Nomadic: Normalising MAliciously-Secure DIstance with Cosine Similarity for Two-Party Biometric Authentication

Anonymous Author(s)

ABSTRACT

Computing the distance between two non-normalized vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, represented by $\Delta(\boldsymbol{x}, \boldsymbol{y})$ and comparing it to a predefined public threshold τ is an essential functionality used in privacy-sensitive applications such as biometric authentication, identification, machine learning algorithms (*e.g.,* linear regression, k-nearest neighbors, etc.), and typo-tolerant password-based authentication. Tackling a widely used distance metric, Nomadic studies the privacy-preserving evaluation of cosine similarity in a two-party (2PC) distributed setting. We illustrate this setting in a scenario where a client uses biometrics to authenticate to a service provider, outsourcing the distance calculation to two computing servers. In this setting, we propose two novel 2PC protocols to evaluate the normalising cosine similarity between non-normalised two vectors 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 function f yielding a binary result with a subsequent binary gate. Overall, our protocols outperform all prior works, requiring only two communication rounds under a strong threat model that also deals with malicious inputs via normalisation. We evaluate our protocols in the setting of biometric authentication using voice, and the obtained results reveal a notable efficiency improvement compared to existing state-of-the-art works.

KEYWORDS

privacy-preserving protocols, malicious security, function secret sharing, cosine similarity

ACM Reference Format:
Anonymous Author(s). 2018. Nomadic: Normalising MAliciously-Secure DIstance with Cosine Similarity for Two-Party Biometric Authentication. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 17 pages. https://doi.org/XXXXXXX.XXXXXXX

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 privacy-sensitive applications including biometric authentication and identification, machine learning (*e.g.,* linear regression, matrix multiplication) as well as typo-tolerant (fuzzy) password based authentication. However, in the absence of trusted parties it is challenging to evaluate thresholded distance metrics over sensitive data. Although privacy-preserving computations of linear distance metrics have been widely covered in the literature, protecting non-linear computations (e.g., comparison) are not easy to compute efficiently.

In this paper, we focus on protecting the computation of *cosine similarity* between two vectors, one of the most commonly employed distance metrics, subsequently comparing the similarity score to a public threshold τ. We present two novel privacy-preserving protocols to realize this thresholded distance functionality with high efficiency in a two-party setting (2PC) for outsourced computation, achieving passive and active security respectively.

As an ideal application of our protocols due to the sensitive nature of the data, biometric authentication consists of verifying a the identity of a user by comparing a fresh compact representation of the biometric trait with a pre-existing (claimed) reference provided during an 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 hold 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 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].

To showcase Nomadic, we implement and evaluate our protocols in a biometric authentication use-case with voice biometrics. 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.

Related Work. Several techniques have been proposed in the literature to address the privacy challenges for the computation of distance metrics (such as Hamming distance [42], cosine similarity) followed by comparison with a public threshold, based on advanced cryptographic primitives like Fully Homomorphic Encryption (FHE) [27, 28], Multi-Party Computation (MPC) [29, 55, 63] and Functional Encryption (FE) [1, 10]. FHE relies on performing an arbitrary number of arithmetic operations (*i.e.,* additions and multiplications) between ciphertexts. MPC-based techniques involve distributing data among non-colluding parties to collectively compute a desired function over the private data, covering two main security models: semi-honest or malicious depending on whether the involved parties are assumed to follow the protocol or may

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

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

Table 1: Number of communication rounds required by recent MPC protocols to compute a Sign, a pre-normalised thresholded inner product $\mathcal{F}_{\mathsf{Sign}(\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})-\tau)}$ and a full thresholded cosine similarity $\mathcal{F}_{\mathsf{ScaledCosAuth}}$. Security covers one semi-honest ○ / malicious client-only ◑ / malicious ● (+ for fairness, $\star$ for robustness) corruption. Primitives include Secret Sharing (SS), unbalanced SS (u-SS), Replicated SS (RSS), optimized SS (o-SS), (Correlated) OT extension (COTe), Garbled Circuits (GC), Binary Adder circuits (BA), Distributed Comparison Function (DCF), Interval Containment (IC), Zero-Knowledge Proofs (ZKP).

Work	Type	Security	Protocol for IP	Protocol for Sign	$\mathcal{F}_{\mathsf{Sign}}$	$\mathcal{F}_{\mathsf{Sign}(\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})-\tau)}$	$\mathcal{F}_{\mathsf{ScaledCosAuth}}$
Blaze [49]	3PC	●$^{+}$	Π_{dotp}, u-3SS	Π_{bitext}, GC	$\log_2(l) + 1$	$\log_2(l) + 2$	$\log_2(l) + 4$
Falcon [62]	3PC	●	SS	Π_{WA}, Arith. curcuit	$\log_2(l) + 3$	$\log_2(l) + 4$	$\log_2(l) + 6$
SWIFT [38]	3PC	●*	Π_{dotp}, 2SS & u-2SS	Π_{bitext}, GC	$\log_2(l) + 1$	$\log_2(l) + 2$	$\log_2(l) + 4$
Trident [17]	4PC	●	Π_{Dotp}, u-4SS	[49]	$\log_2(l) + 1$	$\log_2(l) + 2$	$\log_2(l) + 4$
Flash [15]	4PC	●*	Π_{dp}, 2× u-2SS	Π_{msb}, BA of [44]	$\log_2(l)$	$\log_2(l) + 1$	$\log_2(l) + 3$
SWIFT [38]	4PC	●*	Π_{dotp4}, 3PC & masks	Π_{bitext4}, GC	$\log_2(l)$	$\log_2(l) + 1$	$\log_2(l) + 3$
Fantastic Four [21]	4PC	●*	Mult, RSS	Share splitting, BA	$\log_2(l)$	$\log_2(l) + 1$	$\log_2(l) + 3$
Tetrad [39]	4PC	●*	Π_{dotp}, RSS & masks	Π_{bitext}, BA of [48]	$\log_4(l) + 1$	$\log_4(l) + 2$	$\log_4(l) + 4$
ABY [24]	2PC	○	SS	GC	$\log_2(l) + 2$	$\log_2(l) + 3$	$\log_2(l) + 5$
ABY2.0 [48]	2PC	○	o-SS	BitExtraction, mi-BA	$\log_4(l) + 1$	$\log_4(l) + 2$	$\log_4(l) + 4$
Cryptflow [40]	2PC	○	SS	dReLU, Arith. circuit	$\log_2(l) + 4$	$\log_2(l) + 5$	$\log_2(l) + 7$
Cryptflow2 [52]	2PC	○	Π_{Umult}, COTe	Π_{Mill}, OT ext.	$\log_2(l)$	$\log_2(l) + 1$	$\log_2(l) + 3$
Muse [41]	2PC	◑	Linear layer, o-SS	Non-linear layer, GC	$\log_2(l)$	$\log_2(l) + 1$	$\log_2(l) + 3$
SIMC [16]	2PC	◑	Π_{Lin}, HE & ZKP	$\mathcal{F}_{\mathsf{Non-lin}}$, GC	$\log_2(l)$	$\log_2(l) + 2$	$\log_2(l) + 6$
AriaNN [53]	2PC	○	SS	DCF gate of [13], FSS	1	2	4
Boyle et. al. [11]	2PC	○	SS	IC gate, FSS	1	2	4
Llama [30]	2PC	○	$\mathcal{G}_{\mathsf{smult}}$, FSS	FSS IC gate of [11]	1	2	4
Funshade [33]	2PC	○	o-SS	IC gate of [11], FSS	1	1	3
Ours	2PC	●	**o-SS & CondEval**	CondEval	**1**	**2**	**2**

deviate from it. FE are public-key encryption schemes that enables authorized parties to evaluate specific linear functions (*e.g.,* inner products [4, 23, 58]) 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 [31, 51, 56, 60], 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 in the FHE setting using computationally intensive polynomial approximations [18, 34]. In contrast, many MPC-based solutions and frameworks realize privacy-preserving comparison in various settings [36, 40, 52, 62]. Mixed protocols (*i.e.,* protocols that combine the use of arithmetic circuits with homomorphic encryption, garbled circuits and/or Boolean circuits) compute scalar products using arithmetic protocols and switch from arithmetic to binary/garbled circuits in order to compute comparisons [15, 17, 24, 38, 44, 48, 49]. However, the vast majority of these frameworks either requires an honest majority (3PC, 4PC) to achieve active security (against one malicious corruption) or settles with security against a semi-honest or a one-sided malicious corruption in the 2PC setting. Veugen *et al.* [61] proposed a 2PC framework that improves comparison computation in the SPDZ protocol [22] which is a mixed protocol that takes into account the presence of malicious parties. However, these two solutions incurr in intensive communication costs (size and/or number of rounds). Recently, Function Secret Sharing (FSS)-based protocols have been proposed to efficiently compute the comparison to a public threshold operation [11, 13, 30, 33, 54]. Nevertheless, the mentioned protocols focus mainly on the semi-honest model and barely address security against malicious adversaries.

Privacy-preserving biometrics: Boddeti [9] presented a privacy-preserving solution for facial recognition based on FHE. The face similarity score is computed in the encrypted domain using scalar multiplications and additions. [32] extended FHE-based face identification to also compute the secure comparison in the FHE domain. In [45], the authors made use of an FHE scheme to compute the Hamming distance for iris authentication. The work proposed in [50] 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.* [35] presented a fingerprint authentication system using FHE to compute the Square Euclidean Distance between two encrypted vectors, also performing the threshold comparison in the encrypted domain. Despite the potential shown by FHE in these works, its utilization in real-world applications is still restricted by the significant computational overheads.

Alternatively, MPC-based techniques have been successfully used to protect sensitive data like face [5, 33], iris [5, 26], and voice [47, 59]. Barni *et al.* [5] 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 [26] 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.* [47] provided the first computationally feasible privacy-preserving ASV system with cohort score normalization based on 2PC. However, this work only provides security against a semi-honest adversary. In [59], 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 semi-honest.

Our Contributions. We propose a novel two-party protocol for secure computation of cosine similarity followed by comparison to a threshold, leveraging secure 2PC and FSS primitives for both the semi-honest and the malicious setting. As shown in Table 1, our protocol outperforms prior work by requiring just two rounds of communication in the online phase. In addition, and contrary to prior work that assumes honest normalisation performed by the client, we allow the client to submit any fresh template and we incorporate a mechanism to check that the secret sharing of the fresh (as well as the reference) template was performed correctly. Our main contributions are:

- A 2PC protocol for thresholded cosine similarity computation between two vectors relying on authenticated 2PC secret sharing as well as FSS. We develop two variants: one version for the *semi-honest* setting and one for the *malicious* setting. As a distinguishing feature, the protocol (and not the input holder) carries out the normalisation of the input vectors, guaranteeing their well-formedness.
- A new building block, CondEval, 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, proven secure under both the semi-honest and malicious model, may be of independent interest for general 2PC secure computation.
- A rigorous security analysis of both protocols of Nomadic as well as the CondEval primitive.
- An evaluation of the proposed protocols employing voice biometrics as a use case. Comparing to existing state-of-the-art (SOTA) work, our experiments demonstrate the efficiency improvement of our protocols in both the semi-honest and malicious setting thanks to the reduction of one communication round due to CondEval.

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

2.0 Notation

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

2.1 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 his 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).

2.2 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

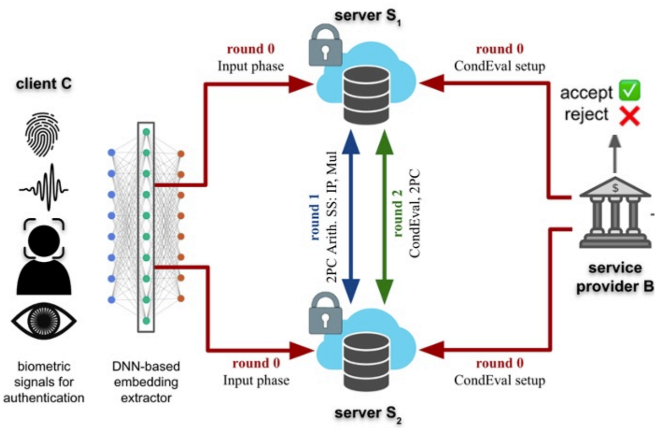


Figure 1: Privacy-preserving biometric authentication system based on cosine similarity in the 2PC setting.

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.

2.3 Cosine Similarity for Authentication

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 between them 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 designing a privacy-preserving protocol for this functionality, we tweak $\mathcal{F}_{\mathsf{CosAuth}}$ to avoid expensive non-linear operations (division, square-root), obtaining an equivalent MPC-friendly circuit $c = C(\boldsymbol{x}, \boldsymbol{y}, \tau)$ as follows:

$$\begin{aligned} c_1 &= \mathsf{Sign}([\mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})]^{\mathsf{A}}) \\ c_2 &= \mathsf{Sign}([1/\tau^2 \cdot \mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})^2 - \mathsf{IP}(\boldsymbol{x}, \boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y}, \boldsymbol{y})]^{\mathsf{A}}) \\ c &= \begin{cases} c_1 \wedge c_2 & \text{if } \tau > 0 \\ \quad c_1 & \text{if } \tau = 0 \\ c_1 \vee c_2 & \text{if } \tau > 0 \end{cases} \end{aligned} \tag{3}$$

One can easily verify the equivalence $C \equiv \mathcal{F}_{\mathsf{CosAuth}}$. For the remnant of this paper, *w.l.o.g.,* we assume $\tau > 0$, and define the following functionality accordingly:

$$\begin{aligned} \mathcal{F}_{\mathsf{ScaledCosAuth}}(\boldsymbol{x}, \boldsymbol{y}, \tau) = \mathsf{Sign}(\mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})) \wedge \\ \mathsf{Sign}(1/\tau^2 \cdot \mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})^2 - \mathsf{IP}(\boldsymbol{x}, \boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y}, \boldsymbol{y})) \end{aligned} \tag{4}$$

3 PRELIMINARIES

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 [6] and replicated SS [3], or Boolean SS like GMW [43]; or *(ii)* Garbled Circuits (CG) like Yao's GCs [64], BMR [7].

Optimized Additive SS. we make use of an optimized additive SS with function dependent pre-processing as our building block [8]. In the setup phase, correlated random offset shares are generated for input wires and output wires of each gate of the arithmetic circuit. These shares are input-independent 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 [7]. 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 $\{\langle x\rangle_b^{\mathsf{A}} = (\delta_x, [r_x]_b^{\mathsf{A}}), \langle y\rangle_b^{\mathsf{A}} = (\delta_y, [r_y]_b^{\mathsf{A}}), [r_x r_y]_b^{\mathsf{A}}\}$ in which $\delta_x = x + r_x, \delta_y = y + r_y$. Then, S_b computes:

$$[z]_b^{\mathsf{A}} = b \cdot \delta_x\delta_y - \delta_x[r_x]_b^{\mathsf{A}} - \delta_y[r_y]_b^{\mathsf{A}} + [r_x r_y]_b^{\mathsf{A}}. \tag{5}$$

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; $\mathsf{SS.MUL}(\langle x\rangle^{\mathsf{A}}, \langle y\rangle^{\mathsf{A}}, [r_x r_y]^{\mathsf{A}})$ as the multiplication process shown in Eq. 5, $\mathsf{SS.IP}(\langle \boldsymbol{x}\rangle^{\mathsf{A}}, \langle \boldsymbol{y}\rangle^{\mathsf{A}}, [\boldsymbol{u}]^{\mathsf{A}})$ as the inner product of two vectors $\boldsymbol{x}, \boldsymbol{y}$ where $\boldsymbol{u}$ is the correlated product of randomness.

SPDZ2k protocol. The SPDZ2k protocol [20] is a MPC protocol works in the context of the dishonest majority setting, or assuming one malicious server in the 2PC setting. In this work, we apply it in a 2PC setting. It performs 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. Through out this paper where we deal with 2PC setting, we denote an authenticated arithmetic 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 by using a final MAC verification algorithm, as shown in Fig.6 of [20]. In the case when the verification fails, malicious behaviors are detected and the parties abort the protocol.

When working with authenticated optimized secret sharing in the malicious setting, we use same notation $\mathsf{SS.MUL}(\langle\!\langle x\rangle\!\rangle^{\mathsf{A}}, \langle\!\langle y\rangle\!\rangle^{\mathsf{A}}, [\![r_x r_y]\!]^{\mathsf{A}})$ 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 6, 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 [44]. 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.* [12] as a cryptographic primitive that secretly shares a function f. Informally, in a two party setting with a function $f : \mathbb{D} \to \mathbb{R}$, *FSS* 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.* [13]. 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 trusted dealer 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 [12, 14].

In constructing all our protocols, we make use the interval containment function secret sharing (IC-FSS) ([11], Section 4.1) as our secure comparison primitive, as the same as in Funshade [33].

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 TECHNICAL OVERVIEW

To realize $\mathcal{F}_{\mathsf{ScaledCosAuth}}$ with passive security, we study its required components and immediately reach two preliminary protocols, Π_{NaiveSH} and Π_{OptimSH}. Further enhancements in this paper will later lead to our novel Nomadic protocols. The online evaluation steps for all the protocols described in this section are outlined in Table 2, with associated online costs detailed in Table 3.

Table 2: Online steps of protocols for $\mathcal{F}_{\mathsf{ScaledCosAuth}}$.

	Round-1	Round-2	Round-3	Round-4
Π_{NaiveSH}	$[\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})]^{\mathsf{A}}$ $[\mathsf{IP}(\boldsymbol{x},\boldsymbol{x})]^{\mathsf{A}}$ $[\mathsf{IP}(\boldsymbol{y},\boldsymbol{y})]^{\mathsf{A}}$	$[c_1]^{\mathsf{B}} \leftarrow \mathsf{Sign}([\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})]^{\mathsf{A}})$ $[u_1]^{\mathsf{A}} \leftarrow [\mathsf{IP}(\boldsymbol{x},\boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y},\boldsymbol{y})]^{\mathsf{A}}$ $[u_2]^{\mathsf{A}} \leftarrow [\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})^2]^{\mathsf{A}}$ $[u]^{\mathsf{A}} \leftarrow [u_1 - 1/\tau^2 \cdot u_2]^{\mathsf{A}}$	$[c_2]^{\mathsf{B}} \leftarrow \mathsf{Sign}([u]^{\mathsf{A}})$	$[c]^{\mathsf{B}} \leftarrow [c_1 \wedge c_2]^{\mathsf{B}}$
Π_{OptimSH}	$[\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})]^{\mathsf{A}}$ $[\mathsf{IP}(\boldsymbol{x},\boldsymbol{x})]^{\mathsf{A}}$ $[\mathsf{IP}(\boldsymbol{y},\boldsymbol{y})]^{\mathsf{A}}$ $[c_1]^{\mathsf{B}} \leftarrow \mathsf{Sign}([\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})]^{\mathsf{A}})$	$[u_1]^{\mathsf{A}} \leftarrow [\mathsf{IP}(\boldsymbol{x},\boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y},\boldsymbol{y})]^{\mathsf{A}}$ $[u_2]^{\mathsf{A}} \leftarrow [\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})^2]^{\mathsf{A}}$ $[u]^{\mathsf{A}} \leftarrow [u_1 - 1/\tau^2 \cdot u_2]^{\mathsf{A}}$ $[c_2]^{\mathsf{B}} \leftarrow \mathsf{Sign}([u]^{\mathsf{A}})$	$[c]^{\mathsf{B}} \leftarrow [c_1 \wedge c_2]^{\mathsf{B}}$	-
$\Pi_{\mathsf{NomadicSH}}$	$[\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})]^{\mathsf{A}}$ $[\mathsf{IP}(\boldsymbol{x},\boldsymbol{x})]^{\mathsf{A}}$ $[\mathsf{IP}(\boldsymbol{y},\boldsymbol{y})]^{\mathsf{A}}$ $[c_1]^{\mathsf{B}} \leftarrow \mathsf{Sign}([\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})]^{\mathsf{A}})$	$[u_1]^{\mathsf{A}} \leftarrow [\mathsf{IP}(\boldsymbol{x},\boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y},\boldsymbol{y})]^{\mathsf{A}}$ $[u_2]^{\mathsf{A}} \leftarrow [\mathsf{IP}(\boldsymbol{x},\boldsymbol{y})^2]^{\mathsf{A}}$ $[u]^{\mathsf{A}} \leftarrow [u_1 - 1/\tau^2 \cdot u_2]^{\mathsf{A}}$ $[c]^{\mathsf{B}} \leftarrow \mathsf{CondEval}([c_1]^{\mathsf{B}}, \mathsf{Sign}([\mu]^{\mathsf{A}}))$	-	-

Table 3: Comparative overview of online costs for Π_{NaiveSH}, Π_{OptimSH}, Π_{Mal1}, $\Pi_{\mathsf{NomadicSH}}$ and Π_{NomadicM}. Here "Comm." refers to the total communication volume among two servers, n is the length of the input vector, ℓ the length of the operational ring, L the communication volume for a single evaluation of $\mathsf{CondEval}$, and κ the amount of parallel $\mathsf{CondEval}$ executions in Π_{NomadicM}.

	Model	Primitive	Comm.	#Rounds
Π_{NaiveSH}	SH	SS + FSS	$6n\ell + 4n\ell + 2\ell + 2$	4
Π_{OptimSH}	SH	Optimized SS + FSS	$6\ell + 2$	3
$\Pi_{\mathsf{NomadicSH}}$	SH	Optimized SS + CondEval	$7\ell + L$	2
Π_{NaiveM}	M	Authenticated (Optimized SS + FSS)	$6\ell + 2$	3
Π_{NomadicM}	M	Authenticated (Optimized SS + CondEval)	$6\ell + \kappa(\ell + L)$	2

As a first approach [11, 30, 53], Π_{NaiveSH} employs additive SS and FSS. This protocol requires four communication rounds.

Next, we follow the guidelines of Funshade [33] to get Π_{OptimSH}, introducing optimized secret sharing paired with FSS comparison gates. By its online phase, Π_{OptimSH} necessitates pre-prepared, circuit-dependent correlated randomness, generated offline by a dealer and distributed to two computing servers. As highlighted in Funshade [33], this approach improves communication efficiency over Π_{NaiveSH} in terms of rounds and data volume. For instance, computing $[c_1]^{\mathsf{B}}$ of $\mathcal{F}_{\mathsf{ScaledCosAuth}}$ in Π_{OptimSH} requires only one round and a single group element exchange, compared to Π_{NaiveSH} (two rounds and $2n$ group elements). Consequently, Π_{OptimSH} reduces to three communication rounds: the first for c_1, the second for c_2, and the third for the final output $c = c_1 \wedge c_2$.

Further refining Π_{OptimSH}, we decrease communication rounds to two using a novel component, $\mathsf{CondEval}$. This new primitive introduces a conditional evaluation gate: $(c, x, f, \circ) \rightarrow c \circ f(x)$, traditionally executed in two rounds but here accomplished in one. It is designed for computing the composition of a function $f(x)$ with a logical AND ($\wedge$) or OR ($\vee$) operation, integrating optimized secret sharing and novel FSS key generation steps. This innovation enables efficient computation of $c = c_1 \wedge c_2$ in the second round, thus saving a communication round.

We specify and prove the security of the $\mathsf{CondEval}$ primitive in the semi-honest setting, incorporating it to the full protocol in $\Pi_{\mathsf{NomadicSH}}$. As a result, $\Pi_{\mathsf{NomadicSH}}$ encompasses the following four distinct phases:

(1) **Setup**: Generating correlated randomness for $\mathcal{F}_{\mathsf{ScaledCosAuth}}$.
(2) **Input**: Both the client and entity B independently input their secret vectors, **x** and **y**, respectively.
(3) **Evaluation**: Conducted by two servers, this phase includes:
    - *First round*: Computation of a Boolean sharing $[c_1]^{\mathsf{B}}$.
    - *Second round*: Computation of the final Boolean sharing $[c]^{\mathsf{B}} = [c_1]^{\mathsf{B}} \circ f(x)$. In a malicious setting, this includes generating $[c]^{\mathsf{B}}$ and a proof $[\sigma]^{\mathsf{A}}$ for verification.
(4) **Reveal**: The service provider B constructs the value of c in clear-text. In a malicious context, B also validates $[c]^{\mathsf{A}}$ and assesses the proof $[\sigma]^{\mathsf{A}}$.

Additionally, we extend the construction and security proof of $\mathsf{CondEval}$ to the malicious setting. To provide active security in the FSS gates we resort to the parallel execution of multiple instances for the conditional evaluation gate 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. For a fully maliciously-secure protocol Π_{NomadicM}, we employ homomorphic MACs when using additive secret shares (as in the SPDZ2k protocol [20] but we use a simplified variant). The full maliciously-secure protocol Π_{NomadicM} follows a similar online evaluation pipeline to what in $\Pi_{\mathsf{NomadicSH}}$, thus, we don't repeat it in Tab. 2.

5 REALIZING CONDEVAL IN 2PC

We introduce $\mathsf{CondEval}$ as key element to optimize our thresholded cosine similarity protocols. Following circuit C (Eq. 3), the evaluation of a biometric matching protocol splits into *(i)* computing shares of boolean values c_1 and c_2 (see Equation 3) and *(ii)* computing $c_1 \wedge c_2$ or $c_1 \vee c_2$ depending on the value of τ. FSS can be used to efficiently compute the Sign for c_1 and c_2. Computing the last boolean gate ($\wedge$ or $\vee$) in secret shares would require at least one communication round. CondEval integrates this boolean gate into the FSS evaluation of c_2, saving up one communication round in the pre-processing model. We formalize this functionality as:

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

where we compute a two-input boolean gate with a bit s and a function $f(x)$ (instantiated with FSS). We would then apply CondEval to $\mathcal{F}_{\mathsf{ScaledCosAuth}}$ by setting $s = c_1$ and $f(x) = c_2 = \mathsf{Sign}([1/\tau^2 \cdot \mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})^2 - \mathsf{IP}(\boldsymbol{x}, \boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y}, \boldsymbol{y})]^{\mathsf{A}})$.

In the reminder of this section, and *w.l.o.g.*, we consider the $\wedge$ operation and we describe how CondEval computes $s \wedge f(x)$ in a symmetric 2PC setting in both the semi-honest and malicious models[2]. It is worth noting that CondEval can be used as an independent building block to compute the boolean composition of a bit s and a function $f(x)$ evaluated with FSS for 2PC protocols.

5.1 High-level Overview

CondEval seeks to evaluate $s \wedge f(x)$ using FSS with two computing servers S_0 and S_1. The inputs to this protocol are mainly secret shares of a bit s and an input x know to both parties. To avoid the additional communication round that the $\wedge$ gate would require, we aim to merge it with the FSS evaluation of x. Indeed, from Table 4 (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 cases where 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 would reconstruct 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 4. Each server must retrieve the correct FSS key according to s, without leaking any information neither about s, nor the other FSS key.

Table 4: Truth table of $[s \wedge f(x)]^{\mathsf{B}}$.

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	$\mathsf{Eval}^f(k_1, 1, \delta_x)$	$\mathsf{Eval}^f(k_0, 0, \delta_x)$	

Such a protocol can be designed if the two FSS keys k_0 and k_1 generated by the bank[3] were randomly mapped to the values of s (k_{1-s} if $s = 0$, k_s if $s = 1$). 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 encrypted FSS keys 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.

Now, we need 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 $b \in \{0, 1\}$):

- since Eval^f also takes as input the index of the FSS key (0 or 1), this index 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 $[s]_b^{\mathsf{B}}$, a random permutation is applied to (k_t, k_{1-t});
- to protect the unused FSS key, the permuted keys are one-time-pad encrypted before their storage and only one decryption key will be sent from S_b to S_{1-b} in the key decryption step.

Overall, CondEval is thus specified with four algorithms:

- 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 running CondEval.Eval to obtain the share of the output $s \wedge f(x)$.

Note that, while we only demonstrate the construction of CondEval for $s \wedge f(x)$, 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. We refer the avid reader to Appendix D for the details.

5.2 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 5. 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

[2] Details on the $\vee$ operation are provided in Appendix D.

[3] The entity in charge of generating pre-processing material (Fig. 1).

Table 5: 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.

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

round of communication, δ_x is revealed and one clear-text FSS key is obtained from both servers with *KeyDec* being performed, and finally it outputs $[y]^\mathsf{B}$.

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 5 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: $[s \wedge f(x)]^\mathsf{B}$.

```
$\delta_x \leftarrow \mathcal{F}_{\mathsf{Reveal}}([x]^\mathsf{A} + [r]^\mathsf{A})$
for $b = 0$ to $1$ do
    $\mathsf{S}_b$: $p^{(1-b)} \leftarrow [s]_b^\mathsf{B} \oplus \pi_{1-b}, \mathsf{sk}^{(1-b)} \leftarrow \mathsf{sk}^{(1-b)}_{[s]_b^\mathsf{B}}$ .
    $\mathsf{S}_b$: Sends $p^{(1-b)} \Vert \mathsf{sk}^{(1-b)}$ to $\mathsf{S}_{1-b}$.        ▷ Key decryption
for $b = 0$ to $1$ do
    $\mathsf{S}_b$: $k^{(b)} \Vert \mathsf{id}^{(b)} \leftarrow C_b[p^{(b)}] \oplus \mathsf{sk}^{(b)}$
    $\mathsf{S}_b$: $[y]_b^\mathsf{B} \leftarrow \mathsf{Eval}^f(k^{(b)}, \mathsf{id}^{(b)}, \delta_x)$
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:

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

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:

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

From this transcript, $\{k_{t \oplus [s]^\mathsf{B}_{1-b}}, [r]_b^\mathsf{A}, \delta_x\}$ correspond to the FSS evaluation in the pre-processing model, and we resort to the security proof in [13] (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 $[s]^\mathsf{B}_{1-b} \oplus \pi b$ and $t \oplus [s]^\mathsf{B}_{1-b}$ provide information-theoretic secrecy for $[s]^\mathsf{B}_{1-b}$. Therefore, from $\mathtt{View}_b$, $\mathcal{A}$ learns nothing about the input x, s, nor the output $s \wedge f(x)$. □

5.3 CondEval in the Malicious Setting

We extend the setting to an active adversary. Since the two servers are symmetric, 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 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 $[s]_0^\mathsf{B}$ during the key decryption step, thus potentially flipping $[y]^\mathsf{B}$ to be equal to $[\neg(s \wedge f(x))]^\mathsf{B}$;
- After the online evaluation, S_0 may submit $\neg[y]_0^\mathsf{B}$ to B, 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 [20]. 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:

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

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

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

$$\mathcal{K}_\wedge^* \leftarrow \mathsf{CondEval.Setup}^*(\wedge, 1^\lambda, \ell_0, \ell_1, m),$$

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

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

$\mathsf{CondEval.Setup}^*$ is shown in Table 6, 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.

The concrete construction of CondEval.Eval* is shown in the following pseudo code, where it takes in the desired operation $\wedge$, as well as $\mathcal{K}^*_\wedge$, $(s)^m$, $[\![x]\!]^{\mathsf{A}}$ 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 clear-text FSS key is obtained by each of the two servers by decrypting one FSS key from other party's choice; By line 16 of CondEval.Eval*, we run $\mathcal{F}_{\mathsf{MacVryGen}}$ in Fig. 4 to generate a proof to verify that each mask value $\delta_{x^{(i)}}$ for $i \in [m]$ is revealed honestly. Finally, CondEval.Eval* outputs $(y)^m$ and an associated proof $[\sigma]^{\mathsf{A}}$ which can only be validated if they could pass the verification procedure in functionality CondEval.Verify as shown below.

We have the theorem of correctness, security and soundness for CondEval that work in the malicious setting in the following, however, due to page limit, we have attached their associated proofs in Appendix A.

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 $p = 1 - 1/2^m$.

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.

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

$$p < 2^{-\ell_1 + \log(\ell_1 + 1)} + 2^{1-m} + 2^{-(\ell_0 + \ell_1)}$$

where ℓ_1 denotes the length of the MAC key.

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

Players: $\mathsf{S}_0, \mathsf{S}_1$.
Functionality: $(y)^m, [\sigma]^{\mathsf{A}} \leftarrow$ CondEval.Eval$^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, [\![x]\!]^{\mathsf{A}})$.
Input: For each $b \in \{0,1\}$ that $\mathcal{K}^*_\wedge[b] = \{\{C_{i,b}, \mathsf{SK}_{i,b}, [\![r_i]\!]^{\mathsf{A}}_b\}_{i \in [m]}, [\psi]^{\mathsf{B}}_b, [\Delta]^{\mathsf{A}}_b\}$ 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: $((s \wedge f(x))^m, [\sigma]^{\mathsf{A}})$.

```
 1: $(\{[s_1]^{\mathsf{B}}, \cdots, [s_m]^{\mathsf{B}}\}) \leftarrow (s)^m$
 2: $\boldsymbol{V} \leftarrow [\emptyset]^m$
 3: for $b = 0$ to $1$ do
 4:     $[\sigma]^{\mathsf{A}}_b \leftarrow 0$
 5:     $\{C_b, \mathsf{SK}_b, [\![\boldsymbol{r}]\!]^{\mathsf{A}}_b, [\psi]^{\mathsf{B}}_b\} \leftarrow \mathcal{K}^*_\wedge[b]$
 6: for $i = 1$ to $m$ do
 7:     $[\![\delta_{x^{(i)}}]\!]^{\mathsf{A}} \leftarrow [\![x]\!]^{\mathsf{A}} + [\![r_i]\!]^{\mathsf{A}}$
 8:     $\delta_{x^{(i)}} \leftarrow \mathcal{F}_{\mathsf{Reveal}}([\delta_{x^{(i)}}]^{\mathsf{A}})$
 9:     $\boldsymbol{V}[i] \leftarrow (\delta_{x^{(i)}}, [\Delta \cdot \delta_{x^{(i)}}]^{\mathsf{A}})$
10:     for $b = 0$ to $1$ do
11:         $\mathsf{S}_b$: Sends $\{[s_i]^{\mathsf{B}}_b \oplus \pi_{1-b}, \mathsf{sk}^{(1-b)}_{[s_i]^{\mathsf{B}}_b}\}$ to $\mathsf{S}_{1-b}$ as $\{p^{(1-b)}, \mathsf{sk}^{(1-b)}\}$
    ▷ Key decryption
12:     for $b = 0$ to $1$ do
13:         $\mathsf{S}_b$: $k^{(b)} \| \mathsf{id}^{(b)} \| \xi^{(b)} \leftarrow C_{i,b}[p^{(b)}] \oplus \mathsf{sk}^{(b)}$
14:         $[\sigma]^{\mathsf{A}}_b \leftarrow [\sigma]^{\mathsf{A}}_b + \xi^{(b)}$
15:         $[y_i]^{\mathsf{B}}_b \leftarrow \mathsf{Eval}^f(k^{(b)}, \mathsf{id}^{(b)}, \delta_{x^{(i)}})$
16: $[\varsigma]^{\mathsf{A}} \leftarrow \mathcal{F}_{\mathsf{MacVryGen}}([\Delta]^{\mathsf{A}}, \boldsymbol{V})$
17: $[\sigma]^{\mathsf{A}} \leftarrow [\varsigma]^{\mathsf{A}} + [\sigma]^{\mathsf{A}}$
18: $(y)^m \leftarrow \{[y_1]^{\mathsf{B}}, \cdots, [y_m]^{\mathsf{B}}\}$
19: Output $(y)^m, [\sigma]^{\mathsf{A}}$
```

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

Players: $\mathsf{S}_0, \mathsf{S}_1, \mathsf{B}$.
Functionality: $s \leftarrow$ CondEval.Verify$(\{((y)^m_b, [\sigma]^{\mathsf{A}}_b)\}_{b \in \{0,1\}}, \psi)$.
Input: $((y)^m_b, [\sigma]^{\mathsf{A}}_b)$ 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 $[\sigma]^{\mathsf{A}}_0 + [\sigma]^{\mathsf{A}}_1 = 0$ then
        for $i = 1$ to $m$ do
            $y_i \leftarrow [y_i]^{\mathsf{B}}_0 \oplus [y_i]^{\mathsf{B}}_1$
            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 $z$.
```

6 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 $[c_1]^{\mathsf{B}}$ and an arithmetic

Table 6: 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.

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

secret sharing $[z]^{\mathsf{A}}$ using optimized secret sharing and FSS. Then, it employs $\mathsf{CondEval.Eval}(\wedge, [c_1]^{\mathsf{B}}, [x]^{\mathsf{A}})$ to compute $[c_1 \wedge \mathsf{Sign}(z)]^{\mathsf{B}}$ as the final authentication bit.

The solutions we propose for evaluating $\mathcal{F}_{\mathsf{ScaledCosAuth}}$ 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 $\mathcal{F}_{\mathsf{ScaledCosAuth}}$. 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 our actual evaluation of $\mathcal{F}_{\mathsf{ScaledCosAuth}}$, where the input vectors are small float point numbers, in order to reduce the cost of the underlying FSS module, we perform truncation once over $1/\tau^2 \cdot \mathsf{IP}(\boldsymbol{x}, \boldsymbol{y})^2 - \mathsf{IP}(\boldsymbol{x}, \boldsymbol{x}) \cdot \mathsf{IP}(\boldsymbol{y}, \boldsymbol{y})$ in $\mathcal{F}_{\mathsf{ScaledCosAuth}}$, such that it results in a smaller input domain of FSS.

6.1 Full protocol in the semi-honest setting

In the semi-honest setting, our final full protocol $\Pi_{\mathsf{NomadicSH}}$ comprises of three sub-protocols: Setup, $\mathcal{F}_{\mathsf{Input}}$ and Eval, executed in the Setup, Input, and Evaluation phases, respectively. Sub-protocol 1 details the setup phase, integrating CondEval.Setup and generating correlated randomness depends on $\mathcal{F}_{\mathsf{ScaledCosAuth}}$. Next, the client and B input their secret vectors $\boldsymbol{x}, \boldsymbol{y}$ by running $\mathcal{F}_{\mathsf{Input}}$ in Fig. 5. Lastly, with correlated randomness and input prepared, Eval produces the Boolean secret sharing of the desired authentication bit $[y]^{\mathsf{B}}$, where $y = \mathsf{Sign}\ (\cos(\mathrm{x}, \mathrm{y}) - \tau)$.

6.1.1 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 $\mathsf{Sign}\ (\cos(\mathrm{x}, \mathrm{y}) - \tau)$ as shown in Eq. 3.

6.1.2 Security. Our goal is to prove that sub-protocols 1 and 2 provide a secure implementation of $\mathcal{F}_{\mathsf{ScaledCosAuth}}$ when faced with a semi-honest PPT adversary $\mathcal{A}$ in a 2PC setting. We assert that, with the correlated randomness provided in sub-protocol 1, and following the online evaluation of sub-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 from [11], thereby concealing the information of $\alpha^{(1)}$ that is contained in $(k_0^{(1)}, k_1^{(1)})$ from $\mathcal{A}$. □

We realize the full protocol Π_{NomadicM} in the malicious setting, due to page limit, we have attached it in Appendix C.

7 EXPERIMENT

In this section, we aim to test the improved efficiency of $\Pi_{\mathsf{NomadicSH}}$ over the naive protocol Π_{OptimSH} in the semi-honest setting, as well as Π_{NomadicM} over the naive protocol Π_{NaiveM} in the malicious setting. To achieve that, we implement all four protocols in Python 3.10 [2] and have performed assessments of them through a use case of voice biometric authentication. In the following, we first

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

```
Players: The central bank B.
Functionality: $([\mathcal{R}]^{\mathsf{A}}, \mathcal{K}) \leftarrow \mathsf{Setup}(1^{\lambda}, \ell, \rho)$.
Input: A security parameter $\lambda$, element encoding length $\ell$, 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 $([\mathcal{R}]^{\mathsf{A}}, \bar{\mathcal{K}_{\wedge}}, \mathcal{K}_0)$
```

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

```
Players: $\mathsf{S}_0, \mathsf{S}_1$.
Functionality: $[c]^{\mathsf{B}} \leftarrow \mathsf{Eval}([\mathcal{R}]^{\mathsf{A}}, \bar{\mathcal{K}_{\wedge}}, \mathcal{K}_0, \langle \boldsymbol{x} \rangle^{\mathsf{A}}, \langle \boldsymbol{y} \rangle^{\mathsf{A}}, \tau, \rho, \ell)$.
Input: $[\mathcal{R}]^{\mathsf{A}}, \mathcal{K}_{\wedge}, \mathcal{K}_0, \langle \boldsymbol{x} \rangle^{\mathsf{A}}, \langle \boldsymbol{y} \rangle^{\mathsf{A}}$ from $\mathsf{S}_0$ and $\mathsf{S}_1$ 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, $T \in \mathbb{Z}_{2^{\ell}}$
  $[z]^{\mathsf{A}} \leftarrow T \cdot \mathsf{SS.MUL}(\langle u \rangle^{\mathsf{A}}, \langle u \rangle^{\mathsf{A}}, [(\alpha^{(1)})^2]^{\mathsf{A}})$
  $[z]^{\mathsf{A}} \leftarrow [z]^{\mathsf{A}} - 2^{\rho} \cdot \mathsf{SS.MUL}(\langle v \rangle^{\mathsf{A}}, \langle w \rangle^{\mathsf{A}}, [r_1 r_2]^{\mathsf{A}})$
  $[c]^{\mathsf{B}} \leftarrow \mathsf{CondEval.Eval_{trunc}}(\wedge, \bar{\mathcal{K}_{\wedge}}, [c_1]^{\mathsf{B}}, [z]^{\mathsf{A}})$        ▷ Second round
  Outputs $[c]^{\mathsf{B}}$
```

describe the dataset being used, then the experimental setting, and finally present the performance comparison of all protocols.

7.1 Data preparation

We propose to use a pre-trained model of the ECAPA-TDNN [25] model available in [4] to extract speaker embeddings. The model was trained using the development part of the VoxCeleb2 dataset [19] with 5994 speakers. The datasets RIR [37] and MUSAN [57] 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. 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-E [46], which consists of 37720 pairs of enrolment and verification ECAPA-TDNN embeddings, in which we have approximately half target and half non-target speakers. It is important to note that the resulting threshold τ in $\mathcal{F}_{\mathsf{ScaledCosAuth}}$ was established at a point where the FAR and FRR are equal.

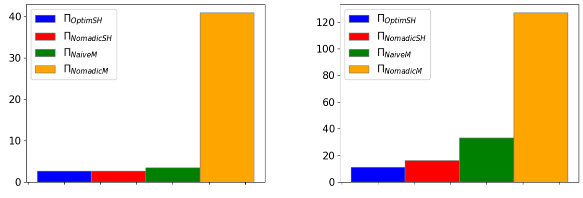
7.2 Experiment setting

We have performed all experiments on a relatively low-spec machine with 12th Gen Intel(R) Core(TM) i7-12700K processor, and 2 x 16 GB dual-channel DDR5 4400 MHz RAM. The hosting machine runs Ubuntu 22.04 LTS. We separately compared the performance of all protocols in the offline and online phase. Specifically, for the evaluation of the online phase, we created three simulated network environments using traffic control tools to emulate real-world network conditions.

To adapt the raw data from the ECAPA-TDNN embeddings [46] with float-point numbers to our protocols that deal with integers, we multiply the floating-point numbers with 2^8 ($\rho = 8$) and keep only the integer digits of the multiplication result to maintain precision in the fractional part of the raw data. We remark that employing a conversion parameter of $\rho = 8$ yields a non-significant deviation (0.003%) in terms of false acceptance rate (FAR) and false rejection rate (FRR), when compared to directly using floating-point numbers.

Regarding the FSS module underlines our protocols, which includes the interval containment FSS and our CondEval, we universally set the security parameter $\kappa = 128$, the input domain $\{0, 1\}^{32}$ that is sufficient to cover our use case, and we do implementation as per in Fig.3 of [11].

Specifically, we perform all the secure computation operation in a ring of 64 bits for protocols Π_{OptimSH} and $\Pi_{\mathsf{NomadicSH}}$ in the semi-honest setting; in the malicious setting we perform all operations in a ring of 64 + 32 bits, where 64 bits covers the circuit computation and 32 bits is the size of the secret authentication parameter α we selected. Additionally, in the implementation of Π_{NomadicM} we set $m = 30$ which is marginally smaller than the length of the secret authentication parameter α, according to Theorem 5 it gives us a soundness $p < 2^{-26}$.



(a) Generation time (ms)

(b) Commu. volume (KB)

Figure 2: Offline costs measured per server

[4] https://github.com/TaoRuijie/ECAPA-TDNN

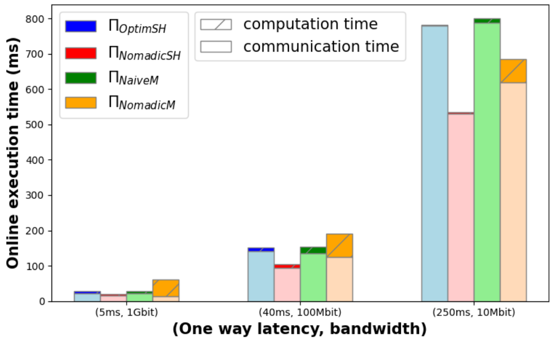


Figure 3: Online performance comparison of $\Pi_{\text{NomadicSH}}$ to Π_{OptimSH}, and $\Pi_{\text{NomadicSH}}$ to Π_{NaiveM} in different network settings.

7.3 Experiment results

We measured the offline computation time and communication volume as illustrated in Fig. 2, required by executing each protocol once. These results are quantified per server; specifically, the communication volume represents the data transmitted from the dealer to each server, while the computation time pertains to the duration required to generate correlated randomness data for each server. It is important to note that, across all four protocols, the dealer distributes correlated randomness in accordance with $\mathcal{F}_{\text{ScaledCosAuth}}$. This distribution is independent of the actual inputs used later in the online phase. Therefore, it is typically acceptable to incur higher costs during the offline phase. As depicted in Fig. 2, when comparing our protocol $\Pi_{\text{NomadicSH}}$ with protocol Π_{OptimSH}, $\Pi_{\text{NomadicSH}}$ incurs a marginally higher cost in both computation time and communication volume. In contrast, for our protocol Π_{NomadicM} compared to Π_{OptimSH}, there is a noticeable increase in both computation time and communication volume. This escalation is attributed to the utilization of multiple FSS keys in computing c_1 and c_2 of $\mathcal{F}_{\text{ScaledCosAuth}}$, and is considered an acceptable trade-off for the offline phase.

Fig. 3 presents the online execution time to run $\mathcal{F}_{\text{ScaledCosAuth}}$ with one pair of enrolment (reference) and verification (fresh) vectors as input in different network settings for all protocols. These measurements started from the moment all input secret sharing was in place and continued until the final secret sharing of the output bit c of $\mathcal{F}_{\text{ScaledCosAuth}}$ was obtained. Here, the execution time for each protocol is further divided into two parts, the local computation time and communication time. We have set up three common network settings with different network latency and bandwidth, which are denoted as LAN (10ms, 1Gbit), WAN1 (80ms, 100Mbit) and WAN2 (500ms, 10Mbit). The cost is measured for 20 runs on average. The online communication volume measured from our experiments are 0.09 KB, 0.69 KB, 0.21 KB, and 20.77 KB respectively for protocol Π_{OptimSH}, $\Pi_{\text{NomadicSH}}$, Π_{NaiveM} and Π_{NomadicM}.

In the semi-honest setting, $\Pi_{\text{NomadicSH}}$ demonstrates a significant improvement in efficiency over the naive protocol Π_{OptimSH}. Specifically, $\Pi_{\text{NomadicSH}}$ achieves a 31% reduction in online execution time compared to Π_{OptimSH}, even though we cost much more communication volume (0.69 KB) than what in Π_{OptimSH} (0.09 KB). This efficiency is attributed to the reduced number of rounds required; $\Pi_{\text{NomadicSH}}$ operates in two rounds, as opposed to Π_{OptimSH} that requires three rounds.

In the malicious setting, Π_{NomadicM} requires less communication time compared to the naive protocol Π_{NaiveM} in each of the network settings, however, it does not exhibit efficiency gains over the naive protocol Π_{NaiveM} in total except for the WAN2 setting. This performance characteristic primarily results from the use of multiple FSS keys, specifically $m = 30$ in our experiments. Our tests revealed a total computational cost of approximately 45ms for completing evaluations of c_1 and c in our protocol Π_{NomadicM}. This computational burden, however, is not intractable. One approach, as demonstrated in Funshade [33], involves using an efficiency-focused programming language. Their C implementation, for instance, requires a local computation time of just 0.55 ms, including the evaluation of one interval containment gate and a local inner product. Based on this, the local computation time for the multiple FSS components in our protocol Π_{NomadicM} is estimated to be no more than 16.5ms if implemented in C. Alternatively, using more powerful hardware can certainly reduce the computation overhead.

Considering that reducing wide area network latency will likely remain challenging in the near future, our Π_{NomadicM} protocol becomes a preferable choice. This is due to one round reduction compared to Π_{NaiveM}, making it more effective despite potentially higher computational demands. In summary, our experimental results demonstrate an online evaluation efficiency improvement for our proposed protocols over the SOTA when computing the cosine similarity functionality in the squared domain, especially in the WAN setting, where the network latency is a bottleneck for efficient secure computation, making it suitable for real-world applications.

8 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, *i.e.,* $\Pi_{\text{NomadicSH}}$ for the semi-honest setting, and Π_{NomadicM} the malicious setting. Both $\Pi_{\text{NomadicSH}}$ and Π_{NomadicM} rely on the recent advances of FSS [11, 13]. And $\Pi_{NomadicM}$ additionally 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 primitive 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.,* $s \circ f(x)$. CondEval 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 round-efficient, necessitating merely two communication rounds, but also proved to exhibit enhanced efficiency though our experiment in comparison to SOTA.

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] Anonymous. 2023. A Python Implementation of Nomadic. https://anonymous.4open.science/r/CondEval-F022.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. 2019. FLASH: Fast and robust framework for privacy-preserving machine learning. *Cryptology ePrint Archive* (2019).

[16] Nishanth Chandran, Divya Gupta, Sai Lakshmi Bhavana Obbattu, and Akash Shah. 2022. {SIMC}:{ML} Inference Secure Against Malicious Clients at {Semi-Honest} Cost. In *31st USENIX Security Symposium (USENIX Security 22)*. 1361–1378.

[17] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. 2019. Trident: Efficient 4pc framework for privacy preserving machine learning. *arXiv preprint arXiv:1912.02631* (2019).

[18] 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.

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

[20] 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.

[21] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2021. Fantastic four:{Honest-Majority}{Four-Party} secure computation with malicious security. In *30th USENIX Security Symposium (USENIX Security 21)*. 2183–2200.

[22] 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.

[23] 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.

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

[25] 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).

[26] Diana-Elena Fălămaş, 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.

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

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

[29] 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.

[30] Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. 2022. Llama: A low latency math library for secure inference. *Cryptology ePrint Archive* (2022).

[31] 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.

[32] Alberto Ibarrondo, Hervé Chabanne, Vincent Despiegel, and Melek Önen. 2023. Grote: Group testing for privacy-preserving face identification. In *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy*. 117–128.

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

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

[35] 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.

[36] 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.

[37] 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.

[38] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. 2021. {SWIFT}: Super-fast and robust {Privacy-Preserving} machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*. 2651–2668.

[39] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. 2021. Tetrad: Actively secure 4pc for secure training and inference. *arXiv preprint arXiv:2106.02850* (2021).

[40] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 336–353.

[41] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. 2021. Muse: Secure inference resilient to malicious clients. In *30th USENIX Security Symposium (USENIX Security 21)*. 2201–2218.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] A. Nagrani, J. S. Chung, and A. Zisserman. 2017. VoxCeleb: a large-scale speaker identification dataset. In *INTERSPEECH*.

[47] 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).

[48] 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.

[49] Arpita Patra and Ajith Suresh. 2020. BLAZE: blazing fast privacy-preserving machine learning. *arXiv preprint arXiv:2005.09042* (2020).

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

[51] 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.

[52] 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.

[53] 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).

[54] 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.

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

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

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

[58] 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.

[59] 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.

[60] 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).

[61] 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.

[62] 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.

[63] 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.

[64] 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 SUPPLEMENT FOR CONDEVAL IN THE MALICIOUS SETTING

A proof of theorem 3. Assume all servers follow $\mathsf{CondEval.Eval}^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, \llbracket x \rrbracket^{\mathsf{A}})$ honestly. $\forall i \in [m]$, 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 $\mathsf{CondEval.Verify}((y)^m, [\sigma]^{\mathsf{A}}, \psi)$ outputs -1. Thus, p is equal to $1 - 1/2^m$.

A proof of theorem 4. After online evaluation, for each $b \in \{0, 1\}$, S_b's transcript view is composed of transcript views of $\mathsf{CondEval.Eval}(\mathcal{K}^*_\wedge[i], [s_i]^{\mathsf{B}}, [x]^{\mathsf{A}})$ for each $i \in [m]$. From Theorem 2, we know that $\mathsf{CondEval.Eval}^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, \llbracket x \rrbracket^{\mathsf{A}})$ 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 [11], a probabilistic polynomial-time (PPT) the probability that $\mathcal{A}$ distinguishes between a trap key instance, a completely random key, or a normal key instance is negligible. Consequently, $\mathcal{A}$ gains no information about ψ.

A proof of theorem 5. If $\mathsf{CondEval.Verify}((y)^m, [\sigma]^{\mathsf{A}}, \psi) = \neg(s \wedge f(x))$, *w.l.o.g.,* we assume S_0 is the malicious server in the functionality $\mathsf{CondEval}^*$. The actual computation can be performed in the following two ways, only:

- Case 1: S_0 honestly follows $\mathsf{CondEval.Eval}^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, \llbracket x \rrbracket^{\mathsf{A}})$.
- Case 2: S_0 arbitrarily deviates from $\mathsf{CondEval.Eval}^*(\wedge, \mathcal{K}^*_\wedge, (s)^m, \llbracket x \rrbracket^{\mathsf{A}})$,
  - either by flipping $[s_i]^{\mathsf{B}}_0 (i \in [m])$ when performing the *KeyDec* step OT,
  - or, by dishonestly flipping partial shares when reporting $(s)^m$ to the receiver.

We denote by p_1 and p_2 the probability that $\mathsf{CondEval.Verify}$ $((y)^m, [\sigma]^{\mathsf{A}}, \psi) = \neg(s \wedge f(x))$ under cases 1 and 2, respectively.

In the first case, from Theorem 3 we know that the protocol outputs -1 with probability $1/2^m$, and $s \wedge f(x)$ with probability of $1 - 1/2^m$ it outputs. Thus, the protocol never outputs $\neg(s \wedge f(x))$ and hence, $p_1 = 0$.

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

(1) *A*: S_0 reports one or more incorrect arithmetic shares, submits a manipulated proof σ^*_0 such that, still, $\sigma^*_0 + \sigma_1 = 0$, and resulting to a valid flipped authenticated bit $(y^*)^m$ where $y^* = \neg(s \wedge f(x))$.

(2) B: S_0 manages to flip the computation 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 flip each $[y_i]^{\mathsf{B}}_0$ for $\psi[i] = 0$, and keeps $[y_i]^{\mathsf{B}}_0$ unchanged for $\psi[i] = 1$. In the end, S_0 submits his manipulated sharing of $(y)^m$ to B.

Each event is considered independently. Concerning event *A*, according to [20], it is known that the probability that event *A* occurs is no greater than $2^{-\ell_1 + \log(\ell_1 + 1)}$.

Regarding event *B*, the final outcome is altered when $f(x) = 1$ and only when all the *normal* indexes ($\psi[i] = 0$) are flipped during the *KeyDec* step. We analyse the probability that S_0 successfully flips all *normal* indexes without being detected as follows: First, we distinguish two complementary events B_1 and B_2 that are covered by *B*. We denote by B_1 the event that S_0 manages to flip the corresponding bit $[s_i]^{\mathsf{B}}_0$ for each $i \in [m]$ where $\psi[i] = 0$, and keeps the corresponding bit $[s_i]^{\mathsf{B}}_0$ unchanged for each $i \in [m]$ where $\psi[i] = 1$. On the other hand, B_2 corresponds to the event when S_0 manages to flip the corresponding bit $[s_i]^{\mathsf{B}}_0$ for each $i \in [m]$ where $\psi[i] = 0$, and at least one of the corresponding bit $[s_i]^{\mathsf{B}}_0$ is wrongly flipped

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 $L = (x_i, [\Delta \cdot x_i]^{\mathsf{A}})_{i \in [n]}$, 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: $[\varsigma]^{\mathsf{A}}$

```
for $b = 0$ to $1$ do
    $[\varsigma]^{\mathsf{A}}_b \leftarrow 0$
    for $i = 1$ to $n$ do
        $[\varsigma]^{\mathsf{A}}_b \leftarrow [\varsigma]^{\mathsf{A}}_b + [\Delta]^{\mathsf{A}}_b \cdot x_i - [\Delta \cdot x_i]^{\mathsf{A}}_b$
$\mathsf{S}_0$ and $\mathsf{S}_1$ re-randomize $[\varsigma]^{\mathsf{A}}$ locally.
Outputs $[\varsigma]^{\mathsf{A}}$
```

Figure 4: Our Homomorphic MAC Verification proof generation in the malicious setting.

for $\psi[i] = 1$, *i.e.,* S_0 does not perfectly correctly guesses all $\psi[i]$ where $\psi[i] = 1$, instead S_0 wrongly determines $\psi[i] = 0$ by at least one index i where in fact $\psi[i]$ is equal to 1. The reason why we distinguish B_1 from B_2 is because for event B_1 the adversary S_0 does not need to do anything to pass the verification. However, for 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 $\psi[i] = 0$ by at least one index i for $\psi[i] = 1$, under this case servers obtain an associated secret share of a non-zero random value.

Because we know ψ is uniformly generated, say S_0 flips each bit $[s_i]^{\mathsf{B}}_0$ independently for each $i \in [m]$ with probability q. Then, we have the probability of event B_1

$$P(B_1) = (\frac{q}{2} + \frac{1-q}{2})^m = 2^{-m},$$

and the probability of event B_2

$$P(B_2) = \left((\frac{q}{2} + \frac{1}{2})^m - P(B_1) \right) \cdot 2^{-(\ell_0+\ell_1)} < 2^{-(\ell_0+\ell_1)}.$$

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

$$P(B) = P^* \cdot (P(B_1) + P(B_2)) < P^* \cdot (2^{-m} + 2^{-(\ell_0+\ell_1)})$$

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,

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

In conclusion,

$$p = p_1 + p_2 < 2^{-\ell_1 + \log(\ell_1+1)} + 2^{1-m} + 2^{-(\ell_0+\ell_1)}.$$

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 $\perp$.

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

Figure 5: Input validation protocol in the semi-honest setting.

B $\mathcal{F}_{\mathsf{Input}}$ IN THE SEMI-HONEST SETTING

C OUR FULL PROTOCOL FOR $\mathcal{F}_{\mathsf{ScaledCosAuth}}$ IN THE MALICIOUS SETTING

We present sub-protocols 3 and 4 which evaluate $\mathcal{F}_{\mathsf{ScaledCosAuth}}$ in a stronger threat model, with one malicious server who might deviate from the protocol description in sub-protocol 2. More specifically:

- In sub-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 FSS 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:

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

  where $\mathcal{F}_{\mathsf{Input}^*}$ is shown in Fig. 6;
- Then in the evaluation phase, the two servers execute the online evaluation sub-protocol 4 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 sub-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 $(\mathcal{I}_x, \mathcal{I}_y, \llbracket\mathcal{R}\rrbracket^{\mathsf{A}}, \bar{\mathcal{K}}^*_\wedge, \mathcal{K}_0) \leftarrow \mathsf{Setup}^*(1^\lambda, m, \ell_0, \ell_1, \rho)$

```
**Players:** B.
**Functionality:** $(\mathcal{I}_x, \mathcal{I}_y, \llbracket\mathcal{R}\rrbracket^{\mathsf{A}}, \bar{\mathcal{K}}^*_\wedge, \mathcal{K}_0) \leftarrow \mathsf{Setup}^*(1^\lambda, m, \ell_0, \ell_1, \rho)$.
**Input:** A security parameter $\lambda, \ell, \ell_1, \rho \in \mathbb{Z}^+$.
**Output:** $(\mathcal{I}_x, \mathcal{I}_y, \llbracket\mathcal{R}\rrbracket^{\mathsf{A}}, \bar{\mathcal{K}}^*_\wedge, \mathcal{K}_0)$.
$\mathcal{K}^*_\wedge \leftarrow \mathsf{CondEval}^*.\mathsf{Setup}(\wedge, 1^\lambda, \ell_0, \ell_1, m)$
$\{\{C_{i,j}, \mathsf{SK}_{i,j}, \llbracket r_i\rrbracket^{\mathsf{A}}\}_{i\in[m],j\in\{0,1\}}, [\psi]^{\mathsf{B}}, [\Delta]^{\mathsf{A}}\} \leftarrow \mathcal{K}^*_\wedge$
$\Delta \leftarrow [\Delta]^{\mathsf{A}}_0 + [\Delta]^{\mathsf{A}}_1, \psi \leftarrow [\psi]^{\mathsf{A}}_0 \oplus [\psi]^{\mathsf{A}}_1$
$\boldsymbol{r}^{(\boldsymbol{x})}_{\boldsymbol{in}} \leftarrow_\$ \mathbb{Z}^n_{2^\ell}, \llbracket\boldsymbol{r}^{(\boldsymbol{x})}_{\boldsymbol{in}}\rrbracket^{\mathsf{A}} \leftarrow \mathsf{SS.Share}(\{\boldsymbol{r}^{(\boldsymbol{x})}_{\boldsymbol{in}}, \Delta\cdot\boldsymbol{r}^{(\boldsymbol{x})}_{\boldsymbol{in}}\}, \mathbb{Z}_{2^{\ell_0+\ell_1}})$
$\boldsymbol{r}^{(\boldsymbol{y})}_{\boldsymbol{in}} \leftarrow_\$ \mathbb{Z}^n_{2^\ell}, \llbracket\boldsymbol{r}^{(\boldsymbol{y})}_{\boldsymbol{in}}\rrbracket^{\mathsf{A}} \leftarrow \mathsf{SS.Share}(\{\boldsymbol{r}^{(\boldsymbol{y})}_{\boldsymbol{in}}, \Delta\cdot\boldsymbol{r}^{(\boldsymbol{y})}_{\boldsymbol{in}}\}, \mathbb{Z}_{2^{\ell_0+\ell_1}})$
$(\boldsymbol{x_{in}}, \boldsymbol{y_{in}}) \leftarrow_\$ \mathbb{Z}^{2n}_{2^\ell}, \boldsymbol{u_{in}} \leftarrow \mathsf{Prod}(\boldsymbol{x_{in}}, \boldsymbol{y_{in}})$
$\boldsymbol{v_{in}} \leftarrow \mathsf{Prod}(\boldsymbol{x_{in}}, \boldsymbol{x_{in}}), \boldsymbol{w_{in}} \leftarrow \mathsf{Prod}(\boldsymbol{y_{in}}, \boldsymbol{y_{in}})$
$(r_1, r_2) \leftarrow_\$ \mathbb{Z}^2_{2^\ell}$
**for** $i = 1$ to $m$ **do**
    $\alpha^{(1,i)} \leftarrow_\$ \mathbb{Z}_{2^\ell}$
    $r \leftarrow_\$ \mathbb{Z}_{2^\rho}$
    $\alpha^{(2,i)} \leftarrow [r_i]^{\mathsf{A}}_0 + [r_i]^{\mathsf{A}}_1$
    $\eta_i \leftarrow \mathsf{Shift}(\alpha^{(2,i)}, -\rho, \ell) + r$
    $(k^{(1,i)}_0, k^{(1,i)}_1) \leftarrow \mathsf{Gen}^{[0,2^{\ell-1}]}_\ell(1^\lambda, \alpha^{(1,i)}, 1-\psi[i], 0, \mathbb{Z}_2)$
    $\llbracket\eta_i\rrbracket^{\mathsf{A}} \leftarrow (\mathsf{SS.Share}(\eta_i, \mathbb{Z}_{2^{\ell_0+\ell_1}}), \mathsf{SS.Share}(\Delta\cdot\eta_i, \mathbb{Z}_{2^{\ell_0+\ell_1}}))$
$\bar{\mathcal{K}}^*_\wedge \leftarrow \{\{C_{i,j}, \mathsf{SK}_{i,j}, \llbracket\eta_i\rrbracket^{\mathsf{A}}\}_{i\in[m],j\in\{0,1\}}, [\psi]^{\mathsf{B}}, [\Delta]^{\mathsf{A}}\}$
$\mathcal{R} \leftarrow \{\boldsymbol{x_{in}}, \boldsymbol{y_{in}}, \boldsymbol{u_{in}}, \boldsymbol{v_{in}}, \boldsymbol{w_{in}}, (r_1, r_2, r_1r_2), (\alpha^{(1,1)})^2\}$
$\llbracket\mathcal{R}\rrbracket^{\mathsf{A}} \leftarrow (\mathsf{SS.Share}(\mathcal{R}, \mathbb{Z}_{2^{\ell_0+\ell_1}}), \mathsf{SS.Share}(\mathcal{R}\cdot\Delta, \mathbb{Z}_{2^{\ell_0+\ell_1}}))$
$\mathcal{K}_0 \leftarrow \{k^{(1,i)}_0, k^{(1,i)}_1\}_{i\in[m]}$
$\mathcal{I}_x \leftarrow (\llbracket\boldsymbol{r}^{(\boldsymbol{x})}_{\boldsymbol{in}}\rrbracket^{\mathsf{A}}, \llbracket\boldsymbol{x_{in}}\rrbracket^{\mathsf{A}}), \mathcal{I}_y \leftarrow (\llbracket\boldsymbol{r}^{(\boldsymbol{y})}_{\boldsymbol{in}}\rrbracket^{\mathsf{A}}, \llbracket\boldsymbol{y_{in}}\rrbracket^{\mathsf{A}})$
Outputs $(\mathcal{I}_x, \mathcal{I}_y, \llbracket\mathcal{R}\rrbracket^{\mathsf{A}}, \bar{\mathcal{K}}^*_\wedge, \mathcal{K}_0)$
```

```
FUNCTIONALITY $\langle\langle x\rangle\rangle^{\mathsf{A}}$ or $\perp \leftarrow \mathcal{F}_{\mathsf{Input}^*}(x, [\Delta]^{\mathsf{A}}, \llbracket r\rrbracket^{\mathsf{A}}, \llbracket r_{\mathsf{in}}\rrbracket^{\mathsf{A}})$:
**Players:** A client C, $\mathsf{S}_0$ and $\mathsf{S}_1$.
**Input:** A secret $x$ from C, and $\llbracket r\rrbracket^{\mathsf{A}}, \llbracket r_{\mathsf{in}}\rrbracket^{\mathsf{A}}$ from $\mathsf{S}_0, \mathsf{S}_1$.
**Output:** $\langle\langle x\rangle\rangle^{\mathsf{A}}$
$\forall b \in \{0,1\}$, $\mathsf{S}_b$ sends $[r]^{\mathsf{A}}_b$ to C.
$\forall b \in \{0,1\}$, C returns $\delta^b_x = x + [r]^{\mathsf{A}}_0 + [r]^{\mathsf{A}}_1$ to $\mathsf{S}_b$.
$\forall b \in \{0,1\}$, $\mathsf{S}_b$ computes
    $$\llbracket x\rrbracket^{\mathsf{A}}_b = (b\cdot\delta^b_x - [r]^{\mathsf{A}}_b, [\Delta]^{\mathsf{A}}_b\cdot\delta_x - [\Delta\cdot r]^{\mathsf{A}}_b)$$
Servers run $\mathcal{F}_{\mathsf{Reveal}}([x + r_{\mathsf{in}}]^{\mathsf{A}})$ and obtain $x + r_{\mathsf{in}}$. After that, $\forall b \in \{0,1\}$ $\mathsf{S}_b$ computes
    $$\langle\langle x\rangle\rangle^{\mathsf{A}}_b = (x + r_{\mathsf{in}}, \llbracket x\rrbracket^{\mathsf{A}}_b, \llbracket r_{\mathsf{in}}\rrbracket^{\mathsf{A}}_b)$$
Outputs $\langle\langle x\rangle\rangle^{\mathsf{A}}$
```

Figure 6: Input validation protocol in the malicious setting.

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

$$p < 2^{-\ell_1 + \log(\ell_1+1)} + 2^{1-m} + 2^{-(\ell_0+\ell_1)}.$$

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

```
**Players:** $\mathsf{S}_0, \mathsf{S}_1$.
**Functionality:** $((y)^m, [\sigma]^{\mathsf{A}}) \leftarrow \mathsf{Eval}^*(\llbracket\mathcal{R}\rrbracket^{\mathsf{A}}, \langle\langle\boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \langle\langle\boldsymbol{y}\rangle\rangle^{\mathsf{A}}, \bar{\mathcal{K}}^*_\wedge, \mathcal{K}_0, \tau, \rho)$.
**Input:** Secret sharing $[\Delta]^{\mathsf{A}}, \llbracket\mathcal{R}\rrbracket^{\mathsf{A}}, \langle\langle\boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \langle\langle\boldsymbol{y}\rangle\rangle^{\mathsf{A}}$ 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:** $((y)^m, [\sigma]^{\mathsf{A}})$.
$\boldsymbol{p} \leftarrow [\emptyset]^{m+2}$
$\llbracket u\rrbracket^{\mathsf{A}} \leftarrow \mathsf{SS.IP}(\langle\langle\boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \langle\langle\boldsymbol{y}\rangle\rangle^{\mathsf{A}}, \llbracket\boldsymbol{u_{in}}\rrbracket^{\mathsf{A}})$
$\llbracket v\rrbracket^{\mathsf{A}} \leftarrow \mathsf{SS.IP}(\langle\langle\boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \langle\langle\boldsymbol{x}\rangle\rangle^{\mathsf{A}}, \llbracket\boldsymbol{v_{in}}\rrbracket^{\mathsf{A}})$
$\llbracket w\rrbracket^{\mathsf{A}} \leftarrow \mathsf{SS.IP}(\langle\langle\boldsymbol{y}\rangle\rangle^{\mathsf{A}}, \langle\langle\boldsymbol{y}\rangle\rangle^{\mathsf{A}}, \llbracket\boldsymbol{w_{in}}\rrbracket^{\mathsf{A}})$
$\delta_v \leftarrow \mathcal{F}_{\mathsf{Reveal}}([v + r_1]^{\mathsf{A}})$                          ▷ First round
$\delta_w \leftarrow \mathcal{F}_{\mathsf{Reveal}}([w + r_2]^{\mathsf{A}})$                          ▷ First round
$\boldsymbol{p}[1] \leftarrow (\delta_v, [\Delta\cdot\delta_v]^{\mathsf{A}}), \boldsymbol{p}[2] \leftarrow (\delta_w, [\Delta\cdot\delta_w]^{\mathsf{A}})$
$\langle\langle v\rangle\rangle^{\mathsf{A}} \leftarrow (\delta_v, \llbracket v\rrbracket^{\mathsf{A}}, \llbracket r_1\rrbracket^{\mathsf{A}})$
$\langle\langle w\rangle\rangle^{\mathsf{A}} \leftarrow (\delta_w, \llbracket w\rrbracket^{\mathsf{A}}, \llbracket r_2\rrbracket^{\mathsf{A}})$
$\{\{C_{i,j}, \mathsf{SK}_{i,j}, \llbracket r_i\rrbracket^{\mathsf{A}}\}_{i\in[m],j\in\{0,1\}}, [\psi]^{\mathsf{B}}, [\Delta]^{\mathsf{A}}\} \leftarrow \mathcal{K}^*_\wedge$
$\{k^{(1,i)}_0, k^{(1,i)}_1\}_{i\in\{0,1\}} \leftarrow \mathcal{K}_0$
**for** $i = 1$ to $m$ **do**
    $\llbracket\delta_{\boldsymbol{u}^{(i)}}\rrbracket^{\mathsf{A}} \leftarrow \llbracket u\rrbracket^{\mathsf{A}} + \llbracket\alpha^{(1,i)}\rrbracket^{\mathsf{A}}$
    $\delta_{\boldsymbol{u}^{(i)}} \leftarrow \mathcal{F}_{\mathsf{Reveal}}([\delta_{\boldsymbol{u}^{(i)}}]^{\mathsf{A}})$                  ▷ First round
    **for** $b = 0$ to $1$ **do**
        $[s_i]^{\mathsf{B}}_b \leftarrow \mathsf{Eval}^{[0,2^{\ell-1}]}_\ell(k^{(1,i)}_b, b, \delta_{\boldsymbol{u}^{(i)}})$
    $\boldsymbol{p}[2+i] \leftarrow (\delta_{\boldsymbol{u}^{(i)}}, [\Delta\cdot\delta_{\boldsymbol{u}^{(i)}}]^{\mathsf{A}})$
$(s)^m \leftarrow \{[s_1]^{\mathsf{B}}, \cdots, [s_m]^{\mathsf{B}}\}$
$T \leftarrow \mathsf{Shift}(\frac{1}{\tau^2}, -\rho, \ell)$
$\llbracket z\rrbracket^{\mathsf{A}} \leftarrow T\cdot\mathsf{SS.MUL}(\langle\langle u^{(1)}\rangle\rangle^{\mathsf{A}}, \langle\langle u^{(1)}\rangle\rangle^{\mathsf{A}}, \llbracket(\alpha^{(1,1)})^2\rrbracket^{\mathsf{A}})$
$\llbracket z\rrbracket^{\mathsf{A}} \leftarrow \llbracket z\rrbracket^{\mathsf{A}} - 2^\rho\cdot\mathsf{SS.MUL}(\langle\langle v\rangle\rangle^{\mathsf{A}}, \langle\langle w\rangle\rangle^{\mathsf{A}}, \llbracket r_1r_2\rrbracket^{\mathsf{A}})$
$((y)^m, [\sigma]^{\mathsf{A}}) \leftarrow \mathsf{CondEval.Eval}^*_{\mathsf{trunc}}(\wedge, \bar{\mathcal{K}}^*_\wedge, (s)^m, \llbracket z\rrbracket^{\mathsf{A}})$     ▷ Second round
$[\varsigma]^{\mathsf{A}} \leftarrow \mathcal{F}_{\mathsf{MacVryGen}}([\Delta]^{\mathsf{A}}, \boldsymbol{p})$
$[\sigma]^{\mathsf{A}} \leftarrow [\sigma]^{\mathsf{A}} + [\varsigma]^{\mathsf{A}}$
Outputs $((y)^m, [\sigma]^{\mathsf{A}})$.
```

C.0.2 Security. We assert that, sub-protocols 3 and 4 provide a secure implementation of $\mathcal{F}_{\mathsf{ScaledCosAuth}}$ when faced with an active PPT $\mathcal{A}$ in the 2PC setting. Specifically, with the correlated randomness provided in Protocol 3, following the online evaluation of sub-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 $\mathsf{CondEval.Eval}^*_{\mathsf{trunc}}(\wedge, \bar{\mathcal{K}}^*_\wedge, (s)^m, \llbracket z\rrbracket^{\mathsf{A}})$. Still, we need to prove that the view transcript

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

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 $\{(k^{(1,i)}_b, \delta_{\boldsymbol{u}^{(i)}})\}_{i\in[m]}$ which are correlated, we argue that $\mathcal{A}$ has only the view $k^{(1,i)}_b$ which is pseudo-random (computationally indistinguishable from a real random key) as proved in Fig. 1 in [11]; Thus, hiding the information of $\alpha^{(1,i)}$ contained in $(k^{(1,i)}_0, k^{(1,i)}_1)$ from $\mathcal{A}$. □

C.1 Truncating δ_x

It is worth to note that while executing 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, it performs

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

in $\mathsf{CondEval.Eval}_{\mathsf{trunc}}$ 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 7.3, such a truncation has almost no impact on the actual accuracy of the protocol.

D 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 7 we provide the details of 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 CondEval.Eval $(\vee, \mathcal{K}_\vee, [s]^{\mathsf{B}}, [x]^{\mathsf{A}})$, which uses the correlated FSS key pairs prepared in Table 7, 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

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

or

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

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

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

or

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

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

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

Table 7: 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 5.3 and we omit the details here.