Privacy-preserving Cosine Similarity Computation with Malicious Security Applied to Biometric Authentication

Nan Cheng
nan.cheng@unisg.ch
University of St. Gallen
Switzerland

Melek Önen
melek.onen@eurecom.fr
EURECOM
France

Aikaterini Mitrokotsa
aikaterini.mitrokotsa@unisg.ch
University of St. Gallen
Switzerland

Oubaïda Chouchane
oubaida.chouchane@eurecom.fr
EURECOM
France

Massimiliano Todisco
massimiliano.todisco@eurecom.fr
EURECOM
France

Alberto Ibarrondo
alberto.ibarrondo@idemia.com
IDEMIA
France

ABSTRACT

Computing $\Delta(\boldsymbol{x}, \boldsymbol{y}) \geq \tau$, the distance between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ chained with a comparison to a predefined public threshold τ, is an essential functionality that is extensively used in privacy-sensitive applications such as biometric authentication and identification, machine learning algorithms (*e.g.,* linear regression, k-nearest neighbors etc.) or typo-tolerant password-based authentication. Cosine similarity is one of the most popular distance metrics employed in these settings. In this paper, we investigate the privacy-preserving computation of cosine similarity in a two-party distributed setting *i.e.,* where a client outsources the distance calculation to two servers, while revealing only the result of the comparison to the service provider. We propose two two-party computation (2PC) protocols of cosine similarity followed by comparison to a public threshold, one in the semi-honest and one in the malicious setting. Our protocols combine additive secret sharing with function secret sharing, saving one communication round by employing a new building block to compute the composition of a bit and a binary function f, thus requiring only two communication rounds under a strong threat model. We evaluate our protocols in the setting of biometric authentication using voice biometrics. Our results show that not only are the proposed protocols efficient, but they also maintain the same accuracy as the plain-text systems.

KEYWORDS

privacy-preservation, malicious security, function secret sharing, cosine similarity

1 INTRODUCTION

Computing distance metrics of sensitive data and comparing to a predefined public threshold in a privacy-preserving way is an indispensable building block for a wide range of applications including privacy-preserving biometric authentication, 1:K biometric identification, privacy-preserving machine learning (*e.g.,* linear regression, matrix multiplication) as well as typo-tolerant (fuzzy) password based authentication. This thresholded distance computation is very challenging to evaluate on sensitive data when multiple untrusted parties are involved in the process.

Although computing distance metrics in a privacy-preserving way has been widely covered in the literature, protecting non-linear computations such as the comparison to a predefined threshold is not easy to achieve in an efficient way. In this paper, we focus on the privacy-preserving computation of *cosine similarity* between two vectors, one of the most commonly employed distance metrics, subsequently comparing the computed result to a public value τ acting as threshold. We propose two novel privacy-preserving protocols to realize this thresholded distance functionality where the computation is outsourced to two computing servers in a two-party computation (2PC) scenario. We leverage recent advances of Function Secret Sharing (FSS) [10, 12] alongside additive secret sharing to achieve strong privacy guarantees for both protocols, since the only information revealed is the result of the comparison (and this only to the service provider). Our protocols are proven secure in the semi-honest setting and in the malicious setting respectively. Crucially, the proposed protocols rely on a new building block named CondEval, which can be employed to compute the composition of an input bit s and a binary function f (evaluated via FSS) *i.e.,* $s \circ f(x)$ for an input value x. This building block is proven secure under both the semi-honest and the malicious models, and may be of independent interest for generic 2PC secure computation.

We apply and evaluate our protocols in the biometric authentication setting with voice biometrics. Biometric authentication consists of veryfing a the identty of a user by comparing a fresh compact representation of the biometric trait with a pre-existing (claimed) reference provided during the enrolment phase. The authentication is granted or denied based on whether or not the obtained similarity score is below a pre-fixed threshold. Although biometrics holds significant promise as a convenient authentication method, privacy concerns have hindered public trust and created obstacles to widespread adoption. For instance, in December 2021, the Flagstar Bank data breach leaked sensitive data of 1.5 million customers and forced the financial institution to pay 5.9 million Dollars in out-of-court settlements[1].

Furthermore, we provide a detailed experimental evaluation of our proposed protocol by employing benchmark speech recognition datasets (*i.e.,* VoxCeleb2). Our results show that the proposed scheme is not only efficient requiring only two communication rounds, but also maintains the same accuracy as the plain-text systems while guaranteeing security under a strong threat model.

[1] https://www.cpomagazine.com/cyber-security/flagstar-bank-data-breach-leaked-sensitive-information-of-1-5-million-customers/

,
.

Table 1: Benchmark of theoretical costs on evaluating a scalar product and comparison to threshold between two vectors of l n-bits integers

Work	Type	#Rounds	Online Blocks	Malicious Server
AriaNN [44]	2PC SS: Arith, FSS	2(1+1)	SS Scalar product, FSS Comparison	N
Boyle et. al [10]	2PC SS: Arith, FSS	2(1+1)	SS Scalar product, FSS IC gate	N
ABY [21]	2PC SS: Boolean&Arith. GC	3(1+2+0)	SS scalar product, Arith. to Yao conversion, GC evaluation	N
ABY2.0 [40]	2PC ΠSS: Boolean&Arith,GC	5(1+1+3)	ΠSS scalar product, Arith. to Boolean conversion, BitExtraction	N
CryptFlow [43]	2PC SS: Arith, OT	5	Linear layer (1-dim weights), dReLU	N
Falcon [53]	3PC Replicated SS: Arith.	8(1+7)	MatMult with 1-dim matrices, Private Compare	Y
Funshade [29]	2PC ΠSS: Arithm., FSS	1	ΠSS scalar product, FSS IC gate	N
Ours	2PC SS: Arith., Boolean, FSS	2	SS scalar product FSS IC gate	Y

Related Work. Several techniques based on advanced cryptographic primitives like Fully Homomorphic Encryption (FHE) [24, 25], secure Multi-Party Computation (MPC) [26, 46, 54] and Functional Encryption (FE) [1, 9], have been proposed in the literature to address the privacy challenges for the computation of some distance measures (such as Hamming distance [35], cosine similarity) followed by comparison with a public threshold. FHE solutions rely on performing an arbitrary number of arithmetic operations (*i.e.,* additions and multiplications) between ciphertexts, without intermediate decryption. MPC-based techniques on the other hand involve distributing data among non-colluding parties to collectively compute a desired function over the private data. MPC protocols cover two main security models: semi-honest or malicious models depending on whether the involved parties follow the protocol or deviate from it. FE approaches are based on a public-key encryption scheme that enables authorized parties to evaluate specific linear functions (*e.g.,* inner products [3, 20, 49]) during ciphertext decryption. Efficient FE-based techniques are restricted to linear function evaluations.

Privacy-preserving comparison: Although linear operations such as scalar products are efficiently addressed by these techniques [28, 42, 47, 51], non-linear operations like comparisons to a public threshold are still a challenge and are often inefficient for real-time applications. Privacy-preserving comparison is possible with FHE setting using computationally intensive polynomial approximations [14, 30]. In contrast, many MPC-based solutions and frameworks realize privacy-preserving comparison in various settings [33, 43, 53]. Mixed protocols (*i.e.,* protocols that combine the use of arithmetic circuits with homomorphic encryption and/or Boolean circuits) compute scalar products using arithmetic protocols and switch from arithmetic to binary circuits in order to compute comparisons [21, 37, 40]. However, these frameworks assume all parties to behave honestly, an often unrealistic assumption for real-world applications. Veugen *et al.* [52] proposed a 2PC framework that improves comparison computation in the SPDZ protocol [17] which is a mixed protocol that takes into account the presence of malicious parties. However, these proposed solutions come with a notable drawback which is the intensive communication cost in terms of communication size and/or number of rounds. Recently, Functional Secret Sharing (FSS)-based protocols have been proposed to efficiently compute the comparison to a public threshold operation [10, 12, 29, 45]. Nevertheless, the mentioned protocols focus mainly on the semi-honest model and barely security against malicious adversaries.

Privacy-preserving biometrics: Multiple initiatives have been proposed in the literature to securely compute similarity distance and comparison to a public threshold for biometric authentication applications. Boddeti [8] presented a privacy-preserving framework for facial recognition based on FHE. The face dissimilarity score is computed in the encrypted domain and consists of performing scalar multiplications and scalar additions. In [38], the authors made use of an FHE scheme to compute the Hamming distance in the encrypted domain for iris authentication. The work proposed in [41] makes use of FHE to privately compute the cosine similarity for an Automatic Speaker Verification (ASV) system. However, the authors focused solely on studying the similarity computation, without carrying out the crucial step of comparing the result to the threshold. They assumed that the decision score would be received and decrypted by the client's device, which could potentially create

a security vulnerability in the event of a malicious client attempting to modify the score to gain unauthorized access. Kim *et al.* [32] presented a fingerprint authentication system using FHE in which fingerprint data is stored and processed in decrypted form. The decision score is computed using the square of Euclidean distance between two encrypted vectors. The threshold comparison is also performed in the encrypted domain. Despite the potential shown by FHE in the previously mentioned work, its utilization in real-world circumstances is still restricted by the significant computational overheads.

Additionally, MPC-based techniques have been successfully used to protect sensitive data like face [4], iris [4, 23], and voice [39, 50]. Barni *et al.* [4] introduced a secure multi-modal biometric authentication, that combines face and iris features, based on secure two-party computation against one malicious party. The two non-colluding parties compute the Hamming distance and the Euclidean distance and later evaluate the comparison of the fused scores with a public threshold. The work in [23] proposes a secure two-party computation solution for an iris verification that is secure in the semi-honest adversary model. The Hamming distance along with the threshold comparison is computed securely.

Nautsch *et al.* [39] provided the first computationally feasible privacy-preserving ASV system with cohort score normalization based on 2PC. However, this work only considers the case where the parties/servers are honest-but-curious, *i.e.,* the parties will not deviate from the described protocol, nevertheless, they can try to collect extra information about the input of the other party. In [50], Treiber *et al.* proposed a 2PC-based ASV system secure against a malicious client device that can change the score to get authenticated by the service provider. However, the 2PC servers are assumed to be honest-but-curious and one party reconstructs the decision.

Our Contributions. In this work, we propose a novel two-party protocol for secure computation of cosine similarity followed by comparison to a predefined threshold. Based on secure 2PC and FSS, we prove two variants of our protocol secure in the semi-honest and malicious settings respectively. Table 1 provides a detailed comparison of our solution with prior work. Contrary to prior work that assumes honest normalisation performed by the client, we require the client to submit a non-normalised reference template and we incorporate a mechanism to check that the secret sharing of the fresh (as well as the reference) template has been secret shared correctly. Falcon is the only work that considers a malicious server. However, it is a symmetric 3PC solution and the maliciousness is introduced thanks to Replicated Secret Sharing (RSS).

The main contributions of our paper are:

- We propose a 2PC framework for privacy-preserving cosine similarity computation and comparison to a pre-defined threshold. We propose two variants: one protocol for the *semi-honest* setting and one for the *malicious* setting. We provide robustness guarantees, since the malicious protocol allows us to control the malicious behaviour of one of the servers. In both protocols, we consider that the normalisation is performed by the servers. Our protocol for the malicious setting relies on authenticated 2PC secret sharing as well as recent advances of FSS.
- We introduce a new building block we call as CondEval that can be employed to compute the composition of an input bit and a binary function f (evaluated via FSS) *i.e.,* $s \circ f(x)$ for an input value x. This building block is provably secure under both the semi-honest and malicious model. This solution is of independent interest and can be employed in a general 2PC secure computation.
- We provide a detailed security analysis of the introduced protocols as well as the new building block CondEval described above.
- We evaluate the proposed protocols experimentally, employing voice biometrics as a use case. Our experiments show that the proposed protocols are very efficient, requiring only two communication rounds while maintaining the same accuracy as the plain-text systems.

The paper is organised as follows. Section 2 describes the problem statement, introducing the application scenario alongside the threat model, and outlining a technical overview of our solution. In Section 3, we revisit the cryptographic 2PC primitives we rely on, namely additive secret sharing and function secret sharing. In Section 4, we introduce CondEval, a building block that allows the composition of an input bit and a binary function f evaluated via FSS, and prove its security in both the semi-honest and malicious settings. Section 5 covers our two novel protocols for the privacy-preserving computation of thresholded cosine similarity for both the semi-honest and the malicious settings as well as their corresponding security analysis. Section 6 presents our experimental evaluation and results, concluding the paper in Section 7.

2 PROBLEM STATEMENT

This section briefly introduces notations, the application scenario and threat model considered as well as the technical overview of our approach to compute cosine similarity and compare to a threshold in a privacy-preserving way.

2.1 Notations

We use the following notations throughout the paper.

ε empty string.
$[n]$ set of integers $\{1, 2, \ldots, n\}$ for a positive integer n.
$s[i]$ i^{th} leftmost bit of a binary string s, where $i \leq |s|$.
$s[i..j]$ substring $\{s[i], s[i+1], \ldots, s[j]\}$ of binary string s.
a_i: i^{th} element of a vector $\boldsymbol{a}$ where $i < |\boldsymbol{a}|$.
$\mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})$ Inner product of two vectors $\boldsymbol{x}, \boldsymbol{y}$.
$\mathsf{Prod}(\boldsymbol{x}, \boldsymbol{y})$ Element-wise product of two vectors $\boldsymbol{x}, \boldsymbol{y}$.
$\mathsf{Shift}(x, \rho, \ell)$ Outputs an encoding in $\mathbb{Z}_{2^\ell}$ of left/right logical shifting $|\rho|$ bits over x respectively if $\rho < 0$ or $\rho > 0$.
$\mathsf{Sign}(x)$ Outputs 1 if $x \geq 0$ and 0 otherwise.
$(M_b, \varepsilon) \leftarrow \mathcal{F}_{\mathsf{OT}}(b, \{M_0, M_1\})$ Given a pair of messages $\{M_0, M_1\}$ from a sender and a choice bit $b \in \{0, 1\}$ from a receiver, it returns M_b to the receiver and ε to the sender.
$\boldsymbol{r} \leftarrow \mathcal{F}_{\mathsf{rand}}(1^\lambda, n, \mathbb{U}_N)$ Given a security parameter λ and a positive integer n, it outputs a random vector $\boldsymbol{r} \in \mathbb{U}_N^n$.
$\mathcal{F}_{\mathsf{Permu}}(\pi, \{a_0, a_1\})$ Given a list $\{a_0, a_1\}$ and a permutation bit $\pi \in \{0, 1\}$, it outputs $\{a_0, a_1\}$ if $\pi = 0$ and $\{a_1, a_0\}$ otherwise.
$[x]^{\mathsf{B}}$ Boolean secret sharing of $x \in \mathbb{Z}_2$.
$[x]^{\mathsf{A}}$ Arithmetic/Additive secret sharing of $x \in \mathbb{U}_N$.

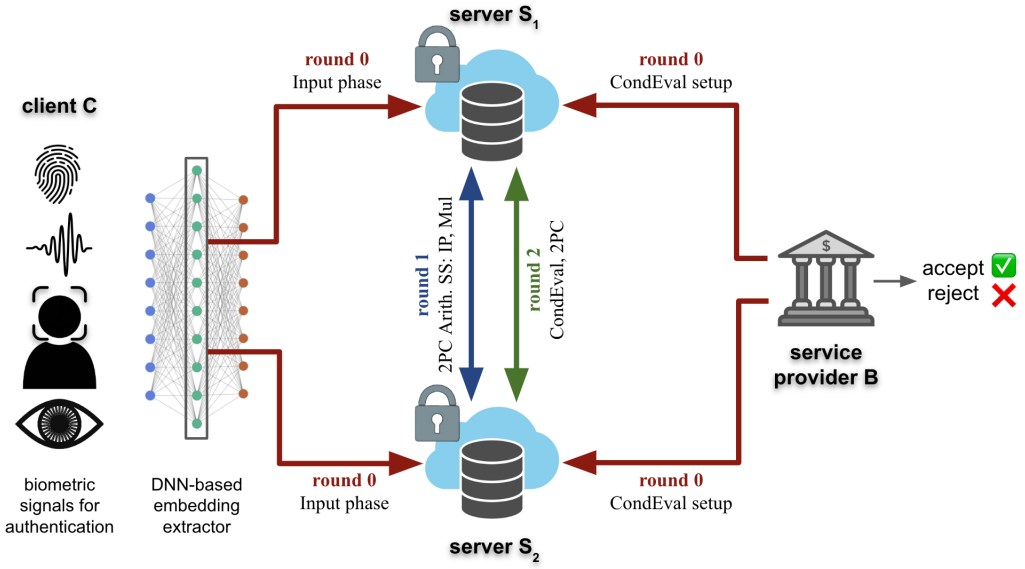

Figure 1: Privacy-preserving biometric authentication system based on cosine similarity in the 2PC setting.

$\langle x \rangle^{\mathsf{A}}$ Optimized secret sharing of x.
$[x]_b^{\mathsf{A}}, [x]_b^{\mathsf{B}}, \langle x \rangle_b^{\mathsf{A}}$ Share of x held by party b.

2.2 Application Scenario

Let us examine a scenario in which a client C requests access to a service, such as an online bank, that uses a biometric-based authentication system. During the authentication process, the client's fresh biometric template is compared to the stored reference biometric template using the cosine similarity distance, and the authentication decision is taken based on a pre-determined threshold. This application scenario is depicted in Figure 1.

For privacy reasons, the client does not want to disclose her biometric data in clear to any party, considering their sensitive nature and the high risk of a data breach in a centralised database. We therefore consider two additional non-colluding servers, S_0 and S_1, in charge of collecting and privately storing the secret shares of the clients' reference templates upon their registration during the enrollment phase, performing the biometric verification process (*i.e.,* matching of the templates) and disclosing the decision output to the service provider B (*i.e.,* the bank).

2.3 Threat Model

In this paper, we assume that the client has already registered and submitted her biometric data (*e.g.,* voice reference template) in a privacy-preserving manner to the two non-colluding servers. Then, in the authentication process the comparison between the fresh biometric template and the registered reference template is performed by the two servers. Note that neither any server nor the service provider have access to any biometric templates in cleartext. We also assume that the protocol is running through secure channels providing security against any external adversary that can compromise the transmission.

In the considered setting, we have four players playing three different roles. We assume a semi-honest dealer (*e.g.,* the central bank B) distributing reliable correlated randomness to two computing servers in the setup phase. We consider that the client may act maliciously *i.e.,* may attempt to impersonate a legitimate user and deduce information for the corresponding templates. Thus, we require the client to submit a non-normalised reference template and we incorporate a mechanism to check that the secret sharing of the fresh (as well as the reference) template have been secret shared correctly. Finally, we consider two non-colluding servers that tolerate one corruption from an active adversary deviating from the correct execution of the protocol (*i.e.,* perform wrong computation in the matching process between the fresh and stored template and/or try to infer information about the fresh and/or stored biometric templates). We highlight that if both servers deviate from the protocol, they don't obtain any useful information related to the client's data as long as they don't collude.

2.4 Privacy-preserving cosine similarity - Technical Overview

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ of dimension n denote the fresh template received from the user requesting to authenticate and the reference template submitted upon enrollment, respectively. The cosine similarity metric would then computed as:

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|} = \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{1}$$

To authenticate the owner of $\boldsymbol{x}$ against the reference $\boldsymbol{y}$, the resulting score of $cos(\boldsymbol{x}, \boldsymbol{y})$ would then compared with a public threshold $\tau \in [-1, 1]$. If the similarity score is greater than or equal to τ, then the user is successfully authenticated, otherwise, the

authentication fails. Overall, we can define the biometric authentication functionality as:

$$\mathcal{F}_{\mathsf{CosAuth}}(\boldsymbol{x}, \boldsymbol{y}, \tau) = \mathsf{Sign}(cos(\boldsymbol{x}, \boldsymbol{y}) - \tau) \tag{2}$$

In order to design a privacy-preserving protocol for this functionality, we must define privacy-preserving protocols to realize all to the underlying operations: addition, multiplication, inner product, division, square-root and sign extraction. Arithmetic operators can be straightforwardly instantiated using additive secret sharing [5]. In contrast, the square root is relatively complex to realize, thus we choose instead to remove it by squaring the inputs to the Sign function with additional terms to maintain the original signs. The computation of $\mathcal{F}_{\mathsf{CosAuth}} = c$ would proceed as follows:

- If $\tau > 0$, we compute $c = c_1 \wedge c_2$ where
$$\begin{aligned} c_1 &= \mathsf{Sign}([\mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})]^{\mathsf{A}}) \\ c_2 &= \mathsf{Sign}([1/\tau^2 \cdot \mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})^2 - \mathsf{IP}(\boldsymbol{x}, \boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y}, \boldsymbol{y})]^{\mathsf{A}}) \end{aligned} \tag{3}$$
- If $\tau = 0$, we compute:
$$c = \mathsf{Sign}(\mathsf{IP}(\boldsymbol{x}, \boldsymbol{y}))$$
- Otherwise if $\tau < 0$, we compute $c = c_1 \vee c_2$ where
$$\begin{aligned} c_1 &= \mathsf{Sign}([\mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})]^{\mathsf{A}}) \\ c_2 &= \mathsf{Sign}([\mathsf{IP}(\boldsymbol{x}, \boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y}, \boldsymbol{y}) - 1/\tau^2 \cdot \mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})^2]^{\mathsf{A}}) \end{aligned}$$

One can easily verify the correctness of this definition. For the remnant of this paper, *w.l.o.g.,* we assume $\tau > 0$, and define the computation pipeline defined in Equation 3 as a circuit C. When naively implemented with a traditional 2PC protocols and FSS, this circuit requires three communication rounds: one round to compute c_1, one for c_2 and one to compute $c = c_1 \wedge c_2$.

We enhance the naïve solution by decreasing the number of communication rounds to 2. To achieve this, we propose a new 2PC protocol named CondEval that combines 2PC authenticated secret sharing and recent advances of FSS in order to compute the composition of a Sign function with an $\wedge$ or $\vee$ operation. Leveraging new building block, the computation of $c = c_1 \wedge c_2$ can be performed at the same time as the computation of c_2 performed through FSS, saving up a communication round.

We prove the security of this new CondEval block, as well as that of the complete privacy-preserving protocol realizing $\mathcal{F}_{\mathsf{CosAuth}}$, under the semi-honest and the malicious models admitting one corruption out of the two computing servers. For maliciously-secure variants of our protocols we employ homomorphic MACs when using additive secret shares (as in the SPDZ protocol [17]). Additionally, to provide active security in the FSS gates we resort to the parallel execution of multiple instances of these gates with independent FSS keys. A subset of these instances, chosen at FSS key generation and unknown to the computing servers, act as "trap" instances, allowing the detection of malicious behavior in a single computing server.

The full maliciously-secure protocol, combining arithmetic secret sharing with FSS, consists of the following four stages:

(1) *Setup*: Generation of relevant correlated randomness;
(2) *Input*: Both the client and B input their secret vectors $\boldsymbol{x}, \boldsymbol{y}$ separately;
(3) *Evaluation*: The two servers:
   - *First round*: Compute a Boolean sharing $[c_1]^{\mathsf{B}}$;
   - *Second round*: Compute the final Boolean sharing $[c]^{\mathsf{B}} = [c_1]^{\mathsf{B}} \circ f(x)$ (or $(c)^m$ along with a proof $[\sigma]^{\mathsf{A}}$ for the computation in the malicious setting);
(4) *Reveal*: The service provider B builds c in cleartext (or verifies $(c)^m$ and check $[\sigma]^{\mathsf{A}}$ in the malicious setting).

3 PRELIMINARIES

In this section, we describe notions used in the rest of the paper *i.e.,* two party secure computations, secure truncation and function secret sharing.

3.1 Two-Party Secure Computation

Secure two-party computation allows two non-colluding parties P_0 and P_1 to compute a function f on their private inputs (*e.g.,* x and y) without revealing any information beyond the final value of $f(x, y)$. 2PC protocols are based on either: *(i)* Secret Sharing (SS) techniques *i.e.,* arithmetic SS, like additive SS [5] and replicated SS [2], or Boolean SS like GMW [36]; or *(ii)* Garbled Circuits (CG) like Yao's GCs [55], BMR [6].

Optimized Additive SS. In this work, we make use of an optimized additive SS with function dependent pre-processing as our building block [7]. In the setup phase, correlated random offset shares are generated and computed for input wires and output wires of each gate to the function circuit to be evaluated. These correlated random shares are input-independent, *i.e.,* they do not depend on the private input of the parties and can be executed at any time before the online/data-dependent phase where the actual function f is evaluated. In the evaluation phase, efficient online secure computation is performed. We denote by $\langle x \rangle^{\mathsf{A}}$ the optimized additive SS of a secret x in a two-party setting of $\mathsf{S}_0, \mathsf{S}_1$, where $\forall b \in \{0, 1\}$, S_b holds partial shares

$$\langle x \rangle_b^{\mathsf{A}} : (\delta_x, [r_x]_b^{\mathsf{A}})$$

in which $\delta_x = x + r_x$, $[r_x]^{\mathsf{A}}$ is the associated random offset of the secret value x. One share alone does not disclose any information about x but when summed together, they reconstruct it ($x = \delta_x - [r_x]_0^{\mathsf{A}} - [r_x]_1^{\mathsf{A}}$). The parties interact with each other to compute any desired function from the input secret shares. We denote by $\mathcal{F}_{\mathsf{Reveal}}([x]^{\mathsf{A}})$ the functionality that inputs an arithmetic secret sharing of x from two servers and outputs x. Given $\langle x \rangle^{\mathsf{A}}$ and $\langle y \rangle^{\mathsf{A}}$, we demonstrate how addition and multiplication gates are performed in the following.

Addition. It requires no communication between the two parties. The two parties simply locally add the shares they hold. More precisely, each party $b \in \{0, 1\}$ computes:

$$\langle x + y \rangle_b^{\mathsf{A}} = (\delta_x + \delta_y, [r_x]_b^{\mathsf{A}} + [r_y]_b^{\mathsf{A}})$$

Multiplication. It requires interaction between the two parties and relies on additional input-independent but function-dependent random pre-computed secret shares named Beaver's triples [6]. To compute $[z]^{\mathsf{A}}$ where $z = xy$, with the corresponding Beaver's triples $[r_x]^{\mathsf{A}}$,$[r_x]^{\mathsf{A}}$, $[r_x r_y]^{\mathsf{A}}$ generated in the setup phase, the parties are able to locally compute the additive secret shares of z. *i.e.,* $\forall b \in \{0, 1\}$, S_b holds $\{\langle x \rangle_b^{\mathsf{A}} = (\delta_x, [r_x]_b^{\mathsf{A}}), \langle y \rangle_b^{\mathsf{A}} = (\delta_y, [r_y]_b^{\mathsf{A}}), [r_x r_y]_b^{\mathsf{A}}\}$

FUNCTIONALITY $[\varsigma]^{\mathsf{A}} \leftarrow \mathcal{F}_{\mathsf{MacVryGen}}([\Delta]^{\mathsf{A}}, L)$

Players: $\mathsf{S}_0, \mathsf{S}_1$.

Input: An arithmetic secret sharing of the MAC authentication key $[\Delta]^{\mathsf{A}}$, along with a list $L = (x_i, [\Delta \cdot x_i]^{\mathsf{A}}, r_i)_{i\in[n]}$, in which $\forall i \in [n]$ that x_i, r_i is public and $[\Delta \cdot x_i]^{\mathsf{A}}$ is the arithmetic secret sharing over $\Delta \cdot x_i$.

Output: $[\varsigma]^{\mathsf{A}}$.

1: **for** $b = 0$ to 1 **do**
2: $\quad [\varsigma]_b^{\mathsf{A}} \leftarrow 0$
3: $\quad$ **for** $i = 1$ to n **do**
4: $\quad\quad [\varsigma]_b^{\mathsf{A}} \leftarrow [\varsigma]_b^{\mathsf{A}} + r_i \cdot ([\Delta]_b^{\mathsf{A}} \cdot x_i - [\Delta \cdot x_i]_b^{\mathsf{A}})$
5: Ouput $[\varsigma]^{\mathsf{A}}$.

Figure 2: Homomorphic MAC Verification proof generation in the malicious setting.

in which $\delta_x = x + r_x, \delta_y = y + r_y$. Then, S_b computes:

$$[z]_b^{\mathsf{A}} = b \cdot \delta_x\delta_y - \delta_x[r_x]_b^{\mathsf{A}} - \delta_y[r_y]_b^{\mathsf{A}} + [r_xr_y]_b^{\mathsf{A}}. \tag{4}$$

However, to conform the multiplication output to an optimized secret sharing format, the parties locally compute $[z + r_z]^{\mathsf{A}}$ and reveal $z+r_z$ in one round, where r_z is a pre-generated random output wire offset in the setup phase. Thus, obtaining $\langle z\rangle^{\mathsf{A}} = (z+r_z, [r_z]^{\mathsf{A}})$.

Throughout this paper, we denote $[x]^{\mathsf{A}} \leftarrow \mathsf{SS.Share}(x, \mathbb{Z}_{2^\ell})$ as the algorithm dividing a secret into secret sharing in the domain $\mathbb{Z}_{2^\ell}$, where x can be a single value or an n-sized vector; $\mathsf{SS.MUL}(\langle x\rangle^{\mathsf{A}}, \langle y\rangle^{\mathsf{A}}, [r_xr_y]^{\mathsf{A}})$ as the multiplication process shown in Eq. 4, $\mathsf{SS.IP}(\langle \boldsymbol{x}\rangle^{\mathsf{A}}, \langle \boldsymbol{y}\rangle^{\mathsf{A}}, [\boldsymbol{u}]^{\mathsf{A}})$ as the inner product of two vectors $\boldsymbol{x}, \boldsymbol{y}$ where $\boldsymbol{u}$ is the correlated product of randomness.

SPDZ2k protocol. The SPDZ protocol [16] is a MPC protocol works in the context of the dishonest majority setting, or assuming one malicious server in the 2PC setting. It performes secure operations over authenticated secret sharing in a ring, and manages to detect any cheating behavior by introducing a message authentication code Δ secretly shared between the two servers. Throught out this paper where we deal with 2PC setting, we denote an authenticated arithemetic secret sharing of a secret x as

$$[\![x]\!]^{\mathsf{A}} : ([x]^{\mathsf{A}}, [\Delta \cdot x]^{\mathsf{A}})$$

and $\langle\!\langle x\rangle\!\rangle^{\mathsf{A}}$ as the authenticated optimized arithmetic secret sharing of x. Naturally, we use notations $[\![x]\!]_b^{\mathsf{A}}$ and $\langle\!\langle x\rangle\!\rangle_b^{\mathsf{A}}$ to denote the share held by S_b. After the online secure computation, two servers work together to verify the integrity of every revealed value in the secure computation. This is done by using a final MAC verification algorithm $\mathcal{F}_{\mathsf{MacVryGen}}([\Delta]^{\mathsf{A}}, L)$ (as shown in Fig. 2), in which Δ denotes the secret authentication key, $L = (x_i, [\Delta \cdot x_i]^{\mathsf{A}}, r_i)_{i\in[n]}$ is the list of all revealed values, their associated authenticated share, and random numbers negotiated between two servers. If the verification fails, malicious behaviors are detected and the parties abort the procedure.

When working with authenticated optimized secret sharing in the malicious setting, we use same notation $\mathsf{SS.MUL}(\langle\!\langle x\rangle\!\rangle^{\mathsf{A}}, \langle\!\langle y\rangle\!\rangle^{\mathsf{A}}, [\![r_xr_y]\!]^{\mathsf{A}})$ and $\mathsf{SS.IP}(\langle\!\langle \boldsymbol{x}\rangle\!\rangle^{\mathsf{A}}, \langle\!\langle \boldsymbol{y}\rangle\!\rangle^{\mathsf{A}}, [\![\boldsymbol{u}]\!]^{\mathsf{A}})$ to denote secure multiplication and secure inner product, respectively.

3.2 Secure Truncation

Given an arithmetical secret sharing $[x]^{\mathsf{A}}$ where $x \in \mathbb{Z}_\ell$ and a truncation parameter $\rho \in \mathbb{Z}^+$, a secret sharing based truncation operation aims to compute the truncated secret sharing $[\mathsf{Shift}(x, \rho, \ell)]^{\mathsf{A}}$. In the proposed protocols of section 5, we use a secure truncation operation over random secret sharing results in the pre-processing model. We employ the truncation pair $(r, \mathsf{Shift}(r, \rho, \ell))$ introduced in [37]. More precisely, in our model, we generate a truncation pair $(r, \mathsf{Shift}(r, \rho, \ell))$ as follows:

(1) In the setup phase, a random offset $r \in \mathbb{Z}_{2^\ell}$ is generated, both $[r]^{\mathsf{A}}$ and $[\mathsf{Shift}(r, \rho, \ell)]^{\mathsf{A}}$ are distributed among S_0 and S_1;
(2) In the evaluation phase, to truncate a secret sharing $[x]^{\mathsf{A}}$ where $x \in \mathbb{Z}_{2^\ell}$, the servers run $\mathcal{F}_{\mathsf{reveal}}([x + r]^{\mathsf{A}})$ to obtain $x + r$, $\forall b \in \{0, 1\}$ then S_b computes $\mathsf{Shift}(x + r, \rho, \ell) - [\mathsf{Shift}(r, \rho, \ell)]_b^{\mathsf{A}}$ as $[\mathsf{Shift}(x, \rho, \ell)]_b^{\mathsf{A}}$.

3.3 Function Secret Sharing

Function Secret Sharing (FSS) was first introduced by Boyle *et al.* [11] as a cryptographic primitive that secretly shares a function f. Informally, in a two party setting with a function $f : \mathbb{D} \to \mathbb{R}$, *function secret sharing* comprises a key generation algorithm $\mathsf{Gen}^f(1^\lambda)$ producing a key pair (k_0, k_1), and an evaluation algorithm Eval^f, which takes in one shared key and a value x, ensuring that $\forall x \in \mathbb{D}$ that the sum of $\mathsf{Eval}^f(k_0, 0, x)$ and $\mathsf{Eval}^f(k_1, 1, x)$ is equal to $f(x)$.

Shortly after, a secure computation scheme with pre-processing via FSS was proposed by Boyle *et al.* [12]. In this scheme, nonlinear gates (*e.g.*, for equality tests, integer comparison, etc.) can be computed with a relatively small amount of communication in one round. We give an example how in this model a secure equality check is conducted between S_0 and S_1.

Let us consider the case of a point function $f_a : \{0, 1\}^\ell \to \{0, 1\}$ corresponding to a special point $a \in \{0, 1\}^\ell$, $\forall x \in \{0, 1\}^\ell$ that outputs 1 if $x = a$ and outputs 0 otherwise. In the pre-processing phase, a trustworthy dealer respectively distributes $\mathsf{S}_0, \mathsf{S}_1$ additive secret shares r_0, r_1 of a random mask value $r \leftarrow \{0, 1\}^\ell$ and FSS key shares k_0, k_1 correspond to the random point function f_r. In the online phase, $\forall i \in \{0, 1\}$, P_i holds x_i, r_i and k_i, and P_i exchanges $x_i + r_i$ with P_{1-i}. After revealing the masked value $x + r$, $\forall i \in \{0, 1\}$, P_i computes $f_{r,i}(x + r - a)$ and thus, obtains secret shares of $f_a(x)$. Boyle *et al.* also proposed FSS schemes for comparison functions in [11, 13].

In constructing all our protocols, we make use the interval containment function secret sharing (IC-FSS) ([10], Section 4.1) as our secure comparison primitive.

Definition 1. *Interval containment function secret sharing.* There is a key generation algorithm $\mathsf{Gen}_\ell^{[p,q]}(\cdot)$, and an evaluation algorithm $\mathsf{Eval}_\ell^{[p,q]}(\cdot)$, given an interval containment $[p, q]$ where $p, q \in \mathbb{Z}_{2^\ell}$ and $p < q$, $\forall \alpha \in \mathbb{Z}_{2^\ell}$, $\forall \beta_1, \beta_2 \in \mathbb{U}_N$,

$$(k_0, k_1) \leftarrow \mathsf{Gen}_\ell^{[p,q]}(1^\lambda, \alpha, \beta_1, \beta_2, \mathbb{U}_N)$$

for $\forall x \in \mathbb{Z}_{2^\ell}$ denote $\delta_x = x + \alpha$, it holds that:

$$\sum_{b=0}^{1} \mathsf{Eval}_\ell^{[p,q]}(k_b, b, \delta_x) = \begin{cases} \beta_1, & \text{if } x \in [p, q] \\ \beta_2, & \text{if } x \text{ not in } [p, q] \end{cases}$$

4 REALIZING CONDEVAL IN 2PC

In order to design our privacy-preserving biometric matching solution in an optimized manner, we propose a new building block that helps us in reducing the number of communication rounds. Indeed, as explained in Section 2.4, the evaluation of a biometric matching protocol consists of the following steps: *(i)* computing shares of the Boolean values c_1 and c_2 (see Equation 3) and then, depending on the value of τ, *(ii)* computing $c_1 \wedge c_2$ or $c_1 \vee c_2$. Since c_1 and c_2 are the output of the Sign operation, FSS can be used efficiently. On the other hand, because computing a Boolean gate ($\wedge$ or $\vee$) in secret sharing requires at least one communication round, we propose a new building block CondEval that integrates this operation within the FSS computation of c_2 and hence, reduces the number of communication rounds to one.

It is worth to note that such a building block can be used as an independent scheme, whenever there is a need to compute the Boolean composition of a certain bit s with a certain function f evaluated with FSS with an input x:

$$[s \circ f(x)]^{\mathsf{B}} \leftarrow \mathsf{CondEval}(\circ, \mathcal{K}_{\circ}, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$$

In the context of biometric matching s corresponds to the already computed c_1 and $f(x)$ corresponds to the evaluation of c_2, namely, $\mathsf{Sign}([1/\tau^2 \cdot \mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})^2 - \mathsf{IP}(\boldsymbol{x}, \boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y}, \boldsymbol{y})]^{\mathsf{A}})$.

Thanks to this newly proposed solution, CondEval is realized in only one communication round (instead of two) among two parties in the pre-processing model. In this Section, we consider only the $\wedge$ operation and we describe how CondEval computes $s \wedge f(x)$ in a symmetric two-party setting. Two versions of CondEval are described: one in the semi-honest model and another one in the malicious model in which one of the servers does not follow the protocol correctly.

4.1 High-level Overview

As previously mentioned the goal of CondEval is to evaluate $s \circ f(x)$ using FSS in a 2PC setting where two computing servers, S_0 and S_1, engage in symmetric computation and communication. The input to this protocol is mainly secret shares of a bit s and an input x. In the sequel of this paper, we consider the $\wedge$ operation only and details on the $\vee$ operation are provided in Appendix C. In order to avoid the additional communication round that is due to the $\wedge$ operation, we propose to integrate this particular operation through the FSS evaluation of x. Indeed, from Table 2 (which corresponds to the truth table of $s \wedge f(x)$), we observe that if $s = 0$, then the output is 0 and if $s = 1$ then the output is the actual output of $f(x)$. In the case when the output range is in $\mathbb{Z}_2$, $f(x)$ is obtained when each server $\mathsf{S}_b (b \in \{0, 1\})$ runs $\mathsf{Eval}^f(k_b, x)$ and the output is reconstructed with an XOR of these. If each server runs Eval^f over the same keys, the reconstructed output would become 0, which corresponds to the case when $s = 0$. Hence, the overall idea is to make sure that when running Eval, the two servers use the same FSS key if $s = 0$ and different FSS keys if $s = 1$. This is illustrated in the 2PC setting (note that s is secretly shared among the two servers as well) in Table 2. The goal is therefore to make sure that each server retrieves the correct FSS key according to s, without leaking any information neither about s, nor the other FSS key.

s	$s \wedge f(x)$	s_0	s_1	$[y]_0^{\mathsf{B}}$	$[y]_1^{\mathsf{B}}$	$[y]_0^{\mathsf{B}} \oplus [y]_1^{\mathsf{B}}$
0	0	0	0	$\mathsf{Eval}^f(k_0, 0, \delta_x)$	$\mathsf{Eval}^f(k_0, 0, \delta_x)$	0
		1	1	$\mathsf{Eval}^f(k_1, 1, \delta_x)$	$\mathsf{Eval}^f(k_1, 1, \delta_x)$	
1	$f(x)$	1	0	$\mathsf{Eval}^f(k_0, 0, \delta_x)$	$\mathsf{Eval}^f(k_1, 1, \delta_x)$	$f(x)$
		0	1	$\mathsf{Eval}^f(k_1, 1, \delta_x)$	$\mathsf{Eval}^f(k_0, 0, \delta_x)$	

Table 2: The truth table of $[s \wedge f(x)]^{\mathsf{B}}$.

Such a protocol could be easily designed if the trusted bank, would store these two FSS keys k_0 and k_1 and randomly map them to the values of s (k_t if $s = 0$, k_{1-t} is $s = 1$, where $t \leftarrow\!\!\$ \{0, 1\}$). Then, each server would run a private information retrieval protocol with the bank to retrieve the actual key corresponding to its share without revealing the actual share. In order not to involve an additional party on the computation and save in the number of communication rounds, we propose to store these two keys at both servers in an encrypted manner. More precisely, server S_0 will store the two FSS keys, encrypted beforehand and will receive one decryption key from server S_1 based on its share of s and vice versa. Hence, the key required to decrypt the FSS key (stored on one server) is stored on the other server.

This implies that, each server S_b will store both FSS keys in an encrypted manner but will only be able to decrypt one of them according to the value of s. The goal is to receive and decrypt the correct FSS key, without disclosing s nor the FSS key not used by S_b.

To prevent any leakage about $[s]_b^{\mathsf{B}}$ from S_{1-b}, and the other FSS key not used by S_{1-b} where $b \in \{0, 1\}$, CondEval integrates the following measures:

- since Eval^f takes as input not only one FSS key but also this FSS key's corresponding index (0 or 1), this index information should not leak any information about s. Hence, the mapping between the FSS key index and $[s]_b^{\mathsf{B}}$ needs to be protected as well, this is realized by set the initial FSS key pair as (k_t, k_{1-t}) where $t \leftarrow\!\!\$ \{0, 1\}$.
- to protect the value of $[s]_b^{\mathsf{B}}$, a random permutation is applied to (k_t, k_{1-t});
- to protect one of the FSS key not used, the permuted key pair are encrypted through one-time-pad before their storage and only one decryption key will be sent from S_b to S_{1-b} in the key decryption step.

In a highlevel overview, CondEval consists of four algorithms executed in two phases:

- During the **setup phase**, a trusted dealer generates the keying material using CondEval.KeyGen and protects the FSS key pair with CondEval.KeyEnc. These encrypted FSS keys and the keying material are distributed to the servers;
- During the **online phase**, each server who has received the share of s and the randomized input $\delta_x = x + r$, will send the correct decryption key sk according to its share of s to the other server. Once the correct FSS key is decrypted using CondEval.KeyDec, the server finishes the protocol by executing the FSS Eval function (CondEval.Eval) to obtain the share of the output $s \wedge f(x)$.

KeyGen	$r \leftarrow_\$ \mathbb{Z}_{2^\ell}, [r]^{\mathsf{A}} \leftarrow \mathsf{SS.Share}(r, \mathbb{Z}_{2^\ell})$ $(k_0, k_1) \leftarrow \mathsf{Gen}^f(1^\lambda, r, 1, 0, \mathbb{Z}_2)(\lvert k_0\rvert = \lvert k_1\rvert = K);$ $t \leftarrow_\$ \mathbb{Z}_2, \pi_0 \leftarrow_\$ \mathbb{Z}_2, \pi_1 \leftarrow_\$ \mathbb{Z}_2; L := K + 1;$ $\mathsf{sk}_0^{(0)} \| \mathsf{sk}_1^{(0)} \| \mathsf{sk}_0^{(1)} \| \mathsf{sk}_1^{(1)} \leftarrow_\$ \{0,1\}^{4L},$ $\lvert \mathsf{sk}_0^{(0)}\rvert = \lvert \mathsf{sk}_1^{(0)}\rvert = \lvert \mathsf{sk}_0^{(1)}\rvert = \lvert \mathsf{sk}_1^{(1)}\rvert = L.$
KeyEnc	$m_0 = \{\mathsf{sk}_0^{(0)} \oplus (k_t \| t), \mathsf{sk}_1^{(0)} \oplus (k_{1-t} \| (1-t))\}$ $C_0 = \mathcal{F}_{\mathsf{Permu}}(\pi_0, m_0); \mathsf{SK}_0 = \{(\mathsf{sk}_0^{(1)}, \mathsf{sk}_1^{(1)}), \pi_1\}$ $m_1 = \{\mathsf{sk}_0^{(1)} \oplus (k_t \| t), \mathsf{sk}_1^{(1)} \oplus (k_{1-t} \| (1-t))\}$ $C_1 = \mathcal{F}_{\mathsf{Permu}}(\pi_1, m_1); \mathsf{SK}_1 = \{(\mathsf{sk}_0^{(0)}, \mathsf{sk}_1^{(0)}), \pi_0\}$
Outputs $\mathcal{K}_\wedge = \{C_i, \mathsf{SK}_i, [r]_i^{\mathsf{A}}\}_{i\in\{0,1\}}$	

Table 3: The construction of $\mathcal{K}_\wedge \leftarrow \mathsf{CondEval.Setup}(\wedge, 1^\lambda, \ell)$ in the semi-honest setting, where λ is the security parameter used in generating FSS keys, ℓ defines the domain of the secret sharing.

Functionality $[s \wedge f(x)]^{\mathsf{B}} \leftarrow \mathsf{CondEval.Eval}(\wedge, \mathcal{K}_\wedge, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$

Players: $\mathsf{S}_0, \mathsf{S}_1$.
Functionality: $[s \wedge f(x)]^{\mathsf{B}} \leftarrow \mathsf{CondEval.Eval}(\wedge, \mathcal{K}_\wedge, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$.
Input: $[s]^{\mathsf{B}}, [x]^{\mathsf{A}}$ from two servers, and $\mathcal{K}_\wedge$ prepared in the setup phase as shown in Table 3 where $\mathcal{K}_\wedge = \{C_b, \mathsf{SK}_b, r_b\}_{b\in\{0,1\}}$, more specifically each $\mathsf{S}_b (b \in \{0,1\})$ inputs $\{C_b, \mathsf{SK}_b, r_b\}$. (Note that (r_0, r_1) constitutes $[r]^{\mathsf{A}}$).
Output: $[s \wedge f(x)]^{\mathsf{B}}$.

1: $\delta_x \leftarrow \mathcal{F}_{\mathsf{Reveal}}([x]^{\mathsf{A}} + [r]^{\mathsf{A}})$
2: **for** $b = 0$ to 1 **do**
3: $\quad \mathsf{S}_b$: $p^{(1-b)} \leftarrow [s]_b^{\mathsf{B}} \oplus \pi_{1-b}, \mathsf{sk}^{(1-b)} \leftarrow \mathsf{sk}^{(1-b)}_{[s]_b^{\mathsf{B}}}$.
4: $\quad \mathsf{S}_b$: Sends $p^{(1-b)} \| \mathsf{sk}^{(1-b)}$ to S_{1-b}. ▷ Key decryption
5: **for** $b = 0$ to 1 **do**
6: $\quad \mathsf{S}_b$: $k^{(b)} \| \mathsf{id}^{(b)} \leftarrow C_b[p^{(b)}] \oplus \mathsf{sk}^{(b)}$
7: $\quad \mathsf{S}_b$: $[y]_b^{\mathsf{B}} \leftarrow \mathsf{Eval}^f(k^{(b)}, \mathsf{id}^{(b)}, \delta_x)$
8: Outputs $[y]^{\mathsf{B}}$.

In the following, we describe CondEval in both the semi-honest and malicious settings. Specifically, we consider the cases of computing $s \wedge f(x)$ and $s \vee f(x)$. We only demonstrate the construction of CondEval for the case of computing $s \wedge f(x)$ in both the semi-honest and malicious settings; The protocol for computing $s \vee f(x)$ is defined similarly and can be performed either by using:

$$s \vee f(x) = \neg(\neg s \wedge \neg f(x))$$

or directly by relying on a slightly different key preparation and evaluation compared with computing $s \wedge f(x)$. The second approach due to space constraints, we have attached in the Appendix C.

4.2 CondEval in the Semi-Honest setting

We propose the building block CondEval which can be used to evaluate $s \circ f(x)$. In this section, we introduce the CondEval construction that is secure in the semi-honest setting and focus on the $\wedge$ operation only. It works in the pre-processing model, and is composed of two functionalities:

$$\mathcal{K}_\wedge \leftarrow \mathsf{CondEval.Setup}(\wedge, 1^\lambda, \ell) \text{ and}$$
$$[y]^{\mathsf{B}} \leftarrow \mathsf{CondEval.Eval}(\wedge, \mathcal{K}_\wedge, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$$

that are run respectively in the setup and online phase. The construction of CondEval.Setup is shown in Table 3. CondEval.Setup, inputs a security parameter 1^λ and outputs $\mathcal{K}_\wedge$. After *KeyGen* and *KeyEnc*, it outputs correlated random keys which are then distributed to the two servers. $\mathsf{CondEval.Eval}(\wedge, \mathcal{K}_\wedge, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$ inputs $\mathcal{K}_\wedge$ generated by CondEval.Setup as well as $[s]^{\mathsf{B}}, [x]^{\mathsf{A}}$. Then, within one round of communication, δ_x is revealed and one cleartext FSS key is obtained from both servers with *KeyDec* being performed, and finally it outputs $[y]^{\mathsf{B}}$.

Theorem 1. *Correctness.* If two servers follow $\mathsf{CondEval.Eval}(\wedge, \mathcal{K}_\wedge, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$ honestly, then it outputs $[y]^{\mathsf{B}} = [s \wedge f(x)]^{\mathsf{B}}$.

Proof. If $s = 0$, then $([s]_0^{\mathsf{B}}, [s]_1^{\mathsf{B}})$ is equal to either $(0,0)$ or $(1,1)$. Thus, in the online phase after the decryption of FSS keys, the servers obtain either $\{k_t, k_t\}$ or $\{k_{1-t}, k_{1-t}\}$, which implies that:

$$y = \begin{cases} \mathsf{Eval}^f(k_0, 0, \delta_x) \oplus \mathsf{Eval}^f(k_0, 0, \delta_x) \text{ or} \\ \mathsf{Eval}^f(k_1, 1, \delta_x) \oplus \mathsf{Eval}^f(k_1, 1, \delta_x). \end{cases}$$

In either case, $y = 0$, as desired. On the other hand, if $s = 1$, then $([s]_0^{\mathsf{B}}, [s]_1^{\mathsf{B}})$ equals either $(0,1)$ or $\{1,0\}$, which implies that $y = \oplus_{t=0}^{1} \mathsf{Eval}^f(k_t, t, \delta_x)$. In either case, we obtain $y = f(x)$, as desired. Thus, $y = s \wedge f(x)$. □

Theorem 2. *Security.* In the presence of a passive PPT adversary $\mathcal{A}$ corrupting one of the two servers in $\mathsf{CondEval.Eval}(\wedge, \mathcal{K}_\wedge, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$, we assert that $\mathcal{A}$ learns nothing about the inputs x, s, nor about the output $s \wedge f(x)$.

Proof. After online evaluation, for each $b \in \{0,1\}$ we denote S_b's transcript view as:

$$\mathtt{View}_b := \{[r]_b^{\mathsf{A}}, \delta_x, k_{t\oplus[s]_{1-b}^{\mathsf{B}}} \| (t \oplus [s]_{1-b}^{\mathsf{B}}), [s]_{1-b}^{\mathsf{B}} \oplus \pi_b\}$$

From this transcript, $\{k_{t\oplus[s]_{1-b}^{\mathsf{B}}}, [r]_b^{\mathsf{A}}, \delta_x\}$ correspond to the FSS evaluation in the pre-processing model, and we resort to the security proof in [12] (Definition 2) of Boyle *et al.* to argue the computational indistinguishability of the ideal and real-world executions; Regarding the remaining items of the view transcripts, since both π_b and t are uniformly selected, both $[s]_{1-b}^{\mathsf{B}} \oplus \pi b$ and $t \oplus [s]_{1-b}^{\mathsf{B}}$ provide information-theoretic secrecy for $[s]_{1-b}^{\mathsf{B}}$. Therefore, from $\mathtt{View}_b$, $\mathcal{A}$ learns nothing about the input x, s, nor the output $s \wedge f(x)$. □

4.3 CondEval in the Malicious Setting

We extend the setting and consider one of the servers be an active adversary. Since the two servers are symmetric (and execute the same protocol), for the sake of clarity, we consider that S_0 is malicious. Assume we run $\mathsf{CondEval.Eval}(\wedge, \mathcal{K}_\wedge, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$ in the malicious setting where S_0 is malicious and $\mathcal{K}_\wedge$ is the output of $\mathcal{K}_\wedge \leftarrow \mathsf{CondEval.Setup}(\wedge, 1^\lambda, \mathbb{Z}_{2^\ell})$, then we report three disruptions from S_0 that might compromise the correctness or security of the scheme:

- S_0 may dishonestly report $[x + r]_0^{\mathsf{A}}$ when revealing $x + r$ to introduce errors and thus disrupt the computation;

- S_0 may flip $[s]_0^B$ during the key decryption step, thus potentially flipping $[y]^B$ to be equal to $[\neg(s \wedge f(x))]^B$;
- After the online evaluation, S_0 may submit $\neg[y]_0^B$ to B, thus potentially resulting in a flipped authentication bit $y = \neg(s \wedge f(x))$.

To counter the first attack, we incorporate a homomorphic MAC scheme from the spdz2k framework [16]. More specifically, during the setup phase, a trustworthy dealer generates random offset shares and the corresponding authenticated shares. This enables an additional verification step to be performed over all partially disclosed intermediate values at the end of the online evaluation phase. The final output is only deemed valid if the verification is successful, thereby deterring any fraudulent disclosure of secret sharing. In order to safeguard against the remaining attacks, we require a separate scheme that protects Boolean secret sharings. As a solution, we propose the use of *authenticated Boolean secret sharing* defined as below.

Definition 2. An *authenticated Boolean secret sharing* $(v)^m$ of bit $v \in \{0,1\}$ is a list of Boolean secret shares defined with a secret authentication key $\psi \in \{0,1\}^m, m \in \mathbb{Z}^+$, denoted as:

$$(v)^m : \{[v_1]^B, \cdots, [v_m]^B\}$$

where $\forall i \in [m]$,

$$v_i = \begin{cases} v, & \text{if } \psi[i] = 0 \\ 0, & \text{if } \psi[i] = 1 \end{cases}$$

By extending the semi-honest CondEval constructions with the above two authentication schemes, we realize CondEval in a malicious setting which contains three functionalities:

$$\mathcal{K}_\wedge^* \leftarrow \mathsf{CondEval.Setup}^*(\wedge, 1^\lambda, \ell_0, \ell_1, m),$$
$$(y)^m, [\sigma]^A \leftarrow \mathsf{CondEval.Eval}^*(\wedge, \mathcal{K}_\wedge^*, (s)^m, [\![x]\!]^A) \text{ and}$$
$$z \leftarrow \mathsf{CondEval.Verify}((y)^m, [\sigma]^A, \psi)$$

CondEval.Setup* is shown in Table 4, where a trustful dealer prepares m instances of FSS key pairs based on a uniform random string $\psi \leftarrow\!\!\$\ \{0,1\}^m$. More precisely, for all $i \in [m]$, let us denote K_i as the i^{th} FSS key pair prepared. Then, if $\psi[i] = 0$, the dealer outputs K_i as a *normal* FSS key pair, which means that the evaluation result is equal to $s \wedge f(x)$; Otherwise, if $\psi[i] = 1$, the dealer outputs K_i as a *trap* FSS key pair which guarantees that the corresponding evaluation result will be equal to 0 for whatever value of x. Furthermore CondEval*.Setup generates additional proof string differently due to the FSS key type (*normal* or *trap*) which are appended at each FSS key. With these designs, any manipulation from $\mathcal{A}$ in the KeyDec step of CondEval.Eval* will be captured with high probability (assuming that $\mathcal{A}$ does not know ψ). As a result, the probability that $\mathcal{A}$ flips $(s \wedge f(x))^m$ to $(\neg(s \wedge f(x)))^m$ without being detected is in negligible probability. In conclusion, compared with the construction of CondEval.Eval, this extended construction CondEval.Eval* comes at a cost of m times the computation time and communication volume when dealing with FSS gates; Nevertheless this malicious construction retains the advantage of the optimized one round communication which is of more impact in practice.

The concrete construction of CondEval.Eval* is shown in the following pseudo code, where it takes in the desired operation $\wedge$, as well as $\mathcal{K}_\wedge^*, (s)^m, [\![x]\!]^A$ from the two servers; During the online evaluation, for each $i \in [m]$, within one round of communication δ_{x^i} is revealed, and simultaneously one cleartext FSS key is obtained by each of the two servers by decrypting one FSS key from other party's choice; Finally, CondEval.Eval* outputs $(y)^m$ and an associated proof $[\sigma]^A$ which can only be validated if they could pass the verification procedure in functionality CondEval.Verify as shown below.

Functionality $(y)^m, [\sigma]^A \leftarrow \mathsf{CondEval.Eval}^*(\wedge, \mathcal{K}_\wedge^*, (s)^m, [\![x]\!]^A)$

Players: S_0, S_1.
Functionality: $(y)^m, [\sigma]^A \leftarrow \mathsf{CondEval.Eval}^*(\wedge, \mathcal{K}_\wedge^*, (s)^m, [\![x]\!]^A)$.
Input: For each $b \in \{0,1\}$ that $\mathcal{K}_\wedge^*[b] = \{\{C_{i,b}, \mathsf{SK}_{i,b}, [\![r_i]\!]_b^A\}_{i\in[m]}, [\psi]_b^B, [\Delta]_b^A\}$ are obtained by S_b in the setup phase, and authenticated secret sharing $(s)^m$ whose secret authentication key is equal to ψ in $\mathcal{K}_\wedge^*$.
Output: $((s \wedge f(x))^m, [\sigma]^A)$.

$(\{[s_1]^B, \cdots, [s_m]^B\}) \leftarrow (s)^m$
$\bar{\boldsymbol{r}} \leftarrow \mathcal{F}_{\mathsf{rand}}(1^\lambda, m, \mathbb{Z}_{2^{\ell_0+\ell_1}})$
$\boldsymbol{V} \leftarrow [\emptyset]^m$
for $b = 0$ to 1 **do**
  $[\sigma]_b^A \leftarrow 0$
  $\{C_b, \mathsf{SK}_b, [\![\boldsymbol{r}]\!]_b^A, [\psi]_b^B\} \leftarrow \mathcal{K}_\wedge^*[b]$
for $i = 1$ to m **do**
  $[\![\delta_{x^{(i)}}]\!]^A \leftarrow [\![x]\!]^A + [\![r_i]\!]^A$
  $\delta_{x^{(i)}} \leftarrow \mathcal{F}_{\mathsf{Reveal}}([\delta_{x^{(i)}}]^A)$
  $\boldsymbol{V}[i] \leftarrow (\delta_{x^{(i)}}, [\Delta \cdot \delta_{x^{(i)}}]^A, \bar{\boldsymbol{r}}[i])$
  for $b = 0$ to 1 **do**
    S_b: Sends $\{[s_i]_b^B \oplus \pi_{1-b}, \mathsf{sk}_{[s_i]_b^B}^{(1-b)}\}$ to S_{1-b} as $\{p^{(1-b)}, \mathsf{sk}^{(1-b)}\}$
▷ Key decryption
  for $b = 0$ to 1 **do**
    S_b: $k^{(b)} \| \mathsf{id}^{(b)} \| \xi^{(b)} \leftarrow C_{i,b}[p^{(b)}] \oplus \mathsf{sk}^{(b)}$
    $[\sigma]_b^A \leftarrow [\sigma]_b^A + \xi^{(b)}$
    $[y_i]_b^B \leftarrow \mathsf{Eval}^f(k^{(b)}, \mathsf{id}^{(b)}, \delta_{x^{(i)}})$
$[\varsigma]^A \leftarrow \mathcal{F}_{\mathsf{MacVryGen}}([\Delta]^A, \boldsymbol{V})$
$[\sigma]^A \leftarrow [\varsigma]^A + [\sigma]^A$
$(y)^m \leftarrow \{[y_1]^B, \cdots, [y_m]^B\}$
Output $(y)^m, [\sigma]^A$.

As the same as in CondEval in the semi-honest setting, we also have correctness, security and additional soundness proof of our constructions in the malicious setting. However, due to page limit, we have attached the corresponding theorems and their associated proofs in Appendix A.

5 SECURE COSINE SIMILARITY COMPUTATION AND VERIFICATION

In this section, we provide comprehensive constructions for cosine similarity back-end verification (matching fresh and reference templates) in both semi-honest and malicious settings, utilizing building blocks from Section 4. For example, in the semi-honest setting within a pre-processing model, our proposed protocol initially calculates a Boolean secret sharing $[c_1]^B$ and an arithmetic secret sharing $[z]^A$ using optimized secret sharing and FSS. Then, it employs $\mathsf{CondEval.Eval}(\wedge, [c_1]^B, [x]^A)$ to compute $[c_1 \wedge \mathsf{Sign}(z)]^B$ as the final authentication bit.

Let $\psi \leftarrow_\$ \{0,1\}^m, [\psi]_0^{\mathsf{B}} \leftarrow_\$ \{0,1\}^m, [\psi]_1^{\mathsf{B}} \leftarrow \psi \oplus [\psi]_0^{\mathsf{B}}; \Delta \leftarrow_\$ \mathbb{Z}_{2^{\ell_1}}, [\Delta]^{\mathsf{A}} \leftarrow \mathsf{SS.share}(\Delta, \mathbb{Z}_{2^{\ell_0+\ell_1}})$ then it does *KeyGen* and *KeyEnc* for each $i \in [m]$ as follows:		
KeyGen	If $\psi[i] = 0$ **(Normal)**	Otherwise if $\psi[i] = 1$ **(Trap)**
	$\beta_1 \leftarrow 1, \beta_2 \leftarrow 0$	$\beta_1 \leftarrow 0, \beta_2 \leftarrow 0$
	$r_i \leftarrow_\$ \mathbb{Z}_{2^{\ell_0}}, [r_i]^{\mathsf{A}} \leftarrow \mathsf{SS.share}(r_i, \mathbb{Z}_{2^{\ell_0+\ell_1}}), [\![r_i]\!]^{\mathsf{A}} \leftarrow \{[r_i]^{\mathsf{A}}, \mathsf{SS.share}(r_i \cdot \Delta, \mathbb{Z}_{2^{\ell_0+\ell_1}})\};$ $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, r_i, \beta_1, \beta_2, \mathbb{Z}_2)(\lvert k_0\rvert = \lvert k_1\rvert = K)$ $t \leftarrow_\$ \mathbb{Z}_2, \pi_0 \leftarrow_\$ \mathbb{Z}_2, \pi_1 \leftarrow_\$ \mathbb{Z}_2; \omega \leftarrow_\$ \mathbb{Z}_2; \xi_1 \leftarrow_\$ \mathbb{Z}_{2^{\ell_0+\ell_1}}, \xi_2 \leftarrow_\$ \mathbb{Z}_{2^{\ell_0+\ell_1}}; L := K + \ell_0 + \ell_1 + 1;$ $\mathsf{sk}_0^{(0)} \| \mathsf{sk}_1^{(0)} \| \mathsf{sk}_0^{(1)} \| \mathsf{sk}_1^{(1)} \leftarrow_\$ \{0,1\}^{4L}, \lvert \mathsf{sk}_0^{(0)}\rvert = \lvert \mathsf{sk}_1^{(0)}\rvert = \lvert \mathsf{sk}_0^{(1)}\rvert = \lvert \mathsf{sk}_1^{(1)}\rvert = L$	
KeyEnc	$M_{i,0} = \{k_t \| t \| \xi_1, k_{1-t} \| (1-t) \| \xi_1\}$ $M_{i,1} = \{k_t \| t \| (-\xi_1), k_{1-t} \| (1-t) \| (-\xi_1)\}$	$M_{i,0} = \{k_t \| t \| \xi_1, k_{1-t} \| (1-t) \| \xi_2\}$ $M_{i,1} = \{k_t \| t \| (-\xi_1), k_{1-t} \| (1-t) \| (-\xi_2)\}$
	$C_{i,0} = \mathcal{F}_{\mathsf{Permu}}(\pi_0, \{M_{i,0}[0] \oplus \mathsf{sk}_0^{(0)}, M_{i,0}[1] \oplus \mathsf{sk}_1^{(0)}\}), \mathsf{SK}_{i,0} = \{(\mathsf{sk}_0^{(1)}, \mathsf{sk}_1^{(1)}), \pi_1\}$ $C_{i,1} = \mathcal{F}_{\mathsf{Permu}}(\pi_1, \{M_{i,1}[0] \oplus \mathsf{sk}_0^{(1)}, M_{i,1}[1] \oplus \mathsf{sk}_1^{(1)}\}), \mathsf{SK}_{i,1} = \{(\mathsf{sk}_0^{(0)}, \mathsf{sk}_1^{(0)}), \pi_0\}$	
Outputs $\psi, \mathcal{K}_\wedge^* = \{\{C_{i,j}, \mathsf{SK}_{i,j}, [\![r_i]\!]^{\mathsf{A}}\}_{i\in[m],j\in\{0,1\}}, [\Delta]^{\mathsf{A}}\}$		

Table 4: The construction of $\mathcal{K}_\wedge^* \leftarrow \mathsf{CondEval.Setup}^*(\wedge, 1^\lambda, \ell_0, \ell_1, m)$ in the malicious setting, it inputs a security parameter λ, ℓ_0 and ℓ_1, where ℓ_0 is both used in generating FSS keys and it also defines the domain of the input secrets, ℓ_1 defines the domain of authenticated secret key Δ, $m \in \mathbb{Z}^+$ defines the secret key for authenticated Boolean secret sharing.

Functionality $z \leftarrow \mathsf{CondEval.Verify}((y)^m, [\sigma]^{\mathsf{A}}, \psi)$

Players: $\mathsf{S}_0, \mathsf{S}_1, \mathsf{B}$.
Functionality: $s \leftarrow \mathsf{CondEval.Verify}(\{((y)_b^m, [\sigma]_b^{\mathsf{A}})\}_{b\in\{0,1\}}, \psi)$.
Input: $((y)_b^m, [\sigma]_b^{\mathsf{A}})$ from S_b for each $b \in \{0,1\}$, and ψ from B.
Output: $z \in \{-1, 0, 1, \perp\}$.

$z \leftarrow -1$
if $\psi \neq \{1\}^m$ **then**
  if $[\sigma]_0^{\mathsf{A}} + [\sigma]_1^{\mathsf{A}} = 0$ **then**
    for $i = 1$ to m **do**
      $y_i \leftarrow [y_i]_0^{\mathsf{B}} \oplus [y_i]_1^{\mathsf{B}}$
      if $\psi[i] = 0$ **then**
        if $z = -1$ **then**
          $z \leftarrow y_i$
        else
          if $y_i \neq s$ **then**
            $z \leftarrow \perp$, **abort**
      else
        if $y_i \neq 0$ **then**
          $z \leftarrow \perp$, **abort**
  else
    $z \leftarrow \perp$, **abort**
Outputs z.

FUNCTIONALITY $\langle x\rangle^{\mathsf{A}}$ or $\perp \leftarrow \mathcal{F}_{\mathsf{Input}}(x, [r_{\mathsf{in}}]^{\mathsf{A}})$:
Players: A client C, S_0 and S_1.
Input: A secret x from C, and $[r_{\mathsf{in}}]^{\mathsf{A}}$ from $\mathsf{S}_0, \mathsf{S}_1$.
Output: $\langle x\rangle^{\mathsf{A}}$ or $\perp$.

1: $\forall b \in \{0,1\}$, S_b sends $[r_{\mathsf{in}}]_b^{\mathsf{A}}$ to C.
2: $\forall b \in \{0,1\}$, C returns $\delta_x^b = x + [r_{\mathsf{in}}]_0^{\mathsf{A}} + [r_{\mathsf{in}}]_1^{\mathsf{A}}$ to S_b.
3: $\forall b \in \{0,1\}$, S_b exchanges δ_x^b with S_{1-b} to check if δ_x^b is equal to δ_x^{1-b}. If both the verification pass, denote $\delta_x = \delta_x^0 = \delta_x^1$, then servers output $\langle x\rangle^{\mathsf{A}} = (\delta_x, [r_{\mathsf{in}}]_b^{\mathsf{A}})$, otherwise output $\perp$.

Figure 3: Robust input protocol in the semi-honest setting.

The solutions we propose for evaluating the circuit C in a 2PC model has the dual objectives of minimizing the number of communication rounds and communication volume. Our method leverages optimized secret sharing and CondEval to perform secure addition, multiplication, and comparison operations within C. The online evaluation process requires two communication rounds, with a communication cost of four ring elements (corresponds to the four times of $\mathcal{F}_{\mathsf{reveal}}$ invocations) and two decryption keys transmitted for decrypting the FSS keys.

When using floating-point numbers as input for a secure computation framework, these secret floating-point numbers need to be converted to fixed-point representation by multiplying them by a precision parameter 2^ρ. This process occurs before entering the secure computation framework, where ρ defines the preserved precision in the fraction part. In a standard way, a secure truncation operation is required to obtain the same precision when performing one multiplication over two fixed point values. This is meaningful in the sense of performing an operation within a smaller ring and also outputs an arithmetical result with the same precision. However, in the context of secure cosine similarity computation, where the input vectors are small float point numbers, while the comparison the circuit C outputs a secret bit that does not depends on the precision, in order to reduce the complexity our final scheme, we perform truncation once over $1/t^2 \cdot \mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})^2 - \mathsf{IP}(\boldsymbol{x}, \boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y}, \boldsymbol{y})$ in C.

5.1 Full protocol in the semi-honest setting

In the semi-honest setting, our final solution comprises of three sub-protocols: CS.Setup, $\mathcal{F}_{\mathsf{Input}}$ and CS.Eval, executed in the Setup, Input, and Evaluation phases, respectively. Protocol 1 details the setup phase, integrating CondEval.Setup and generating correlated randomness based on circuit C. Next, the client and B input their secret vectors $\boldsymbol{x}, \boldsymbol{y}$ by running $\mathcal{F}_{\mathsf{Input}}$ through $\mathcal{F}_{\mathsf{Input}}$, as shown in Fig. 3. Lastly, with correlated randomness and input prepared, CS.Eval produces the boolean secret sharing of the desired authentication bit $[y]^{\mathsf{B}}$, where $y = \mathsf{Sign}\ (\cos(\mathbf{x}, \mathbf{y}) - \tau)$.

5.1.1 Correctness. From Theorem 1 we know that Protocol 2 outputs $[c]^{\mathsf{B}}$ where $c = c_1 \wedge c_2, c_1 = \mathsf{Sign}(\mathsf{IP}(\boldsymbol{x}, \boldsymbol{y}))$ and $c_2 = \mathsf{Sign}(1/t^2 \cdot \mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})^2 - \mathsf{IP}(\boldsymbol{x}, \boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y}, \boldsymbol{y}))$, thus computes Sign $(\cos(\mathbf{x}, \mathbf{y}) - \tau)$ as shown in Eq. 3.

5.1.2 Security. Our goal is to prove that Protocols 1 and 2 provide a secure implementation of the circuit C when faced with a semi-honest PPT adversary $\mathcal{A}$ in a 2PC setting. We assert that, with the correlated randomness provided in protocol 1, and following the online evaluation of protocol 2, $\mathcal{A}$ learns nothing about the input $\boldsymbol{x}, \boldsymbol{y}$, or $\mathsf{Sign}(\cos(\boldsymbol{x}, \boldsymbol{y}) - \tau)$.

Proof. By applying Theorem 2, we ensure that there is no information leakage from the internal view tapes of $\mathsf{CondEval.Eval}_{\mathsf{trunc}}(\wedge, \bar{\mathcal{K}}_\wedge, [c_1]^{\mathsf{B}}, [z]^{\mathsf{A}})$. However, we need to consider the other view transcripts $\mathtt{View}_b$ for each $b \in \{0, 1\}$, where:

$$\mathtt{View}_b := \{k_b^{(1)}, \delta_u, \delta_v, \delta_w\}$$

As δ_v and δ_w hide the correlated intermediate values of $\boldsymbol{x}$ and $\boldsymbol{y}$, we argue that the view $k_b^{(1)}$ is pseudo-random (computationally indistinguishable from the real random key) as proved in Fig. 1 in [10], thereby concealing the information of $\alpha^{(1)}$ that is contained in $(k_0^{(1)}, k_1^{(1)})$ from $\mathcal{A}$. □

Protocol 1 $([\mathcal{R}]^{\mathsf{A}}, \mathcal{K}_\wedge, \mathcal{K}_0) \leftarrow \mathsf{CS.Setup}(1^\lambda, \ell, \rho)$

Players: The central bank B.
Functionality: $([\mathcal{R}]^{\mathsf{A}}, \mathcal{K}) \leftarrow \mathsf{CS.Setup}(1^\lambda, \ell, \rho)$.
Input: A security parameter λ, element encoding length ℓ, and a truncation parameter $\rho \in \mathbb{Z}^+$ where $\rho < \ell$.
Output: $([\mathcal{R}]^{\mathsf{A}}, \mathcal{K})$.

$(\boldsymbol{x_{in}}, \boldsymbol{y_{in}}) \leftarrow\!\$\, \mathbb{Z}_{2^\ell}^{2n}, \boldsymbol{u_{in}} \leftarrow \mathsf{Prod}(\boldsymbol{x_{in}}, \boldsymbol{y_{in}})$
$\boldsymbol{v_{in}} \leftarrow \mathsf{Prod}(\boldsymbol{x_{in}}, \boldsymbol{x_{in}}), \boldsymbol{w_{in}} \leftarrow \mathsf{Prod}(\boldsymbol{y_{in}}, \boldsymbol{y_{in}})$
$\mathcal{K}_\wedge \leftarrow \mathsf{CondEval.Setup}(\wedge, 1^\lambda, \ell)$
$\{C_i, \mathsf{SK}_i, [r]_i^{\mathsf{A}}\}_{i\in\{0,1\}} \leftarrow \mathcal{K}_\wedge$
$\alpha^{(2)} \leftarrow [r]_0^{\mathsf{A}} + [r]_1^{\mathsf{A}}$
$v \leftarrow\!\$\, \mathbb{Z}_{2^\rho}, [\eta]^{\mathsf{A}} \leftarrow \mathsf{SS.Share}(\mathsf{Shift}(\alpha^{(2)}, -\rho, \ell) + v, \mathbb{Z}_{2^{\ell+\rho}})$
$\bar{\mathcal{K}}_\wedge \leftarrow \{C_i, \mathsf{SK}_i, [\eta]_i^{\mathsf{A}}\}_{i\in\{0,1\}}$
$\alpha^{(1)} \leftarrow\!\$\, \mathbb{Z}_{2^\ell}$
$(k_0, k_1) \leftarrow \mathsf{Gen}_\ell^{[0,2^{\ell-1}]}(1^\lambda, \alpha^{(1)}, 1, 0, \mathbb{Z}_2)$
$\mathcal{R} \leftarrow \{\boldsymbol{x_{in}}, \boldsymbol{y_{in}}, \boldsymbol{u_{in}}, \boldsymbol{v_{in}}, \boldsymbol{w_{in}}, (r_1, r_2, r_1 r_2), \alpha^{(1)}, (\alpha^{(1)})^2\}$
$\mathcal{K}_0 \leftarrow \{k_0, k_1\}$
Outputs $([\mathcal{R}]^{\mathsf{A}}, \bar{\mathcal{K}}_\wedge, \mathcal{K}_0)$

We have also realized the full protocol in the malicious setting, due to page limit, we have attached it in Appendix B.

6 EXPERIMENT & RESULTS

In this section, to demonstrate how practical our solution is, we present experimental assessments of the proposed solution through a use case of voice biometric authentication. We first describe the experimental setting and then further evaluate our solution in the malicious model. With the same input data, we compare the performance results between a regular cleartext version implementation, an invocation of spdz2k.x from MP-SPDZ[31] and our own implementation of protocol CSB.Eval* over the circuit C.

Protocol 2 $[c]^{\mathsf{B}} \leftarrow \mathsf{CS.Eval}([\mathcal{R}]^{\mathsf{A}}, \bar{\mathcal{K}}_\wedge, \mathcal{K}_0, \langle \boldsymbol{x}\rangle^{\mathsf{A}}, \langle \boldsymbol{y}\rangle^{\mathsf{A}}, \tau, \rho, \ell)$

Players: $\mathsf{S}_0, \mathsf{S}_1$.
Functionality: $[c]^{\mathsf{B}} \leftarrow \mathsf{CS.Eval}([\mathcal{R}]^{\mathsf{A}}, \bar{\mathcal{K}}_\wedge, \mathcal{K}_0, \langle \boldsymbol{x}\rangle^{\mathsf{A}}, \langle \boldsymbol{y}\rangle^{\mathsf{A}}, \tau, \rho, \ell)$.
Input: $[\mathcal{R}]^{\mathsf{A}}, \mathcal{K}_\wedge, \mathcal{K}_0, \langle \boldsymbol{x}\rangle^{\mathsf{A}}, \langle \boldsymbol{y}\rangle^{\mathsf{A}}$ from S_0 and S_1 respectively. A public threshold $\tau \in (0, 1]$, element encoding length $\ell \in \mathbb{Z}^+$, and a truncation parameter $\rho \in \mathbb{Z}^+$ where $\rho < \ell$.
Output: $[c]^{\mathsf{B}}$.

$\{k_0, k_1\} \leftarrow \mathcal{K}_0$
$\{\boldsymbol{x_{in}}, \boldsymbol{y_{in}}, \boldsymbol{u_{in}}, \boldsymbol{v_{in}}, \boldsymbol{w_{in}}, (r_1, r_2, r_1 r_2), \alpha^{(1)}, (\alpha^{(1)})^2\} \leftarrow \mathcal{R}$
$[u]^{\mathsf{A}} \leftarrow \mathsf{SS.IP}(\langle \boldsymbol{x}\rangle^{\mathsf{A}}, \langle \boldsymbol{y}\rangle^{\mathsf{A}}, [\boldsymbol{u_{in}}]^{\mathsf{A}})$
$[v]^{\mathsf{A}} \leftarrow \mathsf{SS.IP}(\langle \boldsymbol{x}\rangle^{\mathsf{A}}, \langle \boldsymbol{x}\rangle^{\mathsf{A}}, [\boldsymbol{v_{in}}]^{\mathsf{A}})$
$[w]^{\mathsf{A}} \leftarrow \mathsf{SS.IP}(\langle \boldsymbol{y}\rangle^{\mathsf{A}}, \langle \boldsymbol{y}\rangle^{\mathsf{A}}, [\boldsymbol{w_{in}}]^{\mathsf{A}})$
$\delta_u \leftarrow \mathcal{F}_{\mathsf{Reveal}}([u + \alpha^{(1)}]^{\mathsf{A}})$ ▷ First round
$\langle u\rangle^{\mathsf{A}} \leftarrow (\delta_u, [\alpha^{(1)}]^{\mathsf{A}})$
$\langle v\rangle^{\mathsf{A}} \leftarrow (\mathcal{F}_{\mathsf{Reveal}}([v + r_1]^{\mathsf{A}}), [r_1]^{\mathsf{A}})$ ▷ First round
$\langle w\rangle^{\mathsf{A}} \leftarrow (\mathcal{F}_{\mathsf{Reveal}}([w + r_2]^{\mathsf{A}}), [r_2]^{\mathsf{A}})$ ▷ First round
for $b = 0$ to 1 **do**
  $[c_1]_b^{\mathsf{B}} \leftarrow \mathsf{Eval}_\ell^{[0,2^{\ell-1}]}(k_b, b, \delta_u)$
$T \leftarrow \mathsf{Shift}(\frac{1}{\tau^2}, -p, \ell)$ ▷ after shifting, $T \in \mathbb{Z}_{2^\ell}$
$[z]^{\mathsf{A}} \leftarrow T \cdot \mathsf{SS.MUL}(\langle u\rangle^{\mathsf{A}}, \langle u\rangle^{\mathsf{A}}, [(\alpha^{(1)})^2]^{\mathsf{A}})$
$[z]^{\mathsf{A}} \leftarrow [z]^{\mathsf{A}} - 2^\rho \cdot \mathsf{SS.MUL}(\langle v\rangle^{\mathsf{A}}, \langle w\rangle^{\mathsf{A}}, [r_1 r_2]^{\mathsf{A}})$
$[c]^{\mathsf{B}} \leftarrow \mathsf{CondEval.Eval}_{\mathsf{trunc}}(\wedge, \bar{\mathcal{K}}_\wedge, [c_1]^{\mathsf{B}}, [z]^{\mathsf{A}})$ ▷ Second round
Outputs $[c]^{\mathsf{B}}$.

	FAR (%)	FRR (%)	LAN (ms)	WAN (ms)	Com(KB)
Cleartext	0.575	0.575	0.17	-	-
spdz2k	0.573	0.580	74	8020	1789
Ours (m=10)	0.573	0.580	57	457	3.5
Ours (m=20)	0.573	0.580	102	500	6.9
Ours (m=40)	0.573	0.580	201	628	13.72

Table 5: Evaluation comparison results between the cleartext version, spdz2k and CSB.Eval*. The computation and communication cost are measured per party.

6.1 Data preparation

We propose to use a pre-trained model of the ECAPA-TDNN [22] model available in [18] to extract speaker embeddings. The model was trained using the development part of the VoxCeleb2 dataset [15] with 5994 speakers. The datasets RIR [34] and MUSAN [48] were also used for data augmentation. The model is composed of 3 SE-Res2Block modules. The channel size and the dimension of the bottleneck in the SEBlock are set to 1024 and 256, respectively [19]. The entire utterance is fed into the ECAPA-system to finally obtain a vector of 192-dimensional speaker embedding. We get our experiments' input from the test set of VoxCeleb1, which consists of 37720 pairs of enrolment and verification ECAPA-TDNN embeddings, in which we have approximately half target and half non-target speakers.

6.2 Evaluation setting

We perform all benchmarks on three droplets provided by DigitalOcean where two of them are in the same LAN located in Singapore, and the third one located in Frankfurt connected by WAN with one of the two droplets in Singapore. We measured 0.47ms latency with 1.8Gbps in our LAN setting, and 152ms latency with 130Mbps in

our WAN setting. All of the droplets have similar configurations, with each one having a relatively low specification of 4GB memory and 2-core CPUs, running Ubuntu 22.04 LTS. The cleartext version implementation of C is in Python 3.10, directly using float point numbers as input. We implemented our protocol CSB.Eval* in Python 3.10, in which the IC-FSS module are implemented from Fig. 3 of [10].

To use our implementation of CSB.Eval* and spdz2k for the raw input vectors, we need to convert them from floating-point numbers to integers while maintaining precision in the fractional part. We achieve this by multiplying the floating-point numbers with 2^8 ($\rho = 8$) and keeping only the integer digits. In our experiments, we set the same parameters for both experiments, namely $\lambda = 40, \ell_0 = 64, \ell_1 = 48$. This ensures a fair performance comparison between our implementation and the procool spdz2k from MP-SPDZ. We also set the input domain of IC-FSS as 32 bits, which covers our use case. To evaluate the practical performance with different levels of soundness (Theroem 5) where the value of m is dominant, we vary the value of m between $10, 20, 40$.

6.3 Evaluation results

Table 5 presents the experimental results of computing functionality C with one pair of enrolment (reference template) and verification vectors (fresh template) as input. The costs shown for spdz2k and CSB.Eval* exclude the offline phase generation cost, and the computation time cost in both LAN and WAN settings is measured for 100 runs on average.

Our observations reveal that employing a conversion parameter of $\rho = 8$ yields a non-significant deviation (0.003%) in performance, as represented by the false acceptance rate (FAR) and false rejection rate (FRR), when compared to the cleartext version. It is important to note that the threshold was established at a point where the FAR and FRR are equal in the cleartext scenario.

In terms of communication cost, our protocol outperforms spdz2k from MP-SPDZ by more than 100 times for all our test cases ($m = 10, 20, 40$). With respect to computation cost, our round-efficient design enables our protocol to complete the online evaluation in less than 700 milliseconds for all test cases in the WAN setting, whereas spdz2k takes around 8 seconds. However, in a LAN setting where the network latency can be ignored (less than 0.47ms), spdz2k performs better than our system due to their efficient C++ implementation. Since our current implementation is a fast prototype, there is significant room for improvement. This could involve reimplementing our protocol using C++ or using a more efficient variant construction for the core FSS module, such as the one proposed in [27].

In summary, our experimental results demonstrate that our CSB.Eval* offers a practical solution even in the WAN setting, where the network latency is a bottleneck for efficient secure computation, surpassing the state-of-the-art in terms of the communication volume and the computation time for computing circuit C, making it suitable for real-world applications.

7 CONCLUSION

Privacy-preserving cosine similarity computation and comparison to a predefined threshold is an important building block that has multiple applications (*e.g.,* biometric authentication and identification, privacy-preserving machine learning). In this paper, we introduce two novel protocols to compute the cosine similarity and compare to a threshold in a privacy-preserving way. One protocol is suitable for both the semi-honest and malicious setting, both protocols rely on the recent advances of Function Secret Sharing (FSS) [10, 12], and the one for malicious setting addtionally relies on 2PC authenticated secret sharing and our proposed use of multiple instances of random FSS keys (either *normal* or *trap*). All our protocols are based on a new building block we introduce called CondEval that allows computing the composition of an input bit s and a binary function f (evaluated via FSS) on an input x *i.e.,* $s \circ f(x)$. This introduced building block is provably secure under both the semi-honest and malicious setting, is general and of independent interest; Thus, could be used in general 2PC computations.

Furthermore, we provide a detailed security analysis of the proposed protocols and introduced building block and evaluate the proposed protocols in the biometric authentication setting. Our results show that the proposed protocols are not only efficient (requiring two communication rounds) but also maintain the same accuracy as plain-text systems. To the best of our knowledge this is the first biometric authentication protocol in the malicious setting.

ACKNOWLEDGMENTS

This work is supported by the TReSPAsS-ETN project funded from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 860813. It is also supported by the ANR-DFG RESPECT project.

REFERENCES

[1] Shashank Agrawal and David J Wu. 2017. Functional encryption: deterministic to randomized functions from simple assumptions. In *Advances in Cryptology–EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part II 36*. Springer, 30–61.

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 805–817.

[3] Manuel Barbosa, Dario Catalano, Azam Soleimanian, and Bogdan Warinschi. 2019. Efficient function-hiding functional encryption: From inner-products to orthogonality. In *Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings*. Springer, 127–148.

[4] Mauro Barni, Giulia Droandi, Riccardo Lazzeretti, and Tommaso Pignata. 2019. SEMBA: secure multi-biometric authentication. *IET Biometrics* 8, 6 (2019), 411–421.

[5] Donald Beaver. 1992. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology — CRYPTO '91*, Joan Feigenbaum (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 420–432.

[6] Donald Beaver, Silvio Micali, and Phillip Rogaway. 1990. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 503–513.

[7] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. 2019. Turbospeedz: double your online SPDZ! improving SPDZ using function dependent preprocessing. In *International Conference on Applied Cryptography and Network Security*. Springer, 530–549.

[8] Vishnu Naresh Boddeti. 2018. Secure face matching using fully homomorphic encryption. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*. IEEE, 1–10.

[9] Dan Boneh, Amit Sahai, and Brent Waters. 2011. Functional encryption: Definitions and challenges. In *Theory of Cryptography: 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings 8*. Springer, 253–273.

[10] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. 2021. Function secret sharing for mixed-mode

and fixed-point secure computation. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part II*. Springer, 871–900.

[11] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 337–367.

[12] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2019. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part I 17*. Springer, 341–371.

[13] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2019. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography Conference*. Springer, 341–371.

[14] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. 2019. Numerical method for comparison on homomorphically encrypted numbers. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part II*. Springer, 415–445.

[15] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. 2018. Voxceleb2: Deep speaker recognition. *arXiv preprint arXiv:1806.05622* (2018).

[16] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. 2018. SPD$\mathbb{Z}_{2^k}$: efficient MPC mod 2^k for dishonest majority. In *Annual International Cryptology Conference*. Springer, 769–798.

[17] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. 2013. Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security*. Springer, 1–18.

[18] Rohan Kumar Das, Ruijie Tao, and Haizhou Li. 2021. HLT-NUS SUBMISSION FOR 2020 NIST Conversational Telephone Speech SRE. *arXiv preprint arXiv:2111.06671* (2021).

[19] Rohan Kumar Das, Ruijie Tao, and Haizhou Li. 2021. HLT-NUS submission for 2020 NIST conversational telephone speech SRE. *arXiv preprint arXiv:2111.06671* (2021).

[20] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. 2016. Functional encryption for inner product with full function privacy. In *Public-Key Cryptography–PKC 2016: 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*. Springer, 164–195.

[21] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In *NDSS*.

[22] Brecht Desplanques, Jenthe Thienpondt, and Kris Demuynck. 2020. Ecapa-tdnn: Emphasized channel attention, propagation and aggregation in tdnn based speaker verification. *arXiv preprint arXiv:2005.07143* (2020).

[23] Diana-Elena Fălămaș, Kinga Marton, and Alin Suciu. 2021. Assessment of Two Privacy Preserving Authentication Methods Using Secure Multiparty Computation Based on Secret Sharing. *Symmetry* 13, 5 (2021), 894.

[24] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[25] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Stanford university.

[26] Oded Goldreich, Silvio Micali, and Avi Wigderson. 2019. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 307–328.

[27] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. 2023. Half-tree: Halving the cost of tree expansion in cot and dpf. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I*. Springer, 330–362.

[28] Haiping Huang, Tianhe Gong, Ping Chen, Reza Malekian, and Tao Chen. 2016. Secure two-party distance computation protocol based on privacy homomorphism and scalar product in wireless sensor networks. *Tsinghua Science and Technology* 21, 4 (2016), 385–396.

[29] Alberto Ibarrondo, Hervé Chabanne, and Melek Önen. 2022. Funshade: Functional Secret Sharing for Two-Party Secure Thresholded Distance Evaluation. *Cryptology ePrint Archive* (2022).

[30] Ilia Iliashenko and Vincent Zucca. 2021. Faster homomorphic comparison operations for BGV and BFV. *Proceedings on Privacy Enhancing Technologies* 2021, 3 (2021), 246–264.

[31] Marcel Keller. 2020. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 1575–1590.

[32] Taeyun Kim, Yongwoo Oh, and Hyoungshick Kim. 2020. Efficient privacy-preserving fingerprint-based authentication system using fully homomorphic encryption. *Security and Communication Networks* 2020 (2020), 1–11.

[33] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems* 34 (2021), 4961–4973.

[34] Tom Ko, Vijayaditya Peddinti, Daniel Povey, Michael L Seltzer, and Sanjeev Khudanpur. 2017. A study on data augmentation of reverberant speech for robust speech recognition. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5220–5224.

[35] Shaofeng Lu, Cheng Li, Xinyi Feng, Yuefeng Lu, Yulong Hu, and Wenxi Li. 2021. Privacy-preserving Hamming distance Protocol and Its Applications. In *2021 2nd International Conference on Electronics, Communications and Information Technology (CECIT)*. IEEE, 848–853.

[36] Silvio Micali, Oded Goldreich, and Avi Wigderson. 1987. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*. ACM, 218–229.

[37] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 35–52.

[38] Mahesh Kumar Morampudi, Munaga VNK Prasad, and USN Raju. 2020. Privacy-preserving iris authentication using fully homomorphic encryption. *Multimedia Tools and Applications* 79 (2020), 19215–19237.

[39] Andreas Nautsch, Jose Patino, Amos Treiber, Themos Stafylakis, Petr Mizera, Massimiliano Todisco, Thomas Schneider, and Nicholas Evans. 2019. Privacy-preserving speaker recognition with cohort score normalisation. *arXiv preprint arXiv:1907.03454* (2019).

[40] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. {ABY2.0}: Improved {Mixed-Protocol} Secure {Two-Party} Computation. In *30th USENIX Security Symposium (USENIX Security 21)*. 2165–2182.

[41] Yogachandran Rahulamathavan. 2022. Privacy-preserving similarity calculation of speaker features using fully homomorphic encryption. *arXiv preprint arXiv:2202.07994* (2022).

[42] Yogachandran Rahulamathavan, Safak Dogan, Xiyu Shi, Rongxing Lu, Muttukrishnan Rajarajan, and Ahmet Kondoz. 2020. Scalar product lattice computation for efficient privacy-preserving systems. *IEEE Internet of Things Journal* 8, 3 (2020), 1417–1427.

[43] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 325–342.

[44] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis Bach. 2020. Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. *arXiv preprint arXiv:2006.04593* (2020).

[45] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis Bach. 2022. Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. *Proceedings on Privacy Enhancing Technologies* 2022, 1 (2022), 291–316.

[46] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[47] LI Shundong, ZHANG Mengyu, and XU Wenting. 2021. Secure Scalar Product Protocols. *Chinese Journal of Electronics* 30, 6 (2021), 1059–1068.

[48] David Snyder, Guoguo Chen, and Daniel Povey. 2015. Musan: A music, speech, and noise corpus. *arXiv preprint arXiv:1510.08484* (2015).

[49] Junichi Tomida, Masayuki Abe, and Tatsuaki Okamoto. 2016. Efficient functional encryption for inner-product values with full-hiding security. In *Information Security: 19th International Conference, ISC 2016, Honolulu, HI, USA, September 3-6, 2016. Proceedings 19*. Springer, 408–425.

[50] Amos Treiber, Andreas Nautsch, Jascha Kolberg, Thomas Schneider, and Christoph Busch. 2019. Privacy-preserving PLDA speaker verification using outsourced secure computation. *Speech Communication* 114 (2019), 60–71.

[51] Florian Van Daalen, Lianne Ippel, Andre Dekker, and Inigo Bermejo. 2023. Privacy Preserving n-Party Scalar Product Protocol. *IEEE Transactions on Parallel and Distributed Systems* (2023).

[52] Thijs Veugen, Robbert de Haan, Ronald Cramer, and Frank Muller. 2014. A framework for secure computations with two non-colluding servers and multiple clients, applied to recommendations. *IEEE Transactions on Information Forensics and Security* 10, 3 (2014), 445–457.

[53] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. F: Honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies* 2021, 1 (2021), 188–208.

[54] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 162–167.

[55] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 162–167.

A FURTHER THEOREMS AND ASSOCIATED PROOFS

We attach these theorems and their proofs of our CondEval constructions of section 4.3 in the following.

Theorem 3. *Correctness.* Assuming all servers indeed honestly follow CondEval.Eval$^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, [\![x]\!]^{\mathsf{A}})$, *i.e.,* none of the servers deviate from the protocol description, then they obtain $((y)^m, [\sigma]^{\mathsf{A}})$. Denote p the probability that CondEval.Verify$((y)^m, [\sigma]^{\mathsf{A}}, \psi)$ is equal to $s \wedge f(x)$, we claim $p = 1 - 1/2^m$.

Proof. Assume all servers honestly follow CondEval.Eval$^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, [\![x]\!]^{\mathsf{A}})$. $\forall i \in [m]$, if $\psi[i] = 0$. From Theorem 1, we know $[y_i]^{\mathsf{B}}$ is equal to $[s \wedge f(x)]^{\mathsf{B}}$ as expected. Otherwise if $\psi[i] = 1$, since $[s]^{\mathsf{B}}_0, [s]^{\mathsf{B}}_1$ is either $\{0,0\}$ or $\{1,1\}$, which implies $y_i = \mathsf{Eval}^f(k_t, t, \delta_x) \oplus \mathsf{Eval}^f(k_t, t, \delta_x)$ or $y_i = \mathsf{Eval}^f(k_{1-t}, 1-t, \delta_x) \oplus \mathsf{Eval}^f(k_{1-t}, 1-t, \delta_x)$, then $[y_i]^{\mathsf{B}}$ is equal to $[0]^{\mathsf{B}}$. Thus, the functionality outputs a valid authenticated Boolean secret sharing. However, in the case of $\psi = \{0,1\}^m$ which has a probability of $1/2^m$ the protocol CondEval.Verify$((y)^m, [\sigma]^{\mathsf{A}}, \psi)$ outputs -1. Thus, p is equal to $1 - 1/2^m$. □

Theorem 4. *Security.* In the presence of an active PPT adversary $\mathcal{A}$ among two servers in CondEval.Eval$^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, [\![x]\!]^{\mathsf{A}})$, where $\mathcal{K}^*_\wedge$ comes from the output of functionality CondEval.Setup*, we assert that $\mathcal{A}$ learns no information about x, s, $s \wedge f(x)$ or ψ.

Proof. After online evaluation, for each $b \in \{0,1\}$ we denote S_b's transcript view are composed of transcript views of CondEval.Eval$(\mathcal{K}^*_\wedge[i], [s_i]^{\mathsf{B}}, [x]^{\mathsf{A}}_\wedge)$ for each $i \in [m]$, thus, from Theorem 2, we know that CondEval.Eval$^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, [\![x]\!]^{\mathsf{A}})$ does not leak any information about x, s, or $s \wedge f(x)$. Furthermore, owing to the computational indistinguishability of FSS keys from entirely random keys, based on the security of the pseudorandom generator [10], a probabilistic polynomial-time (PPT) bounded $\mathcal{A}$ cannot distinguish between a trap key instance, a completely random key, or a normal key instance with significant probability. Consequently, $\mathcal{A}$ gains no information about ψ. □

Theorem 5. *Soundness.* Assuming the existence of an active PPT adversary $\mathcal{A}$ (S_0 or S_1) when performing CondEval.Eval$^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, [\![x]\!]^{\mathsf{A}})$ that outputs $((y)^m, [\sigma]^{\mathsf{A}})$. We denote by p the probability that CondEval.Verify$((y)^m, [\sigma]^{\mathsf{A}}, \psi) = \neg(s \wedge f(x))$. We claim

$$p < 2^{-\ell_1 + \log(\ell_1+1)} + 2^{1-m} + \left(\frac{3}{4}\right)^m \cdot 2^{-(\ell_0+\ell_1)}$$

where ℓ_1 denotes the length of the MAC key.

Proof. If CondEval.Verify$((y)^m, [\sigma]^{\mathsf{A}}, \psi) = \neg(s \wedge f(x))$, *w.l.o.g.,* we assume S_0 is the malicious server in the functionality CondEval*. The actual computation can be performed in two ways:

- S_0 honestly follows CondEval.Eval$^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, [\![x]\!]^{\mathsf{A}})$.
- S_0 arbitrarily deviates from CondEval.Eval$^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, [\![x]\!]^{\mathsf{A}})$, either by flipping $[s_i]^{\mathsf{B}}_0 (i \in [m])$ when performing the *KeyDec* step OT, or dishonestly flipping partial shares when reporting $(s)^m$ to the receiver.

We denote by p_1 and p_2 as the probability that CondEval.Verify $((y)^m, [\sigma]^{\mathsf{A}}, \psi) = \neg(s \wedge f(x))$ for each of the two cases above respectively.

In the first case, from Theorem 1 we know with probability of $1/2^m$ the protocol outputs -1, and with probability of $1 - 1/2^m$ it outputs $s \wedge f(x)$, but it never outputs $\neg(s \wedge f(x))$. Thus, $p_1 = 0$.

For the second case, there are three events A, B, C that S_0 may manipulate the protocol and thus, may result in outputting a flipped authentication bit $(\neg(s \wedge f(x)))^m$:

(1) A: S_0 reports one or more dishonest arithmetical shares, submits a manipulated proof τ^*_0 such that it still guarantees $\tau^*_0 + \tau_1 = 0$, and resulting to a valid flipped authenticated bit $(y^*)^m$ where $y^* = \neg(s \wedge f(x))$.
(2) $A \wedge B$: S_0 manages to flip the compuation result $(y)^m$ to $(\neg y)^m$ by flipping $[s_i]^{\mathsf{B}}_0$ for all $\psi[i] = 0, i \in [m]$ when performing the *KeyDec* step, and also presents a correct proof that passes the final verification.
(3) C: $\forall i \in [m]$, S_0 manages to guess each $\psi[i]$ correctly, thus, S_0 flips $[y_i]^{\mathsf{B}}_0$ if $\psi[i] = 0$, otherwise if $\psi[i] = 1$ keeps $[y_i]^{\mathsf{B}}_0$ unchanged. In the end S_0 submits his manipulated sharing of $(y)^m$ to B.

Each event is considered independently. Concerning event A, according to [16], it is known that the probability of event A occurring is no greater than $2^{-\ell_1 + \log(\ell_1+1)}$.

Regarding event B, the final outcome is only altered when $f(x) = 1$ and all the *normal* indexes ($\psi[i] = 0$) must be flipped during the oblivious transfer, as we know ψ is uniformly determined. We analyse the probability of S_0 successfully flipping all *normal* indexes without being detected as follows. First, we distinguish two complementary events B_1, B_2 that are covered by B. We denote B_1 as the event that S_0 correctly guesses $\psi[i]$ for all $i \in [m]$. Then we have

$$P(B_1) = \left(\frac{1}{4} + \frac{1}{4}\right)^m = \frac{1}{2^m}.$$

Additionally, we denote B_2 as the event when S_0 correctly guesses all $\psi[i]$ where $\psi[i] = 0$, but not perfectly correctly guesses all $\psi[i]$ where $\psi[i] = 1$, *i.e.,* S_0 wrongly determined $\psi[j] = 0$ by at least one index j where in fact $\psi[j]$ is equal to 0. The reason why we distinguish B_1 and B_2 is because for event B_1 the adversary S_0 does not need to do anything to pass the verification. However, for the event B_2 the adversary S_0 needs to select a random offset value e added to $[\sigma]^{\mathsf{A}}_0$ to pass the verification (it happens with probability $2^{-(\ell_0+\ell_1)}$), the reason is because in event B_2 that S_0 wrongly assumed $\psi[j] = 0$ by at least one index j where in fact when $\psi[j] = 1$, under this case servers obtain an associated secret sharing of a non-zero random value. Thus, we have

$$P(B_2) = \left(\left(\frac{3}{4}\right)^m - P(B_1)\right) \cdot 2^{-(\ell_0+\ell_1)} < \left(\frac{3}{4}\right)^m \cdot 2^{-(\ell_0+\ell_1)}.$$

Denote $p^* \in [0,1]$ the probability when the input $f(x)$ is equal to 1, then we have

$$P(B) = P^* \cdot (P(B_1) + P(B_2)) < P^* \cdot \left(\frac{1}{2^m} + \left(\frac{3}{4}\right)^m \cdot 2^{-(\ell_0+\ell_1)}\right)$$

Denote $P(B^*)$ the probability that an occurrence of event B results in an flipped result $y = \neg(s \wedge f(x))$, which we know it happens only when $\psi \neq \{1\}^m$, so we have $P(B^*) < P(B)$.

For event C, $P(C) = 1/2^m$. Thus,

$$\begin{aligned} p_2 &< P(A) + P(B^*) + P(C) \\ &< P(A) + P(B) + P(C) \\ &< 2^{-\ell_1 + \log(\ell_1+1)} + \frac{p^*+1}{2^m} + \left(\frac{3}{4}\right)^m \cdot 2^{-(\ell_0+\ell_1)} \\ &< 2^{-\ell_1 + \log(\ell_1+1)} + 2^{1-m} + \left(\frac{3}{4}\right)^m \cdot 2^{-(\ell_0+\ell_1)}. \end{aligned}$$

In conclusion,

$$p = p_1 + p_2 < 2^{-\ell_1 + \log(\ell_1+1)} + 2^{1-m} + \left(\frac{3}{4}\right)^m \cdot 2^{-(\ell_0+\ell_1)}.$$

□

B FULL PROTOCOL IN THE MALICIOUS SETTING

We now present Protocols 3 and 4 which evaluate the circuit C in a stronger threat model, with one malicious server who might deviate from the protocol description in Protocol 2. More specifically:

- In protocol 3, B first runs CondEval*.Setup(1^λ), then from those results it obtains Δ and ψ, which are secret keys for authenticating an arithmetical revelation and a Boolean revelation in the later online evaluation. Afterwards, B additionally generates an authenticated random secret sharing $\mathcal{I}_x, \mathcal{I}_y$ to be used in input phase. It also generates a function dependent authenticated correlated random secret sharing $[\![\mathcal{R}]\!]^{\mathsf{A}}$ from Δ, and random function secret sharing keys $\mathcal{K}_0$ from ψ for computing c_1;
- After the setup phase is done, whenever the input is ready, (*i.e.,* the client provides her fresh template $\boldsymbol{x}$), the client and the two servers coordinately run:

$$\langle\!\langle \boldsymbol{x}_i \rangle\!\rangle^{\mathsf{A}} \leftarrow \mathcal{F}_{\mathsf{Input}^*}(\boldsymbol{x}_i, [\Delta]^{\mathsf{A}}, [\![\boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{x})}[i]]\!]^{\mathsf{A}}, [\![\boldsymbol{x}_{\boldsymbol{in}}[i]]\!]^{\mathsf{A}})$$

  where $\mathcal{F}_{\mathsf{Input}^*}$ is shown in Fig. 4;
- Then in the evaluation phase, the two servers execute the online evaluation Protocol 4 CS.Eval* in which we incorporate a variant version of CondEval.Eval* denoted as CondEval.Eval$^*_{\mathsf{trunc}}$, whose sole difference to the former is to perform a local truncation after each δ_x is revealed (similarly to the process applied in the semi-honest setting);
- After completing the evaluation phase in Protocol 4, the servers obtain a final authenticated boolean secret sharing $(y)^m$ and an associated proof $[\sigma]^{\mathsf{A}}$.
- Upon receiving $(y)^m$ and $[\sigma]^{\mathsf{A}}$ from the servers, with value ψ already known in the setup phase, B runs CondEval.Verify $((y)^m, [\sigma]^{\mathsf{A}}, \psi)$ and outputs a symbol $z \in -1, 0, 1, \perp$. B accepts the authentication bit as z if z is not in $\{-1, \perp\}$.

We note here that via Protocol 4, we are able to capture a client who may act maliciously *i.e.,* may attempt to impersonate a legitimate user and deduce information for the corresponding templates. Thus, we incorporate a mechanism to check that the secret sharing of the fresh (as well as the reference) template have been secret shared correctly.

Protocol 3 $(\mathcal{I}_x, \mathcal{I}_y, [\![\mathcal{R}]\!]^{\mathsf{A}}, \bar{\mathcal{K}}^*_\wedge, \mathcal{K}_0) \leftarrow \mathsf{CS.Setup}^*(1^\lambda, m, \ell_0, \ell_1, \rho)$

Players: B.
Functionality: $(\mathcal{I}_x, \mathcal{I}_y, [\![\mathcal{R}]\!]^{\mathsf{A}}, \bar{\mathcal{K}}^*_\wedge, \mathcal{K}_0) \leftarrow \mathsf{CS.Setup}^*(1^\lambda, m, \ell_0, \ell_1, \rho)$.
Input: A security parameter $\lambda, \ell, \ell_1, \rho \in \mathbb{Z}^+$.
Output: $(\mathcal{I}_x, \mathcal{I}_y, [\![\mathcal{R}]\!]^{\mathsf{A}}, \bar{\mathcal{K}}^*_\wedge, \mathcal{K}_0)$.

1: $\mathcal{K}^*_\wedge \leftarrow \mathsf{CondEval}^*.\mathsf{Setup}(\wedge, 1^\lambda, \ell_0, \ell_1, m)$
2: $\{\{C_{i,j}, \mathsf{SK}_{i,j}, [\![r_i]\!]^{\mathsf{A}}\}_{i\in[m], j\in\{0,1\}}, [\psi]^{\mathsf{B}}, [\Delta]^{\mathsf{A}}\} \leftarrow \mathcal{K}^*_\wedge$
3: $\Delta \leftarrow [\Delta]^{\mathsf{A}}_0 + [\Delta]^{\mathsf{A}}_1, \psi \leftarrow [\psi]^{\mathsf{A}}_0 \oplus [\psi]^{\mathsf{A}}_1$
4: $\boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{x})} \leftarrow\$ \mathbb{Z}^n_{2^\ell}, [\![\boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{x})}]\!]^{\mathsf{A}} \leftarrow \mathsf{SS.Share}(\{\boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{x})}, \Delta \cdot \boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{x})}\}, \mathbb{Z}_{2^{\ell_0+\ell_1}})$
5: $\boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{y})} \leftarrow\$ \mathbb{Z}^n_{2^\ell}, [\![\boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{y})}]\!]^{\mathsf{A}} \leftarrow \mathsf{SS.Share}(\{\boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{y})}, \Delta \cdot \boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{y})}\}, \mathbb{Z}_{2^{\ell_0+\ell_1}})$
6: $(\boldsymbol{x}_{\boldsymbol{in}}, \boldsymbol{y}_{\boldsymbol{in}}) \leftarrow\$ \mathbb{Z}^{2n}_{2^\ell}, \boldsymbol{u}_{\boldsymbol{in}} \leftarrow \mathsf{Prod}(\boldsymbol{x}_{\boldsymbol{in}}, \boldsymbol{y}_{\boldsymbol{in}})$
7: $\boldsymbol{v}_{\boldsymbol{in}} \leftarrow \mathsf{Prod}(\boldsymbol{x}_{\boldsymbol{in}}, \boldsymbol{x}_{\boldsymbol{in}}), \boldsymbol{w}_{\boldsymbol{in}} \leftarrow \mathsf{Prod}(\boldsymbol{y}_{\boldsymbol{in}}, \boldsymbol{y}_{\boldsymbol{in}})$
8: $(r_1, r_2) \leftarrow\$ \mathbb{Z}^2_{2^\ell}$
9: **for** $i = 1$ to m **do**
10: $\quad \alpha^{(1,i)} \leftarrow\$ \mathbb{Z}_{2^\ell}$
11: $\quad r \leftarrow\$ \mathbb{Z}_{2^\rho}$
12: $\quad \alpha^{(2,i)} \leftarrow [r_i]^{\mathsf{A}}_0 + [r_i]^{\mathsf{A}}_1$
13: $\quad \eta_i \leftarrow \mathsf{Shift}(\alpha^{(2,i)}, -\rho, \ell) + r$
14: $\quad (k^{(1,i)}_0, k^{(1,i)}_1) \leftarrow \mathsf{Gen}^{[0,2^{\ell-1}]}_\ell(1^\lambda, \alpha^{(1,i)}, 1 - \psi[i], 0, \mathbb{Z}_2)$
15: $\quad [\![\eta_i]\!]^{\mathsf{A}} \leftarrow (\mathsf{SS.Share}(\eta_i, \mathbb{Z}_{2^{\ell_0+\ell_1}}), \mathsf{SS.Share}(\Delta \cdot \eta_i, \mathbb{Z}_{2^{\ell_0+\ell_1}}))$
16: $\bar{\mathcal{K}}^*_\wedge \leftarrow \{\{C_{i,j}, \mathsf{SK}_{i,j}, [\![\eta_i]\!]^{\mathsf{A}}\}_{i\in[m], j\in\{0,1\}}, [\psi]^{\mathsf{B}}, [\Delta]^{\mathsf{A}}\}$
17: $\mathcal{R} \leftarrow \{\boldsymbol{x}_{\boldsymbol{in}}, \boldsymbol{y}_{\boldsymbol{in}}, \boldsymbol{u}_{\boldsymbol{in}}, \boldsymbol{v}_{\boldsymbol{in}}, \boldsymbol{w}_{\boldsymbol{in}}, (r_1, r_2, r_1 r_2), (\alpha^{(1,1)})^2\}$
18: $[\![\mathcal{R}]\!]^{\mathsf{A}} \leftarrow (\mathsf{SS.Share}(\mathcal{R}, \mathbb{Z}_{2^{\ell_0+\ell_1}}), \mathsf{SS.Share}(\mathcal{R} \cdot \Delta, \mathbb{Z}_{2^{\ell_0+\ell_1}}))$
19: $\mathcal{K}_0 \leftarrow \{k^{(1,i)}_0, k^{(1,i)}_1\}_{i\in[m]}$
20: $\mathcal{I}_x \leftarrow ([\![\boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{x})}]\!]^{\mathsf{A}}, [\![\boldsymbol{x}_{\boldsymbol{in}}]\!]^{\mathsf{A}}), \mathcal{I}_y \leftarrow ([\![\boldsymbol{r}_{\boldsymbol{in}}^{(\boldsymbol{y})}]\!]^{\mathsf{A}}, [\![\boldsymbol{y}_{\boldsymbol{in}}]\!]^{\mathsf{A}})$
21: Outputs $(\mathcal{I}_x, \mathcal{I}_y, [\![\mathcal{R}]\!]^{\mathsf{A}}, \bar{\mathcal{K}}^*_\wedge, \mathcal{K}_0)$

FUNCTIONALITY $\langle\!\langle x \rangle\!\rangle^{\mathsf{A}}$ or $\perp \leftarrow \mathcal{F}_{\mathsf{Input}^*}(x, [\Delta]^{\mathsf{A}}, [\![r]\!]^{\mathsf{A}}, [\![r_{\mathsf{in}}]\!]^{\mathsf{A}})$:
Players: A client C, S_0 and S_1.
Input: A secret x from C, and $[\![r]\!]^{\mathsf{A}}, [\![r_{\mathsf{in}}]\!]^{\mathsf{A}}$ from $\mathsf{S}_0, \mathsf{S}_1$.
Output: $\langle\!\langle x \rangle\!\rangle^{\mathsf{A}}$.

1: $\forall b \in \{0,1\}$, S_b sends $[r]^{\mathsf{A}}_b$ to C.
2: $\forall b \in \{0,1\}$, C returns $\delta^b_x = x + [r]^{\mathsf{A}}_0 + [r]^{\mathsf{A}}_1$ to S_b.
3: $\forall b \in \{0,1\}$, S_b computes

$$[\![x]\!]^{\mathsf{A}}_b = (b \cdot \delta^b_x - [r]^{\mathsf{A}}_b, [\Delta]^{\mathsf{A}}_b \cdot \delta_x - [\Delta \cdot r]^{\mathsf{A}}_b)$$

4: Servers run $\mathcal{F}_{\mathsf{Reveal}}([x + r_{\mathsf{in}}]^{\mathsf{A}})$ and obtain $x + r_{\mathsf{in}}$. After that, $\forall b \in \{0,1\}$ S_b computes

$$\langle\!\langle x \rangle\!\rangle^{\mathsf{A}}_b = (x + r_{\mathsf{in}}, [\![x]\!]^{\mathsf{A}}_b, [\![r_{\mathsf{in}}]\!]^{\mathsf{A}}_b)$$

5: Ouput $\langle\!\langle x \rangle\!\rangle^{\mathsf{A}}$.

Figure 4: Robust input protocol in the malicious setting.

B.0.1 Soundness. In Protocol 4, assuming an active adversary $\mathcal{A}$, the soundness of functionality CondEval.Eval$^*_{\mathsf{trunc}}$ has been proven in Theorem 5. Furthermore, due to the protection provided by the MAC scheme of other operations using authenticated arithmetic sharing, let p represent the probability that CondEval.Verify$((y)^m, [\sigma]^{\mathsf{A}}, [\psi]^{\mathsf{B}}) = 1 - \mathsf{Sign}(\cos(\boldsymbol{x}, \boldsymbol{y}) - \tau)$. Consequently, we can deduce

$$p < 2^{-\ell_1 + \log(\ell_1+1)} + 2^{1-m} + \left(\frac{3}{4}\right)^m \cdot 2^{-(\ell_0+\ell_1)}.$$

B.0.2 Security. We assert that, Protocols 3 and 4 provide a secure implementation of the circuit C when faced with an active PPT

Protocol 4 $((y)^m, [\sigma]^{\mathsf{A}}) \leftarrow \mathsf{CS.Eval}^*([\![\mathcal{R}]\!]^{\mathsf{A}}, \langle\langle \boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \langle\langle \boldsymbol{y}\rangle\rangle^{\mathsf{A}}, \bar{\mathcal{K}}^*_{\wedge}, \mathcal{K}_0, \tau, \rho)$

Players: $\mathsf{S}_0, \mathsf{S}_1$.

Functionality: $((y)^m, [\sigma]^{\mathsf{A}}) \leftarrow \mathsf{CS.Eval}^*([\![\mathcal{R}]\!]^{\mathsf{A}}, \langle\langle \boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \langle\langle \boldsymbol{y}\rangle\rangle^{\mathsf{A}}, \bar{\mathcal{K}}^*_{\wedge}, \mathcal{K}_0, \tau, \rho)$.

Input: Secret sharing $[\Delta]^{\mathsf{A}}, [\![\mathcal{R}]\!]^{\mathsf{A}}, \langle\langle \boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \langle\langle \boldsymbol{y}\rangle\rangle^{\mathsf{A}}$ and $\mathcal{K}_1$ from $\mathsf{S}_0, \mathsf{S}_1$, $\mathcal{K}_2$ from B, a public threshold $\tau \in (0, 1]$ and a truncation parameter $\rho \in \mathbb{Z}^+$.

Output: $((y)^m, [\sigma]^{\mathsf{A}})$.

1: $\boldsymbol{p} \leftarrow [\emptyset]^{m+2}$
2: $\boldsymbol{r} \leftarrow \mathcal{F}_{\mathsf{rand}}(1^\lambda, m+2, \mathbb{Z}_{2^{\ell_0+\ell_1}})$
3: $[\![u]\!]^{\mathsf{A}} \leftarrow \mathsf{SS.IP}(\langle\langle \boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \langle\langle \boldsymbol{y}\rangle\rangle^{\mathsf{A}}, [\![\boldsymbol{u_{in}}]\!]^{\mathsf{A}})$
4: $[\![v]\!]^{\mathsf{A}} \leftarrow \mathsf{SS.IP}(\langle\langle \boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \langle\langle \boldsymbol{x}\rangle\rangle^{\mathsf{A}}, [\![\boldsymbol{v_{in}}]\!]^{\mathsf{A}})$
5: $[\![w]\!]^{\mathsf{A}} \leftarrow \mathsf{SS.IP}(\langle\langle \boldsymbol{y}\rangle\rangle^{\mathsf{A}}, \langle\langle \boldsymbol{y}\rangle\rangle^{\mathsf{A}}, [\![\boldsymbol{w_{in}}]\!]^{\mathsf{A}})$
6: $\delta_v \leftarrow \mathcal{F}_{\mathsf{Reveal}}([v + r_1]^{\mathsf{A}})$ ▷ First round
7: $\delta_w \leftarrow \mathcal{F}_{\mathsf{Reveal}}([w + r_2]^{\mathsf{A}})$ ▷ First round
8: $\boldsymbol{p}[1] \leftarrow (\delta_v, [\Delta \cdot \delta_v]^{\mathsf{A}}, \boldsymbol{r}[1]), \boldsymbol{p}[2] \leftarrow (\delta_w, [\Delta \cdot \delta_w]^{\mathsf{A}}, \boldsymbol{r}[2])$
9: $\langle\langle v\rangle\rangle^{\mathsf{A}} \leftarrow (\delta_v, [\![v]\!]^{\mathsf{A}}, [\![r_1]\!]^{\mathsf{A}})$
10: $\langle\langle w\rangle\rangle^{\mathsf{A}} \leftarrow (\delta_w, [\![w]\!]^{\mathsf{A}}, [\![r_2]\!]^{\mathsf{A}})$
11: $\{\{C_{i,j}, \mathsf{SK}_{i,j}, [\![r_i]\!]^{\mathsf{A}}\}_{i\in[m], j\in\{0,1\}}, [\psi]^{\mathsf{B}}, [\Delta]^{\mathsf{A}}\} \leftarrow \mathcal{K}^*_{\wedge}$
12: $\{k_0^{(1,i)}, k_1^{(1,i)}\}_{i\in\{0,1\}} \leftarrow \mathcal{K}_0$
13: **for** $i = 1$ to m **do**
14: $\quad [\![\delta_{u^{(i)}}]\!]^{\mathsf{A}} \leftarrow [\![u]\!]^{\mathsf{A}} + [\![\alpha^{(1,i)}]\!]^{\mathsf{A}}$
15: $\quad \delta_{u^{(i)}} \leftarrow \mathcal{F}_{\mathsf{Reveal}}([\delta_{u^{(i)}}]^{\mathsf{A}})$ ▷ First round
16: $\quad$ **for** $b = 0$ to 1 **do**
17: $\quad\quad [s_i]^{\mathsf{B}}_b \leftarrow \mathsf{Eval}^{[0,2^{\ell-1}]}_{\ell}(k_b^{(1,i)}, b, \delta_{u^{(i)}})$
18: $\quad \boldsymbol{p}[2+i] \leftarrow (\delta_{u^{(i)}}, [\Delta \cdot \delta_{u^{(i)}}]^{\mathsf{A}}, \boldsymbol{r}[2+i])$
19: $(s)^m \leftarrow \{[s_1]^{\mathsf{B}}, \cdots, [s_m]^{\mathsf{B}}\}$
20: $T \leftarrow \mathsf{Shift}(\frac{1}{\tau^2}, -\rho, \ell)$
21: $[\![z]\!]^{\mathsf{A}} \leftarrow T \cdot \mathsf{SS.MUL}(\langle\langle u^{(1)}\rangle\rangle^{\mathsf{A}}, \langle\langle u^{(1)}\rangle\rangle^{\mathsf{A}}, [\![(\alpha^{(1,1)})^2]\!]^{\mathsf{A}})$
22: $[\![z]\!]^{\mathsf{A}} \leftarrow [\![z]\!]^{\mathsf{A}} - 2^\rho \cdot \mathsf{SS.MUL}(\langle\langle v\rangle\rangle^{\mathsf{A}}, \langle\langle w\rangle\rangle^{\mathsf{A}}, [\![r_1 r_2]\!]^{\mathsf{A}})$
23: $((y)^m, [\sigma]^{\mathsf{A}}) \leftarrow \mathsf{CondEval.Eval}^*_{\mathsf{trunc}}(\wedge, \bar{\mathcal{K}}^*_{\wedge}, (s)^m, [\![z]\!]^{\mathsf{A}})$ ▷ Second round
24: $[\varsigma]^{\mathsf{A}} \leftarrow \mathcal{F}_{\mathsf{MacVryGen}}([\Delta]^{\mathsf{A}}, \boldsymbol{p})$
25: $[\sigma]^{\mathsf{A}} \leftarrow [\sigma]^{\mathsf{A}} + [\varsigma]^{\mathsf{A}}$
26: Outputs $((y)^m, [\sigma]^{\mathsf{A}})$.

$\mathcal{A}$ in the 2PC setting. Specifically, with the correlated randomness provided in Protocol 3, following the online evaluation of Protocol 4, $\mathcal{A}$ learns nothing about the inputs $\boldsymbol{x}$, $\boldsymbol{y}$, or $\mathsf{Sign}(\cos(\boldsymbol{x}, \boldsymbol{y}) - \tau)$.

Proof. From Theorem 4, we first exclude information leakage within $\mathsf{CondEval.Eval}^*_{\mathsf{trunc}}(\wedge, \bar{\mathcal{K}}^*_{\wedge}, (s)^m, [\![z]\!]^{\mathsf{A}})$. Still, we need to prove that the view transcript

$$\mathsf{View}_b := \{\{k_b^{(1,i)}, \delta_{u^{(i)}}\}_{i\in[m]}, \delta_v, \delta_w\}$$

of S_b for each $b \in \{0, 1\}$ does not leak any information. This is true, as $\delta_v, \delta_w \sim \mathbb{U}_N$ information theoretically hide the associated intermediate computation results; Nevertheless, for $\{(k_b^{(1,i)}, \delta_{u^{(i)}})\}_{i\in[m]}$ which are correlated, we argue that $\mathcal{A}$ has only the view $k_b^{(1,i)}$ which is pseudo-random (computationally indistinguishable from a real random key) as proved in Fig. 1 in [10]; Thus, hiding the information of $\alpha^{(1,i)}$ contained in $(k_0^{(1,i)}, k_1^{(1,i)})$ from $\mathcal{A}$. □

B.1 Truncating δ_x

It is worth to note that while executing CS.Eval, a variant version of CondEval.Eval which we denote as $\mathsf{CondEval.Eval}_{\mathsf{trunc}}$ is incorporated whereby δ_x is truncated additionally. Namely, in $\mathsf{CondEval.Eval}_{\mathsf{trunc}}$ it performs

$$\delta_x \leftarrow \mathsf{Shift}(\delta_x, 4\rho, \ell)$$

while the remaining steps in CondEval.Eval remain unchanged. Previously, when dealing with the truncation of $[x]^{\mathsf{A}}$, $[\eta]^{\mathsf{A}}$ was generated and corresponds to the secret sharing result of the sum of the shifting of r to the left by ρ and a random value $v \in \mathbb{Z}_{2^\rho}$; In the above equation, the fractional part of δ_x is truncated by 4ρ bits due to the accumulation of the three multiplications over secret sharing and one multiplication over a public scalar value ($T, 2^\rho$ resp. in line 12,14). Such a truncation does not imply any additional communication round and is performed during the second communication round. Furthermore, as it will be shown in section 6.3, such a truncation has almost no impact on the actual accuracy of the protocol.

C CONDEVAL OVER THE OR GATE

In this section, we demonstrate a direct method of computing $s \vee f(x)$ instead of computing $\neg(\neg s \wedge \neg f(x))$. The pipeline is similar to that of computing a logical-and gate, including the CondEval.Setup and CondEval.Eval functions. In Table 6 we provide the details of $\mathsf{CondEval.Setup}(\vee, 1^\lambda, \ell)$ for the semi-honest setting, which only differs from the construction of $\mathsf{CondEval.Setup}((\wedge, 1^\lambda, \ell))$ by appending an additional random secret bit ω. We construct $\mathsf{CondEval.Eval}(\vee, \mathcal{K}\vee, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$ from $\mathsf{CondEval.Eval}(\wedge, \mathcal{K}\wedge, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$, which follows all the steps except tweaking the evaluation step for each $b \in \{0, 1\}$ where *i.e.,* we perform

$$[y_i]^{\mathsf{B}}_b \leftarrow \mathsf{Eval}^f(k^{(b)}, \mathsf{id}^{(b)}, \delta_x) \oplus \omega^{(b)}.$$

Theorem 6. *Correctness.* If S_0 and S_1 honestly follow $\mathsf{CondEval.Eval}(\vee, \mathcal{K}_\vee, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$, which uses the correlated FSS key pairs prepared in Table 6, then $\mathsf{CondEval.Eval}(\vee, \mathcal{K}_\vee, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$ outputs $[y]^{\mathsf{B}} = [s \vee f(x)]^{\mathsf{B}}$.

Proof. If $s = 0$, then $([s]^{\mathsf{B}}_0, [s]^{\mathsf{B}}_1)$ is equal to either $(0, 0)$ or $(1, 1)$, which implies

$$y = (\mathsf{Eval}^f(k_t, t, \delta_x) \oplus \omega) \oplus (\mathsf{Eval}^f(k_{1-t}, 1-t, \delta_x) \oplus \omega)$$

or

$$y = (\mathsf{Eval}^f(k_{1-t}, 1-t, \delta_x) \oplus (1-\omega)) \oplus (\mathsf{Eval}^f(k_t, t, \delta_x) \oplus (1-\omega))$$

In either case, $y = f(x)$, as desired. On the other hand, if $s = 1$, then $([s]^{\mathsf{B}}_0, [s]^{\mathsf{B}}_1)$ equals either $(0, 1)$ or $(1, 0)$, which implies

$$y = (\mathsf{Eval}^f(k_{1-t}, 1-t, \delta_x) \oplus (1-\omega)) \oplus (\mathsf{Eval}^f(k_{1-t}, 1-t, \delta_x) \oplus \omega)$$

or

$$y = (\mathsf{Eval}^f(k_t, t, \delta_x) \oplus \omega) \oplus (\mathsf{Eval}^f(k_t, t, \delta_x) \oplus (1-\omega))$$

In either case, $y = 1$, as desired. Thus, $y = s \vee f(x)$. □

KeyGen	$r \leftarrow\!\!\$\, \mathbb{Z}_{2^\ell}, [r]^{\mathsf{A}} \leftarrow \mathsf{SS.share}(r, \mathbb{Z}_{2^\ell});$ $(k_0, k_1) \leftarrow \mathsf{Gen}^f(1^\lambda, r, 1, 0, \mathbb{Z}_2)(\lvert k_0\rvert = \lvert k_1\rvert = K);$ $t \leftarrow\!\!\$\, \mathbb{Z}_2; \pi_0 \leftarrow\!\!\$\, \mathbb{Z}_2, \pi_1 \leftarrow\!\!\$\, \mathbb{Z}_2; \omega \leftarrow\!\!\$\, \mathbb{Z}_2; L := K + 2$ $\mathsf{sk}_0^{(0)} \| \mathsf{sk}_1^{(0)} \| \mathsf{sk}_0^{(1)} \| \mathsf{sk}_1^{(1)} \leftarrow\!\!\$\, \{0,1\}^{4L},$ $\lvert \mathsf{sk}_0^{(0)}\rvert = \lvert \mathsf{sk}_1^{(0)}\rvert = \lvert \mathsf{sk}_0^{(1)}\rvert = \lvert \mathsf{sk}_1^{(1)}\rvert = L.$
KeyEnc	$m_0 = \{\mathsf{sk}_0^{(0)} \oplus (k_t \| t \| \omega), \mathsf{sk}_1^{(0)} \oplus (k_{1-t} \| (1-t) \| (1-\omega))\}$ $C_0 = \mathcal{F}_{\mathsf{Permu}}(\pi_0, m_0), \mathsf{SK}_0 = \{(\mathsf{sk}_0^{(1)}, \mathsf{sk}_1^{(1)}), \pi_1\}$ $m_1 = \{\mathsf{sk}_0^{(1)} \oplus (k_{1-t} \| (1-t) \| \omega), \mathsf{sk}_1^{(1)} \oplus (k_t \| t \| (1-\omega))\}$ $C_1 = \mathcal{F}_{\mathsf{Permu}}(\pi_1, m_1), \mathsf{SK}_1 = \{(\mathsf{sk}_0^{(0)}, \mathsf{sk}_1^{(0)}), \pi_0\}$
Ouputs $\mathcal{K}_\vee = \{C_i, \mathsf{SK}_i, [r]_i^{\mathsf{A}}\}_{i \in \{0,1\}}$	

Table 6: The construction of $\mathcal{K}_\vee \leftarrow \mathsf{CondEval.Setup}(\vee, 1^\lambda, \mathbb{Z}_{2^\ell})$ in the semi-honest setting, where λ is the security parameter used in generating FSS keys, $\mathbb{Z}_{2^\ell}$ defines the domain of the secret sharing.

Naturally, we have same security guarantee for CondEval.Eval $(\vee, \mathcal{K}_\vee, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$ following theorem 2. The detailed construction in the case of malicious setting follows similarly with the protocols in section 4.3 and we omit the detials here.