{M}(\boldsymbol{x})$. It then corresponds to the number of accepting trees. Hence, the outcome of the decision forest evaluation is the Boolean $(\mathcal{M}(\boldsymbol{x}) \geq \tau)$.

More concretely, as shown on Figure 1, if we have a forest with T trees, we extract all the paths down to accepting leaves, with the successive comparisons $(t_j, v_j)_j$. On a given input $\boldsymbol{x}$, each tree has at most one accepting path, and so at most T accepting paths in total. Then, we can decide to accept $\boldsymbol{x}$ when at least $T/2$ among the P paths (the majority of the trees) are accepting: $\tau = T/2$. We stress that in this scenario, each path will be considered accepting or rejecting. But we will just expect at least $T/2$ accepting paths among P. This will be enough for our protocol in the honest-but-curious setting: we essentially ignore rejecting paths.

For the malicious setting, we will consider more complex sequences of comparisons, with a ternary output: accepting, rejecting, or ignoring. This is the right part of Figure 1: if the last red comparison is reached, a decision is taken, as accept (+1) or reject (-1), whereas when the last comparison is not reached, the path will be ignored (0). This way, most of the paths will be ignored, and exactly one path will be accepting or rejecting, as the global tree would be. We stress that some 'always true' or 'always false' comparisons will have to be added

to make the above technique work properly. This is easy to see that the three ways of representing and evaluating a binary decision tree are equivalent, with a final outcome 'accept' or 'reject'. The last one will allow to prevent malicious behaviours from the client, in order to falsely get accepted.

2.2 Public-Key Encryption

A *public-key encryption* scheme PKE is defined by three algorithms ($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$):

- $\mathsf{KeyGen}(1^\kappa)$: with input κ as security parameter, returns a public encryption key pk and a private decryption key sk.
- $\mathsf{Enc}(\mathsf{pk}, m)$: returns $[\![m]\!]$, an encryption of m under the public encryption key pk
- $\mathsf{Dec}(\mathsf{sk}, [\![m]\!])$: returns m.

Such an encryption scheme should provide secrecy of the message. But as the encryption key is public, anybody can encrypt any message of its choice. We thus talk about *indistinguishability against chosen-plaintext attacks* ($\mathsf{IND\text{-}CPA}$).

2.3 Homomorphic Encryption

An *Additively Homomorphic Encryption scheme* AHE on plaintexts over an additive group of size p (typically, it will be $\mathbb{Z}_p$) is a PKE scheme with two more algorithms (Add, $\mathsf{MultScal}$):

- $\mathsf{Add}(\mathsf{pk}, [\![m_1]\!], [\![m_2]\!])$: Given pk and two ciphertexts $[\![m_1]\!], [\![m_2]\!]$, returns $[\![m_1]\!] \boxplus [\![m_2]\!] = [\![m_1 + m_2]\!]$ an encryption of the sum of the plaintexts under the same public key pk;
- $\mathsf{MultScal}(\mathsf{pk}, [\![m]\!], k)$: Given pk, a ciphertext $[\![m]\!]$, and a scalar $k \in \mathbb{Z}_p$, returns $k \boxdot [\![m]\!] = [\![k \cdot m]\!]$;

Two randomization properties will also be considered:

- $\mathsf{Randomize}(\mathsf{pk}, [\![m]\!])$: Given pk and a ciphertext $[\![m]\!]$, returns a different ciphertext of m. It can be implemented as $\mathsf{Add}(\mathsf{pk}, [\![m]\!], \mathsf{Enc}(\mathsf{pk}, 0))$;
- $\mathsf{MultRand}(\mathsf{pk}, [\![m]\!])$: Given pk, a ciphertext $[\![m]\!]$, returns a ciphertext of $k \cdot m$, for a non-zero random k.

Lifted ElGamal Encryption Scheme. ElGamal [ElG84] encryption is a multiplicative homomorphic encryption scheme. To make it additive, we encode a message as g^m. Also, after decryption, we retrieve g^m. No discrete logarithm computation is required as we only check if m belongs to a given interval $[\tau_{\min}, \tau_{\max}]$, which can be done by checking either $g^m \in \{g^i\}_{i \in [\tau_{\min}, \tau_{\max}]}$ or not.

- $\mathsf{KeyGen}(1^\kappa)$: Generates a cyclic group G of order p with $|p| = \kappa$, with generator g; samples $x \xleftarrow{\$} \mathbb{Z}_p$; returns $\mathsf{pk} = (G, p, g, h = g^x)$ and $\mathsf{sk} = x$;
- $\mathsf{Enc}(\mathsf{pk}, m)$: Generates $y \xleftarrow{\$} \mathbb{Z}_p$; computes $c_1 = g^y$ and $c_2 = g^m \cdot h^y$; returns $[\![m]\!] = (c_1, c_2)$;
- $\mathsf{Dec}(\mathsf{sk}, [\![m]\!] = (c_1, c_2))$: Computes $c_2 \cdot c_1^{-x} = g^m$;
- $\mathsf{Add}(\mathsf{pk}, [\![m_1]\!] = (c_1^1, c_2^1), [\![m_2]\!] = (c_1^2, c_2^2))$: Computes $c_1^3 = c_1^1 \cdot c_1^2$ and $c_2^3 = c_2^1 \cdot c_2^2$; returns $[\![m_1 + m_2]\!] = (c_1^3, c_2^3)$;
- $\mathsf{MultScal}(\mathsf{pk}, [\![m]\!] = (c_1, c_2), k)$: Computes $c_1' = (c_1)^k$ and $c_2' = (c_2)^k$; returns $[\![k \cdot m]\!] = (c_1', c_2')$;
- $\mathsf{Randomize}$ and $\mathsf{MultRand}$ are computed thanks to Add and $\mathsf{MultScal}$ as described above.

2.4 Oblivious Transfer

An *Oblivious Transfer* OT is a two-party protocol between a sender with input a pair of messages (m_0, m_1) and a receiver with input a bit b that allows the receiver to retrieve m_b. The receiver should not learn anything about b, while the receiver should not learn anything about m_{1-b}. A particular family of OT can be defined as a tuple of algorithms $(\mathsf{Encode}, \mathsf{Compute}, \mathsf{Decode})$:

- $\mathsf{Encode}(b)$: Given a bit b, returns the encoded value $\widetilde{b}$;
- $\mathsf{Compute}((m_0, m_1), \widetilde{b})$: Given two messages (m_0, m_1) and an encoded value $\widetilde{b}$, returns the encoding $\widetilde{m}$ associated to m_b;
- $\mathsf{Decode}(\widetilde{m})$: Given $\widetilde{m}$, returns the message m_b.

Security Properties. Two main security notions are expected: the sender-privacy, which hides m_{1-b} to the receiver, and the receiver-privacy, which hides b to the sender. We will focus on two different cases: receiver-privacy against a malicious sender; and sender-privacy against an honest-but-curious receiver. This will be enough for our application to decision trees, where the server (receiver) will be considered honest-but-curious while the client will possibly behave maliciously (sender).

In particular, Even-Goldreich-Lempel [EGL82] proposed such an efficient oblivious transfer from any $\mathsf{IND\text{-}CPA}$ public-key encryption scheme PKE, with an efficient uniform sampling algorithm in the set of the public keys $\mathcal{K}$. It is secure against a malicious sender and an honest-but-curious receiver.

2.5 Garbled Circuits

A *Garbled Circuit* [Yao86] is a primitive that allows two parties, a generator and an evaluator, to jointly compute a function over their respective private inputs. The computation to be performed must be modelled by a Boolean *circuit* using logic gates. Hereafter, we only consider AND or XOR logic gates namely gates with two input bits and one output bit. In the basic form of Garbled Circuits, for each logic gate, the generator generates a pair of random symmetric keys (k_0, k_1) for each input or output bit where k_0 and k_1 are respectively associated to the bit values 0 and 1. Each gate can be encoded (or *garbled*) using a symmetric encryption scheme by generating four ciphertexts where each ciphertext encrypts the output key corresponding to one output of the logic gate under the corresponding input keys. In practice, one can use the point-and-permute [BMR90], free-XOR [KS08] and half-gate [ZRE15] optimizations in order to reduce the number of ciphertexts, and even avoid the use of symmetric encryption with only hash functions. Given all the input keys corresponding to its input bits, the evaluator can recursively get the output key of the last gate thus retrieving the Boolean circuit outcome.

While the generator knows the input keys corresponding to its input α, and can then provide them to the evaluator with all the ciphertexts, it does not know the input β of the evaluator. The input keys corresponding to that input β are obtained using Oblivious Transfer, between the generator as the sender, and the evaluator as the receiver. More details can be found in the appendix.

2.6 Secure Equality Test

Our protocol relies on a secure equality test in the malicious client setting. A garbled circuit testing the equality between two κ-bit values α and β can be computed by

$$\begin{aligned}(\alpha = \beta) &= (\alpha[\ell] = \beta[\ell], \forall \ell \in [\kappa]) = (\alpha[\ell] \oplus \beta[\ell] = 0, \forall \ell \in [\kappa]) \\ &= \left(\bigwedge_{\ell=1}^{\kappa} (\overline{\alpha[\ell] \oplus \beta[\ell]}) = 1 \right) \\ &= \left(\bigwedge_{\ell=1}^{\kappa} (\overline{\alpha[\ell]} \oplus \beta[\ell]) = 1 \right)\end{aligned}$$

Each equality test requires κ XOR-gates of arity 2 and a global AND-gate of arity κ, or $\kappa - 1$ AND-gates of arity 2. Using the free-XOR and half-gate optimizations, such an equality test can be computed using $2(\kappa - 1)$ ciphertexts. One may compare hashed values of α and β, of shorter length, at the cost of possibly false positive cases.

2.7 Zero-Knowledge Proofs

A *Zero-Knowledge Proof* (ZKP) is a protocol between a prover P, who wants to prove to a verifier V, that a given statement belongs to a language, with-

out leaking any information about the witness. It can be made non-interactive (NIZK). Such a proof must be sound, which means that no adversary can generate an acceptable proof when $x \notin L$, but with negligible probability; and zero-knowledge, which guarantees zero-leakage about the witness. More details can be found in the appendix.

3 Honest-but-Curious Client and Server

In this section, we describe a protocol providing secure decision forests evaluation in the case where both the client and the server are honest-but-curious. In this setting, participants genuinely follow the protocol but may attempt to learn information from legitimately received messages. The protocol allows to evaluate the client inputs $\boldsymbol{x} = (x_{i,j})_{i\in[P],j\in[\delta]}$ with respect to the server model represented as binary sequences of comparisons $\mathcal{M} = (P, \delta, \nu, \tau, (t_{i,j}, v_{i,j})_{i\in[P],j\in[\delta]})$. The client should not learn anything on the threshold τ or the comparisons performed by the model which are described using $t_{i,j}$ and $v_{i,j}$ respectively. The server should not learn anything regarding the client inputs $x_{i,j}$. Ideally, the server should only learn the outcome of the evaluation but we tolerate the leakage of the number of paths successfully evaluated by the model so that the server can make a decision according to the threshold τ on the number of successful paths. We have already noted this corresponds to the number of accepting trees, which helps to get a confidence score for the decision.

3.1 Protocol Description

As illustrated on Figure 2, our protocol can be seen as a tuple of algorithms (KeyGen, EncodeModel, EvaluatePaths, RandomizeScores, EvaluateModel) where KeyGen, EncodeModel and EvaluateModel are computed by the server while EvaluatePaths and RandomizeScores are computed by the client. The EncodeModel algorithm is a preprocessing step that returns an encoded model $\boldsymbol{C}$ from the server secret key sk and the model $\mathcal{M}$. The encoded model $\boldsymbol{C}$ is used along with the public key pk and client inputs $\boldsymbol{x}$ by the EvaluatePaths algorithm in order to compute the encoded scores $\boldsymbol{S}$ of each path of the model. Next, these encoded scores are randomized and permuted by the RandomizeScores algorithm which outputs the randomized scores $\widetilde{\boldsymbol{S}}$. The server ends the protocol by computing the EvaluateModel algorithm that takes the secret key sk, the randomized scores $\widetilde{\boldsymbol{S}}$ and the threshold τ as inputs and returns the outcome of the evaluation of $\boldsymbol{x}$ with respect to $\mathcal{M}$.

During the EncodeModel preprocessing step, a ciphertext $C_{i,j}^k$ is computed for each comparison node (i, j) of the model (where i is the index of the path and j the depth of the node) and each possible input value $k \in [2^\nu]$ as follows:

$$C_{i,j}^k = \begin{cases} \mathsf{AHE.Enc}(\mathsf{pk}, 1 - v_{i,j}) & = [\![1 - v_{i,j}]\!] \quad \text{if } k \leq t_{i,j} \\ \mathsf{AHE.Enc}(\mathsf{pk}, v_{i,j}) & = [\![v_{i,j}]\!] \quad \text{otherwise.} \end{cases}$$

As $v_{i,j} = 1$ when the expected comparison result is $(x_{i,j} \leq t_{i,j})$ and $v_{i,j} = 0$ otherwise, $C_{i,j}^k$ is a ciphertext of 0 (respectively a ciphertext of 1) whenever

the input value k satisfies (respectively does not satisfy) the comparison. One can see that the number of $C_{i,j}^k$ ciphertexts is exponential with respect to ν but we stress that one only needs a few bits of precision in order to get a meaningful outcome. This number is thus linear in the number of comparisons in practice. During the EvaluatePaths step, the client retrieves the ciphertexts $C_{i,j}^{x_{i,j}}$ using its inputs $x_{i,j}$ and uses them to compute the encrypted score $[\![S_i]\!]$ of each path i. Such scores are ciphertexts of 0 if all the comparisons of the path are successful and ciphertexts of a non-zero value otherwise. We indeed stress that the ciphertexts encode the negation of the result of the comparison: 0 if true and 1 if false. As soon a false comparison happens, the sum S_i becomes non-zero.

The RandomizeScores step guarantees client privacy by randomizing and permuting the encrypted scores $[\![S_i]\!]$ without altering the fact that $S_i = 0$ if the path i is successful. During the EvaluateModel step, the server decrypts the randomized scores $[\![\widetilde{S}_i]\!]$ in order to retrieve the number of successful paths and returns the outcome of the model evaluation with respect to the threshold τ.

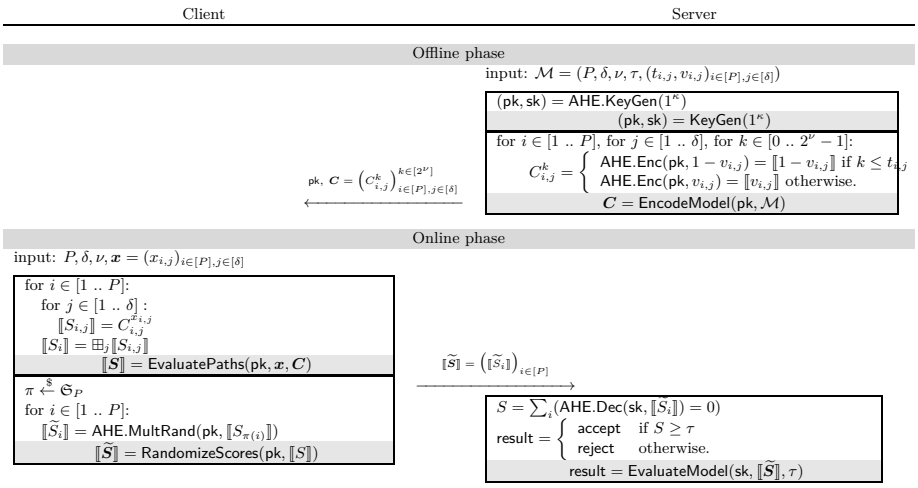

Figure 2: Secure decision forest evaluation for Honest-but-Curious Client and Server

3.2 Protocol Security

Correctness and Soundness. The correctness directly follows from the construction of the ciphertexts. If all the comparisons are correct, each $S_{i,j}$ is equal to 0 and the sum S_i is 0 (correctness), otherwise, at least one $S_{i,j}$ equals 1 and S_i does not equal 0 (soundness). After the client randomizes the S_i, zeros are still zeros, while other S_i become random values. When decrypting, the server counts the zeros. The threshold τ is applied for the final decision.

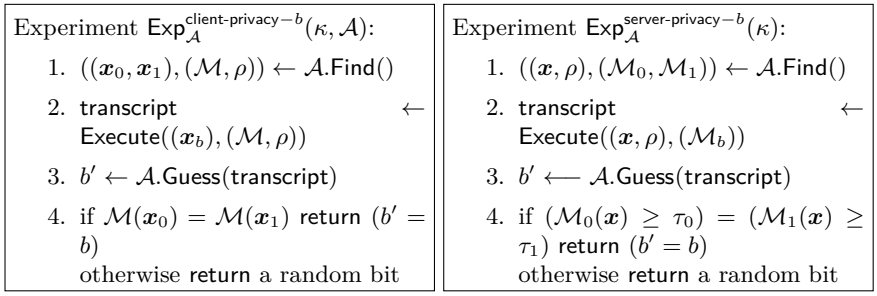

Figure 3: Client-Privacy Security Game (left) and Server-Privacy Security Game (right), in the Honest-but-Curious Setting

Client Privacy. An honest-but-curious server should not learn any client secret information, except what it can learn from the outcome: the number of successful paths (See Figure 3, on the left).

We thus consider an adversary against the privacy of the client: it first chooses a model $\mathcal{M}$ for the server, and two sets of possible inputs $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ for the client. It also provides the random tape ρ of the server. The adversary sees the transcript between a server using $\mathcal{M}$ and ρ, and a client using $\boldsymbol{x}_b$ for a random bit b, and it should guess b. There is the natural restriction that $\mathcal{M}(\boldsymbol{x}_0) = \mathcal{M}(\boldsymbol{x}_1)$. The random tape ρ will be used by the server for encoding the model $\mathcal{M}$.

For our scheme, the client privacy is provided thanks to the permutation and randomization of the encrypted scores: from the expected outcome, one can encrypt the correct number of 0, and the other values are non-zero random values. One can then randomize and permute them. This is indistinguishable from the server point of view.

Server Privacy. An honest-but-curious client should not learn any server secret information, except what it can learn from the outcome namely the accept or reject decision (See Figure 3, on the right).

Hence, we consider an adversary that chooses some inputs $\boldsymbol{x}$ for the client and its random tape ρ, but two different models $\mathcal{M}_0 = (P, \delta, \nu, \boldsymbol{t}_0, \boldsymbol{v}_0, \tau_0)$ and $\mathcal{M}_1 = (P, \delta, \nu, \boldsymbol{t}_1, \boldsymbol{v}_1, \tau_1)$, with the constraint that evaluating $\boldsymbol{x}$ with respect to the two models $\mathcal{M}_0$ and $\mathcal{M}_1$ should produce the same result. The random tape ρ will be used by the client for randomizing the ciphertexts. The adversary should then distinguish transcripts involving the two models.

For our scheme, the server's privacy is provided by the encryption of the model in $\boldsymbol{C}$, during the offline phase. The private server's information are the thresholds $t_{i,j}$ and the Boolean values $v_{i,j}$ for each comparison as well as the final threshold τ. One can note that the client learns which feature is used in a given comparison. This can be avoided by adding dummy comparisons so that each feature (or many features) is used in every path as discussed previously.

For the formal proof, as the scheme leaks the number of accepting paths, this is used by the simulator for the final outcome, without needing the decryption key. Then, as the decryption key is not known anymore, using IND-CPA, we can replace all the ciphertexts in the offline phase by encryptions of 0: the client cannot learn anything anymore.

4 Malicious Client and Honest-but-Curious Server

Unfortunately, a malicious client could trivially bias the outcome of the protocol described in Figure 2 by setting all the $[\![\widetilde{S}_i]\!]$ as encryptions of zeros so that he will be accepted by the server independently of its inputs. He knows accepting paths should encrypt 0, he can force that in his unique flow to the server.

In this section, we describe a protocol providing secure decision forests evaluation even if the client behaves maliciously in order to get accepted, while the server is still honest-but-curious. Our security goals remain unchanged from Section 3, however the client may now deviate from the protocol to influence the evaluation outcome. In order to secure our protocol, we add some randomness within the model through the notion of *path polarity* and rely on secure equality tests.

4.1 Protocol Description

In order to avoid the above attack, we introduce the notion of path polarity p_i: the client cannot predict anymore the expected outcome of a path. With thus now use an enriched model $\mathcal{M} = (P, \delta, \nu, \tau, (t_{i,j}, v_{i,j}, p_i)_{i\in[P],j\in[\delta]})$, with ternary sequences of comparisons (as shown of Figure 1) with a polarity p_i. Indeed, to be able to exploit the polarity, a path should have three possible outcomes: 'accept', 'reject', 'ignore'. Then, when a path is positive ($p_i = 1$), according to the computed value, -1, 0, or +1, it will be 'reject', 'ignore', or 'accept', respectively; if the path is negative ($p_i = -1$), according to the computed value, -1, 0, or +1, it will be 'accept', 'ignore', or 'reject', respectively. From a binary tree, such a path is now the path down to the last node that has two distinct leaves: on an input $\boldsymbol{x}$, if it does not reach the last node (some of the comparisons fails before), one outputs 0, otherwise one outputs -1 or +1, whether the leaf is rejecting or accepting. A tree of depth δ has at most $2^{\delta-1}$ such disjoint paths: an input $\boldsymbol{x}$ must be accepted/rejected by exactly one path only, all the other paths should output 'ignore'. It is possible to extract such paths from any binary decision tree, by possibly adding some 'always true'/'always false' nodes. Such 'always true'/'always false' comparisons will also be added to hide the actually used features. This will lead to an impossibility for the malicious client to guess the outcome of a path from the features used in the comparisons: the client does not know if the comparison is really exploited, and the client does not know the polarity and thus whether it should force +1 or -1 to increase the score.

Intuitively, the best attack of the adversary is by guessing the polarity of the paths to hope to pass the threshold. But as the polarity is random and

hidden (as no information leaks, as proven later), the sum of the outputs of a malicious client will follow a binomial distribution with bias 1/2. And the expected sum is 0. If we set the threshold not too low, the probability to get accepted is negligible. An alternative is also to add some 'always accepting' paths, to artificially increase the expected sum of an honest user. With 20 such paths, we can set $\tau = 20$. Let us thus consider 120 paths with a threshold $\tau = 20$ (with either additional 'always accepting' paths, or an initially high threshold): to pass the threshold, one needs 20 correct guesses (probability bounded by $1/10^6$, on exactly 20 non-zero outputs), or at least 70 successes among 120 (probability less than 3%, with random -1/1 outputs). This remains reasonable with respect to usual accuracy of such models.

To evaluate a path on an input $\boldsymbol{x}$, with a ternary result, we need different weights in each comparisons: the values encrypted in the $C^k_{i,j}$ will be 1 or 0 for all the active comparisons except the last node (in red on Figure 1), that will contain δ or 0, where δ is the length of the paths: if all the comparisons pass, the sum is $2\delta - 1$ and corresponds to +1, if all but the last comparison pass, the sum is $\delta - 1$ and corresponds to -1, all the other cases will lead to a sum between 0 and $\delta - 2$ or δ and $2\delta - 2$ and correspond to 0. In the following, we will show how the conversion of the sum being $2\delta - 1$, $\delta - 1$, or anything else can be converted into +1, -1 and 0, respectively, using a verifiable Garbled Circuit.

To make a path with negative polarity, we just invert the comparison in the last node. The inversion by the server will restore the correct value. We stress that the nodes, after the $C^k_{i,j}$ have been generated, can be randomly permuted to hide which feature is involved in the final node. Furthermore, we remind that since the complexity of our protocol will be independent of the length of the path, any additional comparisons will have no impact to the online phase: we can use them to hide the real structure of the paths and make random guesses of path polarities the best attack for the adversary.

Unfortunately, one cannot rely on path permutations anymore to enforce client privacy once path polarity is used. Indeed, the server needs to know for which path a score has been computed in order to later involve the correct path polarity p_i. To overcome this issue, we rely on a secure equality test based on garbled circuits along with an oblivious transfer, to obliviously convert the above sums between 0 and $2\delta - 1$ into another ciphertext (under the client key) of +1, -1 or 0.

Hereafter, we use $[\![m]\!]_{\mathsf{S}}$ (respectively $[\![m]\!]_{\mathsf{C}}$) to denote an encryption of m under the public key of the server (respectively, the client) in order to avoid any confusion. The client starts by computing the path score $\boxplus_j [\![S_{i,j}]\!]_{\mathsf{S}}$ as previously and masks it with a random value α_i in order to obtain $[\![\beta_i]\!]_{\mathsf{S}} = [\![\alpha_i + S_i]\!]_{\mathsf{S}}$. As $(S_i = \delta - 1) \Leftrightarrow (\alpha_i + \delta - 1 = \beta_i)$ and $(S_i = 2\delta - 1) \Leftrightarrow (\alpha_i + \delta - 1 = \beta_i - \delta)$, the client prepares a garbled circuit testing equality of $\alpha_i + \delta - 1$ with β_i and with $\beta_i - \delta$. The server computes β_i by decrypting $[\![\beta_i]\!]_{\mathsf{S}}$ and retrieves the corresponding circuit inputs using an oblivious transfer, thus allowing it to evaluate the aforementioned equality tests, where $\alpha_i + \delta - 1$ is the common input from the client, and β_i and $\beta_i - \delta$ are the inputs from the server. Note that in our ElGamal setting, one may use g^{β_i} instead of β_i for the comparisons,

which avoids the server to compute a discrete logarithm during the decryption.

Each of the outcomes of the garbled circuit is mapped to AHE ciphertexts, under the client key, either $([\![-1]\!]_{\mathsf{C}}, [\![+1]\!]_{\mathsf{C}})$ or $([\![+1]\!]_{\mathsf{C}}, [\![-1]\!]_{\mathsf{C}})$, according to a random choice, along with a pair of NIZK proving that those ciphertexts are actually encryptions of both $+1$ and -1, without revealing the order.

The first equality test labels output $+1$ in the positive case and -1 otherwise; while the second ones output -1 in the positive case and $+1$ otherwise: the average of the two values is $+1$ if the sum is $2\delta - 1$; -1 if the sum is $\delta - 1$; and 0 otherwise. One can thereafter apply the polarity factor p_i to the ciphertext σ_i of the above mean, to restore the real encrypted outcome of the path, under the client key: the server gets, for each path, $[\![+1]\!]_{\mathsf{C}}$, $[\![0]\!]_{\mathsf{C}}$, or $[\![-1]\!]_{\mathsf{C}}$. All the products $p_i \boxdot \sigma_i$ are summed up into $[\![S_\Omega]\!]_{\mathsf{C}}$, initialized to a random value θ. Hence, $S_\Omega = \theta + \sum_i S_i$, where $S_i \in \{-1, 0, +1\}$ is the outcome of each path. The server will ask the client to help in decrypting this ciphertext, but after having applied a random blinding factor $\zeta \in \mathbb{Z}_p^*$, to get back $\zeta(\theta + \sum_i S_i)$. The server can remove ζ and θ: if the client cheated, the result is random, otherwise this is the number of accepting trees minus the number of rejecting trees. One accepts if this number is between the threshold τ and the number T of trees. In case the client cheats, the probability to be in this window is less than T/p, which is negligible. Again, we stress that discrete logarithms are not needed to check the value is in the window, as the latter is small enough. One can deal with group elements, and not scalars. The global protocol is described in the appendix.

4.2 Protocol Security

Correctness and Soundness. The correctness follows the above analysis, where the two equality tests conclude into ciphertexts of -1, $+1$, or 0, and the path polarity $p_i \in \{-1, 1\}$ is thereafter applied to obtain $+1$ in the accepting case, -1 in the rejecting case, and 0 to ignore the path.

Because of the polarity, we prevent the server from a client arbitrarily choosing the outcome of a path. Indeed, in contrast to the honest-but-curious case where expected paths values were zero, the expected value sent to the server (the outcome of the garbled circuit) will depend on the path polarity: $+1$ if the path has positive polarity, or -1 for the negative polarity, to be an accepting path. 0 values will lead to ignore the path. The paths cannot all be ignored, otherwise, there is no change to be above the threshold τ, hence the two extreme attacks presented before: either the client specifically guesses τ values to be correct, and set all the other to zero (with a success probability bounded by $2^{-\tau}$), or the client tries non-zero values for all the outputs and the success probability follows a binomial distribution with parameters $(P, 1/2)$, where the number of successes must be greater than $(P + \tau)/2$.

The garbled circuits will evaluate the initial scores before polarity, and the unknown polarity bit p_i will restore the exact outcome of the path. A malicious client has no other choice than a random guess of the polarity to fake the output labels of the garbled circuits. He could cheat with a bad encoding of the circuit

to bias the output, but only with $+1/-1$ or $-1/+1$ as the output table is proven to contain encryptions of $+1$ and -1 with a zero-knowledge proof. But since the player has no idea about the polarity bit p_i, the final outcome for the path is -1 or $+1$ with identical probability, if positive and negative polarities are balanced. Hence, alteration of the result of a path, without knowing the polarity, will make the sum closer to 0. If the threshold is not too close to 0, the probability for the adversary to impersonate the user is negligible. Or at least, the impact of the malicious behaviour of the client on false positive outcome will be small, compared to initial accuracy of the system (in clear).

Of course, another cheating strategy can be sending a false zero-knowledge proof or wrong ciphertexts for the garbled circuit gates. This would lead to failure attacks with a random value in S_Ω (enforced in the protocol). Similarly, an incorrect decryption of $[\![\widetilde{S_\Omega}]\!]_{\mathsf{C}}$ would lead to a random value for $\zeta^{-1} \cdot S_\Omega - \theta$, with the detection probability greater than $1 - T/p$, which is overwhelming. This concludes in a reject.

As a consequence, we just have to take care of the accuracy of the model in the clear, for an honest execution, and we will also have to consider the impact of malicious behaviours on the false positive decisions, which is the most critical in the case of continuous authentication. In some other applications, false negative decisions might be more important to limit (such as for spam detection).

Client Privacy. We are still considering the client privacy against an honest-but-curious server. However, in this protocol, we no longer use permutations, because of the path polarity. However, from the client privacy of the oblivious transfers in the garbled circuits, there is no leakage about the α_i's. Then the outcome S_i of the path is encrypted under the client key, which hides it from the server. Eventually, the server only gets the decryption of $\sum_i S_i$, which is the number of accepting trees, the expected result to obtain the classification with confidence score.

Server Privacy. Our main goal was the soundness against a malicious client that would try get falsely accepted. However, this is also important, for our argument of random guess only of the path polarities as the best attack, to show that the adversary cannot learn anything that could help him to make a better guess than at random. Eventually, the client does not get back the decision, so if all the received information looks random, we have proven server privacy.

The first messages received by the client are the encryptions of the comparison gates. Under the indistinguishability of the public-key encryption scheme, they do not leak any information about these gates. Then, the server sends encodings for his inputs β_i and $\beta_i - \delta$, which are just keys for the garbled circuit. This does not reveal any information about them to the client. Eventually, the client receives the encryption of $\widetilde{S_\Omega}$, which is randomized by θ and ζ. The latter is used to avoid the client to increase his score after decryption while the former completely hides the real value of $\sum_i S_i$

As a consequence, the view of the client does not contain any information about the model, nor the outcome. Of course, the information to be known to be client is which feature is used in each comparison, in order to use the appropriate $x_{i,j}$. But again, because of possible dummy comparisons, and the random permutations of the comparisons along a path, the client cannot know which gates are real gates, and which gate is the last critical gate.

5 Performances and Applications

5.1 Storage and Bandwidth Costs

In this section, we describe the storage and bandwidth cost of our protocols. Let $\lambda_{\mathsf{OT}}^{\mathsf{R}} = |\tilde{b}| = |\mathsf{OT.Encode}(b)|$, $\lambda_{\mathsf{OT}}^{\mathsf{S}} = |\mathsf{OT.Compute}(\tilde{b})|$ and $\lambda_{\mathsf{AHE}} = |[\![m]\!]_{\mathsf{S}}| = |[\![m]\!]_{\mathsf{C}}|$.

Offline Storage Cost. During the offline phase, the server sends $\boldsymbol{C}$ to the client which requires to store $2^{\nu} \cdot \delta \cdot P \cdot \lambda_{\mathsf{AHE}}$ bits. Using lifted ElGamal with Elliptic Curves and p over 256 bits as $\kappa = 128$, one has $\lambda_{\mathsf{AHE}} = 512$. With inputs over $\nu \in [2, \ldots, 8]$ bit, depth $\delta \in [2, \ldots, 16]$, and P ranging from 10 to 100, the storage is between a few KB and a few MB.

Honest-but-Curious Bandwidth Cost. During the online phase, the client sends P ciphertexts (one for each path score) to the server. Using the lifted ElGamal AHE, the message sent by the client is of size $P \cdot 512$ bits (see Table 2).

Malicious Bandwidth Cost. During the online phase, the client sends P tuples, each formed with a ciphertext of size λ_{AHE} as the path score along with the AND-gate encodings. To reduce communication costs, we use a hash function on the garbled circuit inputs to be compared, with output length λ_{GC}. Thus, the client will sent $\lambda_{\mathsf{GC}} - 1$ AND-gate encodings, the λ_{GC} input labels (where each label is a hash with size κ_H) and the transition table consisting in 4-tuple with a garbled circuit output label, a ciphertext, and a ZKP.

This results for each path in the following bit-length:

$$\underbrace{\lambda_{\mathsf{AHE}}}_{|[\![S_i]\!]|} + \underbrace{\kappa_H \cdot 2(\lambda_{\mathsf{GC}} - 1)}_{|\mathfrak{C}_i|} + \underbrace{\lambda_{\mathsf{GC}} \cdot \kappa_H}_{|\mathfrak{I}_{\alpha_i}^{\mathsf{G}}|} + \underbrace{2 \cdot (2\kappa_H + \lambda_{\mathsf{AHE}}))}_{|\mathfrak{T}_i|}$$

which is $3(\lambda_{\mathsf{AHE}} + \lambda_{\mathsf{GC}} \cdot \kappa_H) + 2\kappa_H$. With $\kappa_H = 256$, $\lambda_{\mathsf{GC}} = 64$ and $\lambda_{\mathsf{AHE}} = 512$, the first client's message can be expressed as $51200P$ bits, i.e. 6.25 KB per path. During the oblivious transfers, the server encodes $P \times |\beta_i| = P \cdot \lambda_{\mathsf{GC}}$ bits. As a consequence, he sends to the client a message of $P \cdot \lambda_{\mathsf{GC}} \cdot \lambda_{\mathsf{OT}}^{\mathsf{R}}$ bits. Using classical ElGamal as PKE in the oblivious transfer, one has $\lambda_{\mathsf{OT}}^{\mathsf{R}} = 512$ and $\lambda_{\mathsf{OT}}^{\mathsf{S}} = 1024$ resulting in a 4 KB long message for each path. The client responds with a $P \cdot \lambda_{\mathsf{GC}} \cdot \lambda_{\mathsf{OT}}^{\mathsf{S}}$ bit long message corresponding to 8 KB per path. Then, the server sends the encrypted randomized score to the client who returns the plaintext

value he retrieves when decrypting. Table 2 shows the total bandwidth cost for each party in both protocols, according to the number P of paths.

Table 2: Communications During the Online Phase.

P	50	100	150	200
Honest-but-Curious				
Client	3.13 KB	6.25 KB	9.38 KB	12.5 KB
Server	0 bit			
Malicious				
Client	712.5 KB	1.4 MB	2.1MB	2.8MB
Server	200 KB	400 KB	600KB	800KB

5.2 Application to Continuous Authentication and Spam Filtering

We run our tests in Python with the scikit-learn library, using 75% of the dataset as the training set and the remaining 25% as the testing set. We optimize the training with the Orthogonal Matching Pursuit Algorithm [MZ93]: we generate 100 times more trees than expected. We then apply the OMP algorithm on the global set, such that the outcome is the best linear combination with the expected number of trees.

We first deal with continuous authentication. We used an internal database of 20531 samples splitted in 35 profiles built with 222 features. In this context, low *False Positive Rate* (FPR) is privileged to low *False Negative Rate* (FNR), since it is preferable to ask the client to use a second authentication factor rather than being impersonated. Moreover, high accuracy for each test is not required, as multiple tests will amplify the quality. Table 3 shows the mean results ont the 35 profiles (where, for a given profile, all other profiles are considered as imposter), depending on the number of paths P and the depth of the model δ, while ν is set to 6, leading to 64 ciphertexts for each comparison, stored by the

Table 3: Accuracy on our continuous authentication Database

Γ		50%			55%		
T	δ	FPR	FNR	F1 Score	FPR	FNR	F1 Score
10	6	0.02	0.15	0.92	0.02	0.16	0.91
	8	0.01	0.19	0.91	0.01	0.19	0.91
25	6	0.04	0.13	0.92	0.03	0.15	0.92
	8	0.02	0.14	0.92	0.02	0.17	0.91

60%			65%		
FPR	FNR	F1 Score	FPR	FNR	F1 Score
0.01	0.23	0.89	0.01	0.23	0.89
0.01	0.25	0.88	0.01	0.26	0.88
0.02	0.2	0.9	0.01	0.23	0.89
0.01	0.24	0.89	0.01	0.26	0.88

Table 4: Accuracy on the spambase Database

Γ		50%			55%		
T	δ	FPR	FNR	F1 Score	FPR	FNR	F1 Score
10	2	0.02	0.24	0.88	0.01	0.28	0.87
	4	0.03	0.20	0.89	0.04	0.18	0.90
25	2	0.03	0.24	0.88	0.02	0.25	0.88
	4	0.04	0.19	0.89	0.04	0.21	0.88

60%			65%		
FPR	FNR	F1 Score	FPR	FNR	F1 Score
0.02	0.24	0.88	0.01	0.45	0.81
0.02	0.27	0.87	0.01	0.30	0.86
0.01	0.47	0.80	0.00	0.56	0.78
0.02	0.23	0.89	0.01	0.34	0.85

client. Also, we consider several values for the acceptation threshold (Γ) (which equals 50% by default, for the simple majority).

We compute the FPR and FNR, then the accuracy is defined as the F1-score (defined by $(1-\mathsf{FPR})/(1+(\mathsf{FNR}-\mathsf{FPR})/2)$). Random decision would lead to an accuracy of $1/2$, and perfect filter should have F1-score equal to 1. We determine the best accuracy depending on those parameters in Table 3. For the honest-but-curious security setting, there is no constraint on the threshold, while against malicious clients, the higher the threshold is, the higher the security level is against active impersonation attempts.

Secondly, we worked on the spambase database [HRFS99] (with 4601 samples $\times$ 57 features) which determines if an email should be considered as spam or not. Results are shown in Table 4

6 Conclusion

In this paper, we proposed new constructions to securely evaluate decision forests with two output classes. As we targeted applications where the interactions between the client and the server should be as low as possible (both in term of number of rounds and online bandwidth cost), we designed two-steps protocols in which some part of the computation can be performed offline. This introduces an interesting trade-off between the storage and the number of rounds during the online step of the protocol.

References

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EURO-*

CRYPT 2014, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014.

[BFK+09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In Michael Backes and Peng Ning, editors, *ESORICS 2009*, volume 5789 of *LNCS*, pages 424–439. Springer, Heidelberg, September 2009.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 498–507. ACM Press, October 2007.

[BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS 2015*. The Internet Society, February 2015.

[DDH+16] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson C. A. Nascimento, Stacey C. Newman, and Wing-Sea Poon. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on precomputation. Cryptology ePrint Archive, Report 2016/736, 2016. `https://eprint.iacr.org/2016/736`.

[DPB18] Aurélien Dupin, David Pointcheval, and Christophe Bidan. On the leakage of corrupted garbled circuits. In Joonsang Baek, Willy Susilo, and Jongkil Kim, editors, *ProvSec 2018*, volume 11192 of *LNCS*, pages 3–21. Springer, Heidelberg, October 2018.

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 205–210. Plenum Press, New York, USA, 1982.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

[HRFS99] Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermondt. *Spambase Data Set*, 1999. `http://archive.ics.uci.edu/ml/datasets/Spambase/`.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, Heidelberg, August 2013.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, Heidelberg, May 2007.

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 476–494. Springer, Heidelberg, August 2014.

[MF06] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473. Springer, Heidelberg, April 2006.

[MZ93] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[TKK19] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. Private evaluation of decision trees using sublinear cost. *PoPETs*, 2019(1):266–286, January 2019.

[TMZC17] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part II*, volume 10493 of *LNCS*, pages 494–512. Springer, Heidelberg, September 2017.

[WFNL16] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. Privately evaluating decision trees and random forests. *PoPETs*, 2016(4):335–355, October 2016.

[WMK17] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster secure two-party computation in the single-execution setting. In

Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 399–424. Springer, Heidelberg, April / May 2017.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

A Auxiliary Material

We first give a few more formal description of some advanced tools, to be used the global protocol, and then more details about the global security decision forest evaluation for malicious clients.

A.1 Advanced Primitives

Garbled Circuits From our modeling of them, we can define Garbled Circuits with two algorithms ($\mathsf{Generate}, \mathsf{Eval}$), in addition to an OT, as follows:

- $\mathsf{Generate}(\mathcal{C}, \alpha)$: Given a circuit $\mathcal{C}$ and the generator input α, returns:

  The ciphertexts corresponding to the gates of $\mathcal{C}$: $\mathfrak{C}$

  The input labels corresponding to the generator input α: $\mathfrak{I}_\alpha^{\mathsf{G}}$

  The input labels $\mathfrak{I}^{\mathsf{E}}$ corresponding to the possible inputs of the evaluator (but not published)

  The transition table mapping the output labels with arbitrary values: $\mathfrak{T}$;

- OT Execution: on input $\mathfrak{I}^{\mathsf{E}}$ from the generator (as sender) and the input β of the evaluator (as receiver), for the latter to receive the input labels $\mathfrak{I}_\beta^{\mathsf{E}}$;

- $\mathsf{Eval}(\mathfrak{C}, \mathfrak{I}_\alpha^{\mathsf{G}}, \mathfrak{I}_\beta^{\mathsf{E}}, \mathfrak{T})$: From the garbled circuit $\mathfrak{C}$, input labels $\{\mathfrak{I}_\alpha^{\mathsf{G}}, \mathfrak{I}_\beta^{\mathsf{E}}\}$ and transition table $\mathfrak{T}$, retrieve $\mathfrak{O}_b \in (\mathfrak{O}_0, \mathfrak{O}_1)$ by evaluating the garbled circuit $\mathfrak{C}$ with input labels $\mathfrak{I}_\alpha^{\mathsf{G}}, \mathfrak{I}_\beta^{\mathsf{E}}$ and return the value mapped by $\mathfrak{O}_b$ in $\mathfrak{T}$. If the garbled circuit evaluation fails (the output value is not in $(\mathfrak{O}_0, \mathfrak{O}_1)$, return $\perp$, as a failure outcome.

The generator should not learn any information about the evaluator input α, while the evaluator should not learn any information about the generator input β, except what the result reveals. These privacy notions rely on the privacy of the OT: with receiver-privacy against a malicious sender, we then get the privacy of the evaluator against a malicious generator, and with sender-privacy against an honest-but-curious receiver, we get the privacy of the generator against an honest-but-curious evaluator. Another important notion is of course the correct evaluation of the function, a.k.a. the soundness. Since the evaluator is interested in the correct result, the attack can come from the generator that provides a wrong encoding of the circuit. A classical technique to avoid incorrect circuits is based on a costly cut-and-choose. One of our contributions is an efficient alternative in our particular case.

Zero-Knowledge Proofs A *Zero-Knowledge Proof* (ZKP) is an interactive protocol between a prover P, who wants to prove to a verifier V, that a given statement belongs to a language, without leaking any information about the

witness. It can be made non-interactiveWe will consider Non-Interactive Zero-Knowledge proofs (NIZK):

- $\mathsf{ZKGen}(x, L, w)$ outputs π_x, a zero-knowledge proof of the statement ($x \in L$), using the witness w;
- $\mathsf{ZKVerify}(x, L, \pi_x)$ verifies if π_x is a correct proof of the statement ($x \in L$). Return accept if true, reject otherwise;
- $\mathsf{ZKSim}(x, L)$ simulates a proof π_x of any (possibly false) statement ($x \in L$) without any witness.

A NIZK requires three properties:

- **Completeness**: ZKGen always generate an acceptable proof when $x \in L$;
- **Soundness**: no adversary can generate an acceptable proof when $x \notin L$, but with negligible probability;
- **Zero-Knowledge**: using possibly a different setup, $\mathsf{ZKSim}(x, L)$ generates proofs that are indistinguishable to proofs generated by $\mathsf{ZKGen}(x, L, w)$, on valid statements but without the witness.

A.2 Detailed Protocol

In this section we give a detailed description of the protocol in the malicious client setting. We first describe the steps executed by the server then the steps done by the client. We recall that $[\![m]\!]_{\mathsf{S}} = \mathsf{AHE.Enc}(\mathsf{pk_S}, m)$ and $[\![m]\!]_{\mathsf{C}} = \mathsf{AHE.Enc}(\mathsf{pk_C}, m)$.

a) Client steps
input : $P, \delta, \nu, \boldsymbol{x} = (x_{i,j})_{i\in[P], j\in[\delta]}$
roles: GC: Generator , OT: Sender

$(\mathsf{pk_C}, \mathsf{sk_C}) = \mathsf{AHE.KeyGen}(1^\kappa)$
$(\mathsf{pk_C}, \mathsf{sk_C}) = \mathsf{KeyGen_C}(1^\kappa)$
for $i \in [1 \,..\, P]$: $\alpha_i \xleftarrow{\$} \mathbb{Z}_p$
$(\mathfrak{C}_i, \mathfrak{I}^{\mathsf{G}}_{\alpha_i}, \mathfrak{I}^{\mathsf{E}}_i, \mathfrak{T}_i) \leftarrow \mathsf{GC.Generate}(\mathsf{pk_C}, \mathcal{C}_{EQ}, \alpha_i)$
$(\boldsymbol{\alpha}, \mathfrak{C}, \mathfrak{I}^{\mathsf{G}}_{\boldsymbol{\alpha}}, \mathfrak{I}^{\mathsf{E}}, \mathfrak{T}) = \mathsf{InitializeGC}(\mathsf{pk_C})$
for $i \in [1 \,..\, P]$:
for $j \in [1 \,..\, \delta]$: $[\![S_{i,j}]\!]_{\mathsf{S}} = C_{i,j}^{x_{i,j}}$
$[\![\beta_i]\!]_{\mathsf{S}} = [\![\alpha_i]\!]_{\mathsf{S}} \boxplus \left(\boxplus_j [\![S_{i,j}]\!]_{\mathsf{S}}\right)$
$[\![\boldsymbol{\beta}]\!]_{\mathsf{S}} = \mathsf{EvaluatePaths}(\mathsf{pk_S}, \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{C})$
for $i \in [1 \,..\, P]$, for $k \in [1 \,..\, \lambda_{\mathsf{AHE}}]$:
$\widetilde{\mathfrak{I}^{\mathsf{E}}_{i,k}} \leftarrow \mathsf{OT.Compute}(\mathfrak{I}^{\mathsf{E}}_i, \widetilde{\beta}^k_i)$
$\widetilde{\mathfrak{I}^{\mathsf{E}}} = \mathsf{ComputeOT}(\mathfrak{I}^{\mathsf{E}}, \widetilde{\boldsymbol{\beta}})$

$\widetilde{S_\Omega} \leftarrow \mathsf{AHE.Dec}(\mathsf{sk_C}, [\![\widetilde{S_\Omega}]\!]_\mathsf{C})$

$\widetilde{S_\Omega} = \mathsf{DecryptScore}(\mathsf{sk_C}, [\![\widetilde{S_\Omega}]\!]_\mathsf{C})$

b) Server steps

input: $\mathcal{M} = (P, \delta, \nu, \tau, (p_i, t_{i,j}, v_{i,j})_{i\in[P], j\in[\delta]})$

roles: GC: Evaluator , OT: Receiver

$(\mathsf{pk_S}, \mathsf{sk_S}) = \mathsf{AHE.KeyGen}(1^\kappa)$

$(\mathsf{pk_S}, \mathsf{sk_S}) = \mathsf{KeyGen_S}(1^\kappa)$

for $i \in [1 \,..\, P]$: $\pi \leftarrow \mathfrak{S}_\delta$

for $k \in [0 \,..\, 2^\nu - 1]$, for $j \in [1 \,..\, \delta - 1]$:

$$C^k_{i,\pi(j)} = \begin{cases} [\![1 - v_{i,j}]\!]_\mathsf{S} \text{ if } k \le t_{i,j} \\ [\![v_{i,j}]\!]_\mathsf{S} \text{ otherwise.} \end{cases}$$

$$C^k_{i,\pi(\delta)} = \begin{cases} [\![((1 + p_i)/2 - v_{i,\delta} p_i) \cdot \delta]\!]_\mathsf{S} \text{ if } k \le t_{i,j} \\ [\![((1 - p_i)/2 + v_{i,\delta} p_i) \cdot \delta]\!]_\mathsf{S} \text{ otherwise.} \end{cases}$$

$\boldsymbol{C} = \mathsf{EncodeModel}(\mathsf{pk_S}, \mathcal{M})$

for $i \in [1 \,..\, P] : \beta_i = \mathsf{AHE.Dec}(\mathsf{sk_S}, [\![\beta_i]\!]_\mathsf{S})$

for $k \in [1 \,..\, \lambda_{\mathsf{AHE}}] : \widetilde{\beta}^k_i = \mathsf{OT.Encode}(\beta_i[k])$

$\widetilde{\boldsymbol{\beta}} = \mathsf{EncodeOT}(\mathsf{sk_S}, [\![\boldsymbol{\beta}]\!]_\mathsf{S})$

$\theta \xleftarrow{\$} \mathbb{Z}_p$; $[\![S_\Omega]\!]_\mathsf{C} \leftarrow [\![\theta]\!]_\mathsf{C}$

for $i \in [1 \,..\, P]$:

for $k \in [1 \,..\, \lambda_{\mathsf{AHE}}] : \mathfrak{I}^{\mathsf{E}}_{\beta^k_i} \leftarrow \mathsf{OT.Decode}(\widetilde{\mathfrak{I}^{\mathsf{E}}_{i,k}})$

if $\mathsf{GC.Eval}(\mathfrak{C}_i, \mathfrak{I}^{\mathsf{G}}_{\alpha_i}, \mathfrak{I}^{\mathsf{E}}_{\beta_i}, \mathfrak{T}_i) = \perp$:

$\theta' \xleftarrow{\$} \mathbb{Z}_p$; $[\![S_\Omega]\!]_\mathsf{C} \leftarrow [\![\theta']\!]_\mathsf{C}$

else: $((\sigma_0, \pi_0), (\sigma_1, \pi_1)) \leftarrow \mathsf{GC.Eval}(\mathfrak{C}_i, \mathfrak{I}^{\mathsf{G}}_{\alpha_i}, \mathfrak{I}^{\mathsf{E}}_{\beta_i}, \mathfrak{T}_i)$

if $\mathsf{ZKVerify}(((\sigma_0, \pi_0), (\sigma_1, \pi_1))) = \mathsf{accept}$:

$[\![S_\Omega]\!]_\mathsf{C} \leftarrow [\![S_\Omega]\!]_\mathsf{C} \boxplus (p_i \boxdot (\sigma_0 \boxplus \sigma_1))$

else: $\theta' \xleftarrow{\$} \mathbb{Z}_p$; $[\![S_\Omega]\!]_\mathsf{C} \leftarrow [\![\theta']\!]_\mathsf{C}$

$\zeta \xleftarrow{\$} \mathbb{Z}^*_p$; $[\![\widetilde{S_\Omega}]\!]_\mathsf{C} \leftarrow \zeta \boxdot [\![S_\Omega]\!]_\mathsf{C}$

$([\![\widetilde{S_\Omega}]\!]_\mathsf{C}, \theta, \zeta) = \mathsf{ComputeScore}(\mathsf{pk_C}, \mathcal{M}, \mathfrak{C}, \mathfrak{I}^{\mathsf{G}}_{\boldsymbol{\alpha}}, \widetilde{\mathfrak{I}^{\mathsf{E}}}, \mathfrak{T})$

$S \leftarrow \zeta^{-1} \cdot \widetilde{S_\Omega} - \theta$

if $S \in [\tau, T]$: $\mathsf{result} = 1$; else: $\mathsf{result} = 0$

$\mathsf{result} = \mathsf{ComputeResult}(\mathcal{M}, \theta, \zeta, \widetilde{S_\Omega})$

c) Protocol We now describe the protocol execution: