Funshade: Functional Secret Sharing for Two-Party Secure Thresholded Distance Evaluation

Alberto Ibarrondo
Idemia & EURECOM
Sophia Antipolis, France
ibarrond@eurecom.fr

Hervé Chabanne
Idemia & Telecom Paris
Paris, France

Melek Önen
EURECOM
Sophia Antipolis, France

ABSTRACT

We propose a novel privacy-preserving, two-party computation of various distance metrics (e.g., Hamming distance, Scalar Product) followed by a comparison with a fixed threshold, which is known as one of the most useful and popular building blocks for many different applications including machine learning, biometric matching, etc. Our solution builds upon recent advances in functional secret sharing and makes use of an optimized version of arithmetic secret sharing. Thanks to this combination, our new solution named Funshade is the first to require only one round of communication and two ring elements of communication in the online phase, outperforming all prior state-of-the-art schemes while relying on lightweight cryptographic primitives. Lastly, we implement the solution from scratch in Python using efficient C++ blocks, testifying its high performance.

KEYWORDS

Functional Secret Sharing, Secure Two Party Computation, Scalar Product, Hamming Distance

1 INTRODUCTION

The computation of privacy-preserving distance metrics $f_{dist}(\boldsymbol{x}, \boldsymbol{y})$ between two vectors $\boldsymbol{x}, \boldsymbol{y}$ followed by a comparison with a threshold θ is a very popular building block in many applications in need of privacy protection, including machine learning (e.g., k-nearest neighbors[63], linear regression [34]), biometrics (e.g., biometric authentication [44, 53], biometric identification [31]) etc.

The literature counts many solutions based on various cryptographic techniques that allow computation over sensitive data while preserving its privacy: *Secure Multiparty Computation* (MPC) (garbled circuits [60], secret sharing (SS) [38, 56]) to split the distance computation across multiple entities [19, 29, 32], *Fully Homomorphic Encryption* (FHE) [23, 33, 35] supporting addition and multiplication between ciphertexts [4, 7, 22], and *Functional Encryption* [2, 9] as a public-key encryption scheme that supports evaluation of scalar products when decrypting the ciphertexts [3, 10].

However, not all operations are born equal. While linear operations are widely covered by all the privacy-preserving techniques, the protection of non-linear operations including the comparison to a threshold θ is much harder to attain. Computing this non-linear operation with most MPC primitives is often communication intensive (e.g., [29, 59]) both in terms of communication size and in number of rounds; FHE-based techniques must resort to computation-intensive algorithms [24, 40]; and FE-based techniques are limited to linear function evaluations. Luckily, recent solutions [11, 14, 54] have demonstrated a considerable improvement to securely realize the comparison to θ by resorting to Functional Secret Sharing (FSS) primitives.

In [15], the authors specifically study the computation of distance metrics. They propose GSHADE, a decomposition of each metric into a combination of local single-input functions and a cross-product, employing Oblivious Transfer [50] to preserve the privacy of their construction.

We draw inspiration from the family of distance metrics covered in GSHADE and integrate FSS-based threshold comparison primitives from [11] with an optimized version of Secret Sharing [49] in a two-party computation (2PC) protocol to perform privacy-preserving distance metric computations with a subsequent comparison to θ. To summarize our contributions, our solution:

- requires just one round of communication in the online phase, lowering the communication costs with respect to the two-round state-of-the-art solutions from AriaNN [54] and Boyle et. al. [11] by merging the round of communication required for the scalar product with that of the comparison to θ,
- sends two ring elements only in the online phase, reducing the communication size of previous solutions by a factor of $2l$ (for input vectors with l elements),
- features 100% correctness in the comparison result, as opposed to [54],
- is implemented and open-sourced in a standalone Python library with efficient C++ primitives.

The paper is outlined as follows. Section 2 describes the preliminaries, the distance metrics we consider in this work and some applications. Section 3 details the proposed solution, including its security analysis. Section 4 addresses previous work and positions our contribution, wrapping up with the conclusions and next steps in Section 5.

2 PRELIMINARIES

Notation

We use bold letters to denote vectors (e.g., $\boldsymbol{x}, \boldsymbol{y}$) and non-bold letters for scalars. $\boldsymbol{x}^{(i)}$ denotes the ith element of vector $\boldsymbol{x}$. For convenience we omit the $^{(i)}$ superscripts in lengthy element-wise additions of the form $\Sigma[\boldsymbol{a}^{(i)} + \boldsymbol{b}^{(i)} + \dots]$. We write $\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{c}$ to denote the element-wise multiplication of two vectors where $\boldsymbol{c}^{(i)} = \boldsymbol{a}^{(i)}\boldsymbol{b}^{(i)}$, and $\boldsymbol{a}^T\boldsymbol{b}$ to denote the inner (scalar) product between two vectors.

We reserve the notation P_{descr} to indicate a party/player in our scenario with a certain description (e.g., P_{setup} for the party in charge of the setup, P_{in_x} for the party holding the input vector $\boldsymbol{x}$), and label (P_0, P_1) for the two computing parties in the 2PC paradigm. We generalize behavior common to the two computing parties by resorting to P_j, where $j \in \{0, 1\}$. We use $r \longleftarrow 4$ to set the local variable r to 4, and P_a :SEND $r \Rightarrow P_b$ for party a sending value r to party b. We note $\mathcal{U}_{[S]}$ as the uniform random distribution in the set S, and write $r \sim \mathcal{U}_{[S]}$ to indicate sampling that distribution and assigning the sample to r. We employ $1_{x \in A}$ to denote the indicator function (e.g., $1_{x>5} = 1 \iff x > 5$):

$$1_{x \in A} \equiv 1_A(x) \triangleq \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \notin A, \end{cases}$$

As a special case of indicator function, the unit step function is defined as $H(x) = 1_{x \in \mathbb{R}^*_+} = 1_{x \geqslant 0}$. We implicitly consider a twos-complement encoding to map between signed and unsigned n-bit integers, a bijective mapping between $[-2^{n-1}, 2^{n-1} - 1]$ and $[0, 2^n - 1]$ by applying $\mod 2^n$, where the interval of negative values $[-2^{n-1}, -1]$ is mapped to the upper half of the unsigned interval $[2^{n-1}, 2^n - 1]$. As such, the unit step function for n-bit integers $H_n(x)$ corresponds to $H_n(x) = 1_{x \in \mathbb{Z}_{n_+}} = 1_{0 \leqslant x \leqslant 2^{n-1}-1}$.

We write $\langle x \rangle$ to indicate that value x is arithmetically secret shared into shares (x_0, x_1) among computing parties (P_0, P_1) such that P_0 holds the share x_0 and P_1 holds the share x_1. Likewise, we write $\langle\!\langle x \rangle\!\rangle$ to indicate that value x is Π-secret shared (Section 2.1.2) into three shares $(\Delta_x, \delta_{x0}, \delta_{x1})$, where both parties (P_0, P_1) hold Δ_x and each party P_j holds δ_{xj}.

2.1 Multi-Party Computation

Secure multi-party computation (or MPC) [6, 21, 38, 60] allows two or more parties to compute any mathematical function on private inputs without revealing anything but the output of the function. Typically, MPC is instantiated in the preprocessing model, where specially crafted randomness is generated in an offline input-independent phase from either a trusted dealer or via an offline interaction, and then this randomness is used in the online phase to compute the function, once the inputs are known. This two-phase approach yields considerable performance benefits. Some examples of this correlated randomness include Beaver multiplication triples [5] and garbled circuit preprocessing [29, 60].

When used to evaluate circuits based on only binary or only arithmetic interactions, MPC protocols present very fast online execution. However, applications such as biometrics or machine learning require a combination of linear operations (additions and multiplications over a large ring) and non-linear operations such as integer comparison or truncation. The cost of blindly implementing these two types of operations with only one MPC circuit type can be prohibitively high. To address this, many works have tackled mixed-mode MPC to provide efficient conversions between arithmetic and binary domains, supporting both linear and non-linear operations [19, 29, 48, 49]. Yet, these conversions often entail a hefty communication overhead in the online phase.

In line with the TinyTable protocol [27] to secret share truth tables in a succinct manner, Boyle et al. propose a very promising approach [11, 14] based on Functional Secret Sharing (FSS) [12, 13]. Offering the same online communication and round complexity for non-linear function evaluations as for pure arithmetic computations in arithmetic-only circuits, FSS relies on fast symmetric cryptography primitives to also yield fast online evaluation.

The present work will benefit from standard arithmetic secret sharing techniques [5], more evolved secret sharing techniques emanating from research in mixed-mode operations [49] and modern FSS approaches [11] to achieve a lightweight and highly communication efficient biometric matching protocol. As such, we now delve into the details of these techniques.

2.1.1 *Additive Secret Sharing.*

Secret sharing is a cryptographic primitive that allows a secret x to be shared among n parties, such that any t of them can reconstruct the secret. The secret sharing scheme is defined by k, the number of parties, and threshold t, minimum number of parties required to reconstruct the secret. In the domain of two-party computation (2PC), the number of parties is $n = 2$ and the threshold is $t = 2$. This work focuses on 2PC arithmetic secret sharing in rings (shortened to SS for convenience), where a secret value x is split into two random shares x_0 and x_1 such that $x = x_0 + x_1 \mod N$, with N being the ring size. The shares are distributed to the two computing parties such that party P_j receives the share x_j. With this sharing scheme, the two parties can perform local addition/subtraction of two secret shared values. Additionally, parties can resort to Beaver's multiplication triples [5] to perform multiplication at the cost of one round of communication:

$$\begin{aligned}
\texttt{SS.add:}\ & \mathit{Online}(P_0, P_1): && \langle x \pm y \rangle = \langle x \rangle \pm \langle y \rangle \\
\texttt{SS.mult:}\ & \mathit{Offline}(P_{setup}): && \langle a \rangle, \langle b \rangle \sim \mathcal{U}_{[\mathbb{Z}_N^{2\times 2}]} \\
& && \langle c \rangle \leftarrow \langle a \cdot b \rangle \\
& && \textsc{Send}(a_j, b_j, c_j) \Rightarrow P_j \\
& \mathit{Online}(P_0, P_1): && \textsc{Send}(x_j - a_j, y_j - b_j) \Rightarrow P_{1-j} \\
& && \langle x \cdot y \rangle = \langle b \rangle (x-a) + \langle a \rangle (y-b) + \langle c \rangle \\
& && \quad + (x-a)(y-b)
\end{aligned} \tag{1}$$

At the end of the computation, the resulting secret shared value can be reconstructed by sending both shares to a chosen party P_{res}, to add the two shares together and reconstruct the result. We work with $N = 2^n$ for values of $n \in \{8, 16, 32, 64\}$ to benefit from a considerable speedup when dealing with n-bit modular arithmetic thanks to native integer types present in modern computers.

Of special interest for this work,computing a scalar product $\boldsymbol{x}^T \boldsymbol{y} = \sum_{i=1}^{l} \boldsymbol{x}^{(i)} \cdot \boldsymbol{y}^{(i)}$ with SS requires sending 2 terms per multiplication, for a total of $2l$ values sent.

2.1.2 *Π-Secret Sharing.*

Originally inspired by ASTRA [20] in the 3PC scenario, ABY2.0 [49] introduced a novel way to perform additive secret sharing in 2PC[1], where a value x is split into three random shares $(\Delta_x, \delta_{x0}, \delta_{x1})$ such that $\Delta_x = x + \delta_{x0} + \delta_{x1} \mod N$. The δ-shares δ_{xj} are distributed to each computing party P_j forming an arithmetic secret sharing $\langle \delta_x \rangle$ of $\delta_x = \delta_{x0} + \delta_{x1}$, while the Δ-share Δ_x is held by both parties at once. We name this sharing scheme as Π-secret sharing, due to the "horizontally" mutual Δ-share and the two "vertically" separated δ-shares, and denote the Π-sharing of value x as $\langle\!\langle x \rangle\!\rangle$. The Π-sharing scheme allows local addition/subtraction, and multiplication at the cost of one round of communication. The essential difference with respect to the SS scheme is that the δ-shares can be precomputed (leaving only the Δ-share to be determined in the online phase), and thus carry extra correlation that was not possible with standard SS. The main arithmetic operations in ΠSS are defined as follows:

[1] Note that ABY2.0 [49] refers to arithmetic secret sharing as $[\cdot]$-sharing and Π-secret sharing as $\langle \cdot \rangle$-sharing.

$$
\begin{aligned}
\Pi\texttt{SS.add:}\ \ & \textit{Online}(P_0, P_1): && \langle\!\langle x \pm y \rangle\!\rangle = \langle\!\langle x \rangle\!\rangle \pm \langle\!\langle y \rangle\!\rangle \\
\Pi\texttt{SS.mult:}\ \ & \textit{Offline}(P_{setup}): && \langle \delta_x \rangle, \langle \delta_y \rangle, \langle \delta_z \rangle \sim \mathcal{U}_{[\mathbb{Z}_N^{3\times 2}]} \\
& && \langle \delta_{xy} \rangle \leftarrow \langle \delta_x \cdot \delta_y \rangle \\
& && \textsc{Send}\ (\delta_{xj}, \delta_{yj}, \delta_{xyj}) \Rightarrow P_j \\
& \textit{Online}(P_0, P_1): && \langle x \cdot y \rangle \equiv \langle z \rangle \leftarrow j \cdot \Delta_x \Delta_y - \Delta_x \langle \delta_y \rangle \\
& && \qquad - \Delta_y \langle \delta_x \rangle + \langle \delta_{xy} \rangle \\
& && \langle \Delta_z \rangle \equiv \langle z \rangle + \langle \delta_z \rangle \\
& && \textsc{Send}\ (\Delta_{zj}) \Rightarrow P_{1-j} \\
& && \langle\!\langle z \rangle\!\rangle \equiv (\Delta_{z0} + \Delta_{z1}, \langle \delta_z \rangle)
\end{aligned}
\tag{2}
$$

Crucially, the online phase of the Π-sharing multiplication first computes a local arithmetic sharing of the result, and then uses one round of communication to convert the result back into Π-shares. As promptly explained in [49], this moves the communication from the multiplication inputs to the multiplication outputs, which yields sizeable advantages in terms of communication size for operations such as the scalar product: computing a scalar product $\boldsymbol{x}^T\boldsymbol{y} = \sum_{i=1}^{l} \boldsymbol{x}^{(i)} \cdot \boldsymbol{y}^{(i)}$ with ΠSS requires sending 2 values only for the entire operation, thus reducing the communication size by a factor of l with respect to SS.

2.1.3 Functional Secret Sharing. A 2PC Functional Secret Sharing (FSS) scheme [12, 13] for a function family $\mathcal{F}$ splits a function $f \in \mathcal{F}$ into two additive shares (f_0, f_1), such that each f_j hides f and $f_0(x) + f_1(x) = f(x)$ for every input x. Beyond trivial solutions such as secret-sharing the truth-table of f, FSS schemes seek succinct descriptions of f_j (function keys $\boldsymbol{k}_0, \boldsymbol{k}_1$) with efficient online execution. Since both function shares must evaluate on the same value x, this value must be made public to both computing parties P_j. To maintain input data privacy, a random mask r is added to the secret input x, so that the opened value $\hat{x} = x + r$ completely hides x before using it as input to the FSS evaluation. In order to obtain full correctness on the function evaluation with respect to $f(x)$, the class of functions $\mathcal{F}$ is restricted to $f_r(x) = f(x + r)$, where the mask is known by the dealer and used for the key generation.

For addition and multiplication gates over a ring $\mathbb{Z}_{2^n}$, the FSS gates correspond to Beaver's protocol [5]. A much more interesting case arises in [11, 14], where non-linear operations including zero-test, integer comparison or bit decomposition are efficiently constructed using a small number of invocations of FSS primitives. Luckily, these FSS gates make a black-box use of any secure pseudorandom generator (PRG), yielding short keys and fast implementations based on AES.

Grounded on the MPC preprocessing model, a FSS gate is composed of two algorithms:

- $\mathsf{Gen}(1^\lambda, f) \rightarrow (\boldsymbol{k}_0, \boldsymbol{k}_1)$ is a PPT key generation algorithm that, given the security parameter λ and the description of a function $f : \mathbb{G}_{in} \mapsto \mathbb{G}_{out}$, outputs a pair of functional keys ($\boldsymbol{k}_0$, $\boldsymbol{k}_1$) containing the descriptions for f_0, f_1 and the input mask shares r_0, r_1 respectively.
- $\mathtt{Eval}(j, \boldsymbol{k}_j, \hat{x}) \rightarrow f(x)$ is a polynomial-time deterministic algorithm that, given the party index j, the functional key k and the masked input $\hat{x}$ outputs an additive share $f_j(\hat{x})$, such that $f_0(\hat{x}) + f_1(\hat{x}) = f(x)$.

As central building block of many FSS gates, we recall the concept of Distributed Comparison Function (DCF) (Section 3 of [11]) to be a comparison function $f^{<}_{\alpha,\beta}$ outputting β if $x > \alpha$ and zero otherwise. Built on top of two evaluations of DCF, [11] later proposes a FSS gate for Interval Containment (IC) computing $f_{p,q}(x) = 1_{x \in [p,q]}$ (Section 4.1 of [11]). To compute the unit step function of a n-bit signed integer, it suffices to employ their construction (detailed in Figure 3 if [11]) setting $p = 0$ and $q = 2^{n-1} - 1$, obtaining $1_{p \leqslant x \leqslant q} = H_n(x)$. For convenience, we detail this FSS gate instantiation in Protocols 1 (key generation) and 2 (evaluation), keeping the DCF calls to the original protocol in [11].

Protocol 1 FSS.Gen$^{IC}(\lambda, n, r) \rightarrow \boldsymbol{k}_0^{IC}, \boldsymbol{k}_1^{IC}$

Players: P_{setup} carries out the generation.
Input: λ: computational security parameter.
r: Mask for the input to the function.
Output: $\boldsymbol{k}_0, \boldsymbol{k}_1$: preprocessing keys, to send to P_0, P_1 respectively.
$\langle \delta_{\boldsymbol{x}} \rangle, \langle \delta_{\boldsymbol{y}} \rangle$: δ-shares of input vectors, to send to $P_{in_{\boldsymbol{x}}}, P_{in_{\boldsymbol{y}}}$ (input owners) resp.

Note: All arithmetic operations (+,−,·) are defined in $\mathbb{Z}_{2^n}$, thus their results are susceptible to "overflow" due to modular reduction.

Define the interval $[p, q]$ *for sign extraction:*
$p \leftarrow 0; \qquad q \leftarrow 2^{n-1} - 1$
Generate a DCF for γ*, am arbitrary value above the two interval limits:*
$\gamma \leftarrow (2^n - 1)$
$(\boldsymbol{k}_{\gamma 0}, \boldsymbol{k}_{\gamma 1}) \leftarrow \mathsf{Gen}^{<}_{n}(1^\lambda, \gamma + r, 1, \mathcal{U}_{[\mathbb{Z}_{2^n}]})$
Generate the correction terms[2] *to fix overflows:*
$c \leftarrow 1_{p+r>q+r} + 1_{q+r+1>q+1} - 1_{p+r>p} + 1_{p+r=2^n-1}$
$c_0 \sim \mathcal{U}_{[\mathbb{Z}_{2^n}]}; \quad c_1 \leftarrow c - c_0$
Compose keys:
$\boldsymbol{k}_0^{IC} \leftarrow (\boldsymbol{k}_{\gamma 0}, c_0); \qquad \boldsymbol{k}_1^{IC} \leftarrow (\boldsymbol{k}_{\gamma 1}, c_1)$
return $\boldsymbol{k}_0^{IC}, \boldsymbol{k}_1^{IC}$

2.1.4 On security guarantees. This work focuses on 2PC with security against a semi-honest adversary non-adaptively corrupting at most one computing party. Also referred to as *Honest -but-Curious*, the computing parties P_j are to follow the protocol faithfully, while a party corrupted by the adversary will try to extract as much information as possible from his computation.

Employing simulation based security proofs [17, 37], previous works have proven SS and ΠSS to be perfectly information theoretic secure against computationally unbounded semi-honest adversaries [29, 49]. In contrast, FSS schemes FSS schemes rely on the security of the underlying PRG to be proven computationally secure against time bounded adversaries [11].

[2] The correction terms test three standard overflow cases and one corner case. The standard case terms test if $q+r$ overflows ($1_{p+r>q+r}$), if $q+r+1$ overflows ($1_{q+r+1>q+1}$), and if $p + r$ does not overflow ($1_{p+r>p}$, which is always 1 in our instantiation since $p = 0$ and $r < 2^n - 1$). The corner case term tests whether $p + r = 2^n - 1$ ($1_{p+r=2^n-1}$, yielding zero except if $r = 2^n - 1$ in our case). Proofs of the need of these correctness terms are given in [11].

Table 1: Reformulation of the distance metrics into a composition of local evaluations of f_{local} and the cross product $f_{cp} \cdot \boldsymbol{x}^T \boldsymbol{y}$

Distance Metric	Formula	$f_{local}(\boldsymbol{x}) + f_{local}(\boldsymbol{y}) + f_{cp} \cdot \boldsymbol{x}^T \boldsymbol{y}$	$f_{local}(\boldsymbol{v})$	f_{cp}
Scalar/Inner Product	$\sum \boldsymbol{x}^{(i)} \cdot \boldsymbol{y}^{(i)}$	$0 + 0 + 1 \sum(\boldsymbol{x}^{(i)} \cdot \boldsymbol{y}^{(i)})$	0	1
Hamming Distance	$\sum \boldsymbol{x}^{(i)} \oplus \boldsymbol{y}^{(i)}$	$\sum(\boldsymbol{x}^{(i)})^2 + \sum(\boldsymbol{y}^{(i)})^2 - 2\sum(\boldsymbol{x}^{(i)} \cdot \boldsymbol{y}^{(i)})$	$\sum(\boldsymbol{v})^2$	-2
Squared Euclidean	$\sum(\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)})^2$	$\sum(\boldsymbol{x}^{(i)})^2 + \sum(\boldsymbol{y}^{(i)})^2 - 2\sum(\boldsymbol{x}^{(i)} \cdot \boldsymbol{y}^{(i)})$	$\sum(\boldsymbol{v})^2$	-2
Squared Mahalanobis	$(\boldsymbol{x}-\boldsymbol{y})^T \boldsymbol{M} (\boldsymbol{x}-\boldsymbol{y})$	$\boldsymbol{x}^T \boldsymbol{M} \boldsymbol{x} + \boldsymbol{y}^T \boldsymbol{M} \boldsymbol{y} - 2\,(\boldsymbol{x}^T \boldsymbol{M}) \cdot \boldsymbol{y}$	$(\boldsymbol{v}^T \boldsymbol{M} \boldsymbol{v})$	-2

Protocol 2 $\text{FSS.Eval}^{IC}(j, \boldsymbol{k}_j, \hat{x}) \rightarrow o_0, o_1$

Players: P_j the selected computing party j.
Input: j: The party number, $j \in \{0, 1\}$.
$\boldsymbol{k}_j$: The key for P_j, composed of a DCF key for γ and a correction share c_j.
$\hat{x}$: Masked public input, result of reconstructing $x + r$.
Output: o_0, o_1: Additive secret shares of $1_{x \in [0, 2^{n-1}-1]}$.

Define the interval $[p, q]$ for sign extraction:
1: $p \leftarrow 0; \quad q \leftarrow 2^{n-1} - 1$
Deserialize key and obtain local overflow term η:
2: $(\boldsymbol{k}_{\gamma j}, c_j) \leftarrow \boldsymbol{k}_j$
3: $\eta \leftarrow 1_{\hat{x} > p} - 1_{\hat{x} > q+1}$
Evaluate the DCF with two inputs and compute result:
4: $o_j^L \leftarrow \mathtt{Eval}_n^{<}(j, \boldsymbol{k}_{\gamma j}, 1, \hat{x} - 1)$
5: $o_j^R \leftarrow \mathtt{Eval}_n^{<}(j, \boldsymbol{k}_{\gamma j}, 1, \hat{x} - q)$
6: **return** $o_j \leftarrow j \cdot \eta - o_j^L + o_j^R + c_j$

2.2 Thresholded distance metrics and applications

Inspired by GSHADE[15], we now introduce the thresholded distance metrics that we seek to protect in this work alongside motivating real-world applications:

- **Scalar Product**: $f_{SP}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y} = \sum_{i=1}^{n} \boldsymbol{x}^{(i)} \boldsymbol{y}^{(i)}$ is a common distance metric in face recognition where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ are two vectors of the same dimension. It is used to measure the similarity between two vectors.
- **Hamming Distance**: $f_{HD}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} (\boldsymbol{x}^{(i)} \oplus \boldsymbol{y}^{(i)})$ is a distance metric frequently used in information theory and computer science to measure the distance between two bitstrings. Besides its interest in iris and fingerprint recognition, it is the base of the perceptual hashing technique [47] used in image comparison, with applications ranging from image watermarking [30] to detection of Child Sexual Abuse Material (CSAM)[25].
- **Squared Euclidean Distance**: $f_{SED}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} (\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)})^2$ is a distance metric used in many machine learning applications, such as clustering [46]. It is also used in the context of face recognition [36].
- **Squared Mahalanobis Distance**: $f_{MD}(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^T \boldsymbol{M} (\boldsymbol{x} - \boldsymbol{y})$ is a distance metric used in many machine learning applications, such as clustering [46] and recognition of hand shape/keystrokes/signatures [15].

3 OUR SOLUTION

We now describe our solution for a lightweight and efficient 2PC distance metric with comparison, requiring a single round of communication and two ring elements in the online phase.

3.1 Distance Metrics

We start off by writing the generic function we wish to protect:

$$f(f_{dist}, \theta, \boldsymbol{x}, \boldsymbol{y}) = 1_{f_{dist}(\boldsymbol{x}, \boldsymbol{y}) \geqslant \theta} = \begin{cases} 1 & \text{if } f_{dist}(\boldsymbol{x}, \boldsymbol{y}) \geqslant \theta, \\ 0 & \text{if } f_{dist}(\boldsymbol{x}, \boldsymbol{y}) < \theta, \end{cases} \quad (3)$$

To adapt to 2PC, we reformulate the distance metrics f_{dist} from Section 2.2 as

$$z = f_{dist}(\boldsymbol{x}, \boldsymbol{y}) = f_{local}(\boldsymbol{x}) + f_{local}(\boldsymbol{y}) + f_{cp} \cdot \sum (\boldsymbol{x}^{(i)} \cdot \boldsymbol{y}^{(i)}) \quad (4)$$

where f_{local} is a function that can be computed locally by each input data holder, and f_{cp} is the "cross product" constant factor that applies to the scalar product evaluation present in all the metrics. Using this blueprint, we rewrite all the distance metrics in Table 1.

We remark that the Hamming Distance can be reformulated as the Squared Euclidean Distance as long as the input vectors are composed of binary values $\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \in \{0, 1\} \forall i$, since the boolean XOR operation between two binary values can be rewritten in the arithmetic domain as $\boldsymbol{x}^{(i)} \oplus \boldsymbol{y}^{(i)} = (\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)})^2$, the square of its difference.

3.2 Roles in 2PC scenario

Our solution is set in a two-party computation scenario, requiring two parties taking the role of "computing servers", yet there are more duties to cover. In total, we distinguish up to six different roles in our system and model them as player types:

- P_{setup}: The setup party is responsible for generating the preprocessing material during the offline phase, and distribute it to the parties involved in the online phase. The setup party must be trusted.
- P_0, P_1: Computing parties in the 2PC semihonest paradigm.
- P_{in_x}, P_{in_y}: The owners/holders of the input vectors, to be shared with the computing parties at the beginning of the online phase.
- P_{res}: The party that will receive the result of the full protocol execution.

These roles are not forcefully separate entities. In a strict 2PC scenario the computing parties will jointly perform the role of P_{setup}, and optionally even provide the inputs (e.g., $P_{in_x} \equiv P_0$ and $P_{in_y} \equiv P_1$).

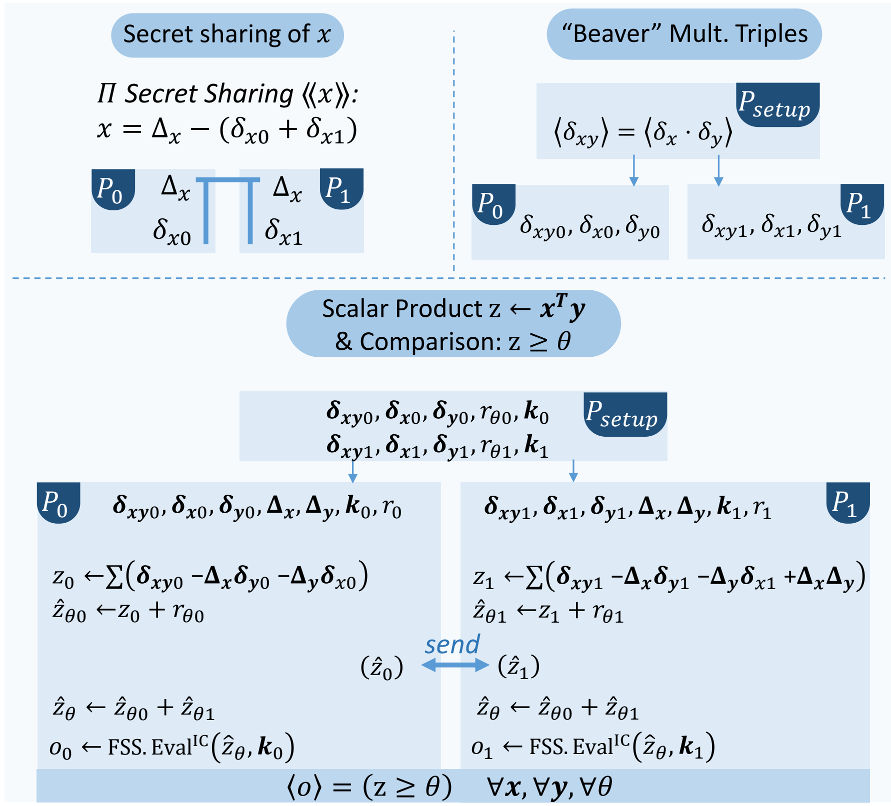

Figure 1: Overview of Funshade primitives

3.3 Sketching the solution

With the different roles in place, we are now ready to sketch our solution. In a nutshell, we combine Π-sharing to locally compute a scalar product with the FSS gate for interval containment from [11] with full correctness.

The key insight driving our design stems from the intermediate SS state in the Π-sharing multiplication (Equation 2). By providing Π-shared input vectors to the computing parties P_j, we can locally obtain the SS shares of the element-wise multiplication, and perform local cumulative addition to obtain shares of the scalar product result. Compared to the pure SS approach, we no longer need a round of communication to reconstruct the intermediate values $x-a$ and $y-b$ masked by Beaver triples (Mult. in Equation 1). As pointed out in ABY2.0 [49], the communication in a ΠSS multiplication gate happens at the output wires, as opposed to SS multiplication gates where the round of communication is tied to the input wires.

The subsequent FSS gate for interval containment requires a publicly reconstructed input held by both parties, which, to preserve the input data privacy, must be masked prior to its reconstruction (in line with previous FSS-based works [11, 14, 54]). Crucially, the masking of the private input via local shares addition followed by its reconstruction (at the cost of one round of communication) happens at the input wire of the FSS gate.

All we have left is to put together the two pieces of the puzzle. We can skip the Π-sharing reconstruction and instead add the input mask directly to the scalar product output, and then reconstruct this masked value to serve as public input for the FSS interval containment gate. Figure 1 depicts our idea applied to the scalar product metric.

To obtain the other metrics we would have each input data holder P_{in_x}, P_{in_y} run f_{local} on its inputs and secret share the result with the computing parties to add it to the output of the scalar product. In addition to that, both parties would multiply the shares of the scalar product result with the corresponding f_{cp}, resulting in the correct distance metric evaluation $z = f_{dist}(\boldsymbol{x}, \boldsymbol{y})$.

To keep the threshold θ hidden from the computing parties (and known only by P_{setup}), we subtract the value of θ from the additive random mask r during the offline/setup phase, employing an IC gate (Protocols 1 and 2) and then compute the of $z_\theta = z - \theta$.

3.4 Protocol specification

Embracing this combination of ΠSS for the locally computed scalar product and FSS for the comparison to θ, we can now outline each of the protocols that compose the full solution.:

(1) FUNSHADE.Setup (Protocol 3): P_{setup} generates the correlated randomness required for the scalar product multiplications, as well as the keys for the interval containment, and distributes the preprocessing material to the parties involved in the online phase.

Protocol 3 FUNSHADE.Setup(l, n, λ, θ) $\rightarrow \boldsymbol{k}_0, \boldsymbol{k}_1, \langle \delta_{\boldsymbol{x}} \rangle, \langle \delta_{\boldsymbol{y}} \rangle$

Players: P_{setup} carries out all the setup.
Input: l: length of the input vectors.
n: number of bits for the secret sharing ring $\mathbb{Z}_{2^n}$.
λ: security parameter.
θ: threshold for the comparison $\in \mathbb{Z}_{2^n}$.
Output: $\boldsymbol{k}_0, \boldsymbol{k}_1$: preprocessing keys, sent to P_0, P_1 respectively.
$\langle \delta_{\boldsymbol{x}} \rangle, \langle \delta_{\boldsymbol{y}} \rangle$: δ-shares of input vectors, sent to $P_{in_{\boldsymbol{x}}}, P_{in_{\boldsymbol{y}}}$ (input owners) resp.

Note: All arithmetic operations (+,–,·) are defined in $\mathbb{Z}_{2^n}$.

Beaver Triples for Π*-sharing scalar product:*
1: $\langle \boldsymbol{\delta}_x \rangle, \langle \boldsymbol{\delta}_y \rangle \equiv ((\boldsymbol{\delta}_{x_0}, \boldsymbol{\delta}_{x_1}),\ (\boldsymbol{\delta}_{y_0}, \boldsymbol{\delta}_{y_1})) \sim \mathcal{U}_{[\mathbb{Z}_{2^n}^{l\times 4}]}$
2: $\boldsymbol{\delta}_{xy_0} \sim \mathcal{U}_{[\mathbb{Z}_{2^n}^{l}]}$
$\boldsymbol{\delta}_{xy_1} \leftarrow (\boldsymbol{\delta}_{x_0} + \boldsymbol{\delta}_{x_1}) \cdot (\boldsymbol{\delta}_{y_0} + \boldsymbol{\delta}_{y_1}) - \boldsymbol{\delta}_{xy_0}$
$\langle \boldsymbol{\delta}_{xy} \rangle \equiv (\boldsymbol{\delta}_{xy_0}, \boldsymbol{\delta}_{xy_1})$
3: $\langle r \rangle \equiv (r_0, r_1) \sim \mathcal{U}_{[\mathbb{Z}_{2^n}^{\times 2}]}$ $\quad r \leftarrow r_0 + r_1$
$\langle r_\theta \rangle \equiv (r_{\theta 0}, r_{\theta 1}) \leftarrow (r_0, r_1 - \theta)$

FSS interval containment:
4: $\boldsymbol{k}_0^{IC}, \boldsymbol{k}_1^{IC} \leftarrow \text{FSS.Gen}^{IC}(\lambda, n, r)$
5: $\boldsymbol{k}_j \equiv (\boldsymbol{\delta}_{x_j}, \boldsymbol{\delta}_{y_j}, \boldsymbol{\delta}_{xy_j}, r_{\theta j}, \boldsymbol{k}_j^{IC}),\ j \in \{0, 1\}$

Dealing the preprocessing material :
6: SEND $\boldsymbol{k}_0 \Rightarrow P_0$, $\quad (\boldsymbol{\delta}_{\boldsymbol{x}_0}, \boldsymbol{\delta}_{\boldsymbol{x}_1}) \Rightarrow P_{in_{\boldsymbol{x}}}$
$\boldsymbol{k}_1 \Rightarrow P_1$, $\quad (\boldsymbol{\delta}_{\boldsymbol{y}_0}, \boldsymbol{\delta}_{\boldsymbol{y}_1}) \Rightarrow P_{in_{\boldsymbol{y}}}$

(2) FUNSHADE.Share (Protocol 4): $P_{in_{\boldsymbol{x}}}, P_{in_{\boldsymbol{y}}}$, the input holder players, prepare the Π-shares of their corresponding inputs using the correlated randomness and then send these shares to the computing parties P_0, P_1.

Protocol 4 FUNSHADE.Share($\boldsymbol{v}$, $\boldsymbol{\delta}_{\boldsymbol{v}0}$, $\boldsymbol{\delta}_{\boldsymbol{v}1}$) $\rightarrow \Delta_v, \langle d_{\boldsymbol{v}} \rangle$

Players: $P_{in_{\boldsymbol{v}}}$, holding the input vector $\boldsymbol{v}$ (where $\boldsymbol{v} \in \{\boldsymbol{x}, \boldsymbol{y}\}$).
Input: $\boldsymbol{v}$: input vector $\in \mathbb{Z}_{2^n}^l$ held by $P_{in_{\boldsymbol{v}}}$.
$\boldsymbol{\delta}_{\boldsymbol{v}j}$: Precomputed δ-shares $\in \mathbb{Z}_{2^n}^l$.
Output: Δ_v: Δ-shares of vector $\boldsymbol{v}$ distributed to both P_0 & P_1.
$d_{\boldsymbol{v}_j}$: Arithmetic shares of the local computation $f_{local}(\boldsymbol{v})$.

1: $\Delta_v \leftarrow (\boldsymbol{v} + \boldsymbol{\delta}_{\boldsymbol{v}0} + \boldsymbol{\delta}_{\boldsymbol{v}1})$
2: $d_{\boldsymbol{v}} \leftarrow f_{local}(\boldsymbol{v})$; $\quad \langle d_v \rangle \equiv (d_{\boldsymbol{v}0}, d_{\boldsymbol{v}1}) \leftarrow (\sim \mathcal{U}_{[\mathbb{Z}_{2^n}]}, d_{\boldsymbol{v}} - d_{\boldsymbol{v}0})$
3: SEND $(\Delta_v, d_{\boldsymbol{v}_0}) \Rightarrow P_0$, $\quad (\Delta_v, d_{\boldsymbol{v}_1}) \Rightarrow P_1$

(3) FUNSHADE.Eval (Protocol 5): P_0, P_1 engage in an online protocol upon acquiring the Π-shares of both inputs, using local multiplication and addition to compute the scalar product, and then evaluate the interval containment FSS scheme to determine whether the result is below the threshold θ.
(4) FUNSHADE.Result (Protocol 6): P_0, P_1 send the arithmetic shares of the result to the player designed to receive the output P_{res} for its reconstruction.

Protocol 5 FUNSHADE.Eval($j, \Delta_{\boldsymbol{x}}, \Delta_{\boldsymbol{y}}, \langle d_{\boldsymbol{x}} \rangle, \langle d_{\boldsymbol{y}} \rangle, \boldsymbol{k}_j) \rightarrow \langle o \rangle$

Players: $P_j,\ j \in \{0, 1\}$ computing parties.
Input: $\Delta_{\boldsymbol{x}}, \Delta_{\boldsymbol{y}}$: Δ-shares of $\langle\!\langle \boldsymbol{x} \rangle\!\rangle, \langle\!\langle \boldsymbol{y} \rangle\!\rangle$ (Π-shared inputs $\boldsymbol{x}, \boldsymbol{y}$) held by both P_0 and P_1.
$\langle d_{\boldsymbol{x}} \rangle, \langle d_{\boldsymbol{y}} \rangle$: Arithmetic shares of locally computed single-input terms $f_{local}(\boldsymbol{x}), f_{local}(\boldsymbol{y})$ of $f_{dist}(\boldsymbol{x}, \boldsymbol{y})$.
$\boldsymbol{k}_j$: preprocessing keys from FUNSHADE.Setup containing:
$\boldsymbol{\delta}_{\boldsymbol{x}_j}, \boldsymbol{\delta}_{\boldsymbol{y}_j}$: δ-shares of Π-shared input vectors $\boldsymbol{x}, \boldsymbol{y}$,
$\boldsymbol{\delta}_{\boldsymbol{xy}_j}$: arith. shares of Beaver triple s.t. $\langle \boldsymbol{\delta}_x \rangle \langle \boldsymbol{\delta}_y \rangle = \langle \boldsymbol{\delta}_{xy} \rangle$,
$r_{\theta j}$: arith. shares of FSS input mask r minus threshold θ,
$\boldsymbol{k}_j^{IC}$: FSS key for the IC gate of [11].
Output: $\langle o \rangle$: arithmetic shares of the result $o = f(\boldsymbol{x}, \boldsymbol{y}) \geq \theta$.

Note: All steps apply to both computing parties $P_j,\ j \in \{0, 1\}$. All arithmetic operations (+,–,·) are defined in $\mathbb{Z}_{2^n}$.

Π*-sharing based scalar product:*
1: $\hat{z}_{\theta j} \leftarrow r_{\theta j} + d_{\boldsymbol{x}_j} + d_{\boldsymbol{y}_j} + f_{cpf} \cdot \sum^l [j \cdot \Delta_{\boldsymbol{x}} \cdot \Delta_{\boldsymbol{y}} - \Delta_{\boldsymbol{x}} \cdot \delta_{\boldsymbol{y}_j} - \Delta_{\boldsymbol{y}} \cdot \delta_{\boldsymbol{x}_j} + \delta_{\boldsymbol{xy}_j}]$
Reconstruction of masked input to FSS gate:
2: P_j: SEND $\hat{z}_{\theta j} \Rightarrow P_{1-j}$; $\quad \hat{z}_\theta \leftarrow \hat{z}_{\theta 0} + \hat{z}_{\theta 1}$
Interval Containment for sign extraction:
3: $o_j \leftarrow \text{FSS.Eval}^{IC}(j, \boldsymbol{k}_j^{IC}, \hat{z}_\theta)$
4: **return** o_j

Protocol 6 FUNSHADE.Result($\langle o \rangle$) $\rightarrow o$

Players: $P_j,\ j \in \{0, 1\}$ computing parties, P_{res} result holder.
Input: $\langle o \rangle$: secret shares $o_0, o_1 \in \mathbb{Z}_{2^n}$ of the result o held by P_0, P_1.
Output: o: Output value.

1: P_j: SEND $o_j \Rightarrow P_{res}$.
2: P_{res}: $o \leftarrow (o_0 + o_1)$

3.5 Applications and Practical considerations

We display a diagram of our solution applied to biometrics/CSAM detection in Figure 2. The FUNSHADE protocol can be easily computed in parallel for different inputs $\boldsymbol{y}$ in cases where the reference database contains more than one record, such as CSAM detection against a large database of hashes or biometric identification against multiple subjects. Additionally, these use-cases normally gather their reference databases ahead of time. To speed up the online phase, the reference database held by party $P_{in_{\boldsymbol{y}}}$ could be Π-shared as part of the offline phase, leaving only the live input to be shared in the online phase. In addition, biometric identifications / CSAM detections might output one single bit to determine whether there is a match in the entire database. In this case the individual secret shared outputs $\boldsymbol{o}_j^{(i)}$ could be locally summed up to yield a single number as output.

As an alternative to the trusted setup carried by P_{setup}, the two computing parties P_0, P_1 could follow an interactive protocol in the offline phase to jointly realize the role of P_{setup} (execution of FUNSHADE.Setup and distribution of key material), resorting to distributed generation via generic 2PC techniques for the FSS gate key generation (Appendix A.2 of [11]), and either Oblivious

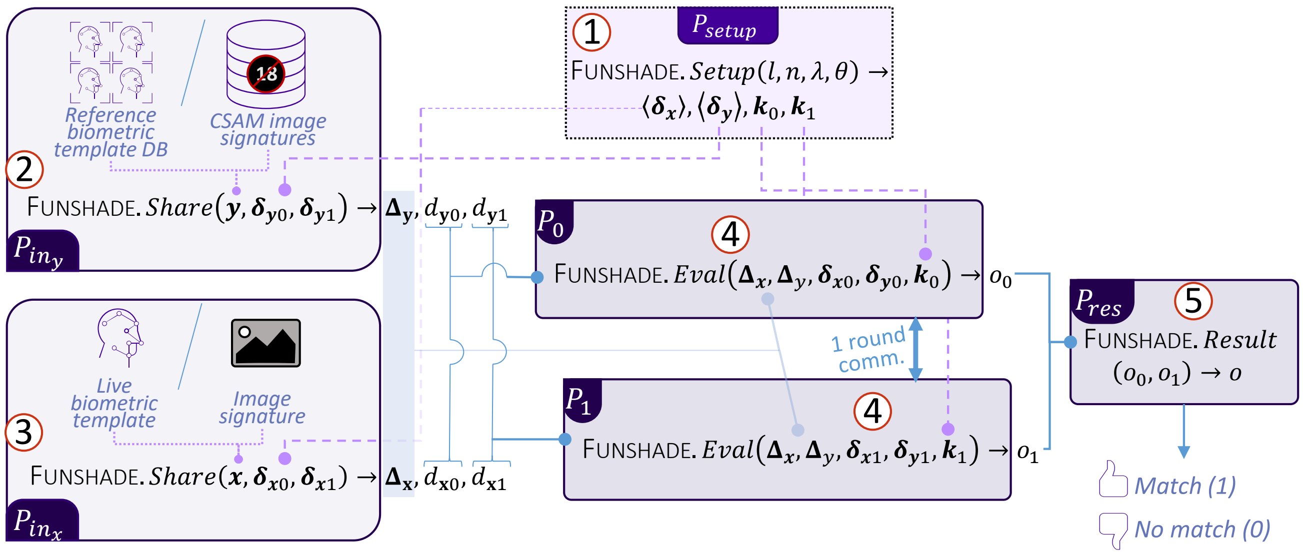

Figure 2: Diagram of our Funshade protocol applied to biometrics/CSAM detection

Transfer or Homomorphic Encryption for the ΠSS scalar product preprocessing material (Section 3.1.3 of [49]). [3]

3.6 Security Analysis

We consider security against a Honest-but-Curious adversary $\mathcal{A}$ that corrupts up to one of the two computing parties P_j. We consider a static corruption model where the adversary must choose which participant to corrupt before the execution of the computations. This is a standard security model in previous MPC frameworks [11, 19, 29, 48, 49, 54]. Under this threat model, we define and later prove the security and correctness of our constructions.

We employ the standard real world - ideal world paradigm, providing the simulation for the case of a corrupt P_j. The ideal world simulation contains an additional trusted party that receives all the inputs from P_0, P_1, computes the ideal functionality correctly and sends the corresponding results back to P_0, P_1. Conversely, the real world simulation executes the protocol as described in the FUNSHADE algorithms in the presence of $\mathcal{A}$.

Our security proof works in the $\mathcal{F}_{\text{FUNSHADE}.setup}$-hybrid model where $\mathcal{F}_{\text{FUNSHADE}.setup}$ represents the ideal functionality corresponding to protocol FUNSHADE.setup.

DEFINITION 1 (SECURITY OF FUNSHADE). *For each $j \in \{0, 1\}$, there is a PPT algorithm $\mathcal{S}$ (simulator) such that $\forall \theta \in \mathbb{Z}^*_{n+}, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}^l_n$ and every function $f_{dist}(\boldsymbol{x}, \boldsymbol{y}) : \mathbb{Z}^l_n \rightarrow \mathbb{Z}_n$ from Table 1, $\mathcal{S}$ realizes the ideal functionality $\mathcal{F}_{th-dist}$, such that its behavior is computationally indistinguishable from a real world execution of protocols 4-5-6 in the presence of a semi-honest adversary $\mathcal{A}$.*

Ideal Functionality $\mathcal{F}_{th-dist}$

$\mathcal{F}_{th-dist}$ interacts with the parties P_0, P_1 and the adversary $\mathcal{S}$ and is parametrized by a publicly know function $f_{dist}(\boldsymbol{x}, \boldsymbol{y})$ and a threshold θ.

- ***Inputs***: $\mathcal{F}_{th-dist}$ receives the inputs $\Delta_{\boldsymbol{x}}, \Delta_{\boldsymbol{y}}, \delta_{\boldsymbol{x}_j}, \delta_{\boldsymbol{y}_j}$ from the computing parties P_0, P_1.
- ***Computation***: $\mathcal{F}_{th-dist}$ reconstructs $\boldsymbol{x} = \Delta_{\boldsymbol{x}} - (\delta_{\boldsymbol{x}_0} + \delta_{\boldsymbol{x}_1})$ and $\boldsymbol{y} = \Delta_{\boldsymbol{y}} - (\delta_{\boldsymbol{y}_0} + \delta_{\boldsymbol{y}_1})$, computes $z = f_{dist}(\boldsymbol{x}, \boldsymbol{y})$ and $o = 1_{z \geqslant \theta}$.
- ***Output***: Sends o_j to P_{res}.

THEOREM 1. *In the $\mathcal{F}_{\text{FUNSHADE}.setup}$-hybrid model, protocols 4-5-6 (online phase) securely realize the functionality $\mathcal{F}_{th-dist}$.*

PROOF. The semi-honest adversary corrupts P_j during the sequential execution of protocols 4-5-6. For this case, $\mathcal{S}$ executes the setup phase honestly on the behalf of P_{1-j} (in case of interactive setup), and will simulate the entire circuit evaluation, assuming the circuit-inputs of P_{1-j} to be 0. In the FUNSHADE.Result protocol, $\mathcal{S}$ adjusts the shares of $\langle o \rangle$ on behalf of P_{1-j} so that $\mathcal{A}$ sees the same transcript as in the real-world protocol.

- FUNSHADE.Setup: For the offline phase, we consider it as an ideal functionality $\mathcal{F}_{\text{FUNSHADE}.setup}$, which generates the required FSS preprocessing keys and δ-shares. Since we make only black-box access to FUNSHADE.setup, its simulation follows from the security of the underlying primitive used to instantiate it (OT or HE for the ΠSS preprocessing material stemming from setupMULT of [49], generic 2PC for the FSS keys following Appendix A.2 of [11]), or alternatively a trusted party can be used.
- FUNSHADE.Share: For the instances where P_j is the owner of the values (e.g., $P_j \equiv P_{in_x}$), $\mathcal{S}$ has to do nothing since $\mathcal{A}$ is

[3] That being said, the trusted setup might be justified in a context of biometrics/CSAM detection. Not trusting the reference database would immediately defeat the purpose of the system. Hence, the system must trust the entity in possession of the reference database (e.g., P_{in_Y}), and thus this entity could naturally play the role of P_{setup}.

not receiving any messages. $\mathcal{S}$ receives $\Delta_{\boldsymbol{v}}$ from $\mathcal{A}$ on behalf of P_{1-j}. For the instances where P_{1-j} is the owner, $\mathcal{S}$ sets $v = 0$ and performs the protocol steps honestly.
- Funshade.Eval: During the online phase, $\mathcal{S}$ follows the protocol steps honestly using the data obtained from the setup phase. The scalar product requires l local additions (non-interactive and thus they don't need to be simulated) and a subsequent reconstruction of $\langle \hat{z}_\theta \rangle$ as $\hat{z}_\theta = \hat{z}_{\theta 0} + \hat{z}_{\theta 1}$ that behaves just like Funshade.Result and serves as input to the FSS IC gate. For the FSS IC gate, we resort to the Simulation-based security of [11] (Definition 2) to argue computational indistinguishability of the ideal and real world executions, hiding the information of r contained in $\boldsymbol{k}_0$ and $\boldsymbol{k}_1$ from $\mathcal{A}$.
- Funshade.Result: To reconstruct a value $\langle o \rangle$, $\mathcal{S}$ is given the output o, which is the output of $\mathcal{A}$. Using o and the share o_{1-j} corresponding to P_{1-j}, $\mathcal{S}$ computes $o_j = o - o_{1-j}$ and sends this to $\mathcal{A}$ on behalf of P_{1-j}. $\mathcal{S}$ receives o_j from $\mathcal{A}$ on behalf of P_{1-j}.

□

Definition 2 (Correctness of Funshade). *For every threshold $\theta \in \mathbb{Z}^*_{n+}$, every pair of input vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}^l_n$ and every function $f_{dist}(\boldsymbol{x}, \boldsymbol{y}) : \mathbb{Z}^l_n \rightarrow \mathbb{Z}_n$ from Table 1,*

$$\begin{aligned} &\textit{if } (\boldsymbol{k}_0, \boldsymbol{k}_1, \langle \delta_{\boldsymbol{x}} \rangle, \langle \delta_{\boldsymbol{y}} \rangle) \leftarrow \text{Funshade.Gen}(l, n, \lambda, \theta) \\ &\textit{and } (\Delta_{\boldsymbol{x}}, \langle d_{\boldsymbol{x}} \rangle \leftarrow \text{Funshade.Share}(\boldsymbol{x}, \langle \delta_{\boldsymbol{x}} \rangle), \\ &\quad \Delta_{\boldsymbol{y}}, \langle d_{\boldsymbol{y}} \rangle \leftarrow\leftarrow \text{Funshade.Share}(\boldsymbol{y}, \langle \delta_{\boldsymbol{y}} \rangle)) \\ &\textit{then } \Pr[\text{Funshade.Eval}(0, \Delta_{\boldsymbol{x}}, \Delta_{\boldsymbol{y}}, d_{\boldsymbol{x}_0}, d_{\boldsymbol{y}_0}, \boldsymbol{k}_0) \\ &\quad + \text{Funshade.Eval}(1, \Delta_{\boldsymbol{x}}, \Delta_{\boldsymbol{y}}, d_{\boldsymbol{x}_1}, d_{\boldsymbol{y}_1}, \boldsymbol{k}_1) \\ &\quad = 1_{f_{dist}(\boldsymbol{x},\boldsymbol{y}) \geqslant \theta}] = 1. \end{aligned} \tag{5}$$

Theorem 2. *Jointly, protocol 3 (offline phase), and protocols 5-4-6 (online phase), realize the function $f(f_{dist}, \theta, \boldsymbol{x}, \boldsymbol{y}) = 1_{f_{dist}(\boldsymbol{x},\boldsymbol{y}) \geqslant \theta}$ correctly.*

Proof. We first decompose the Π-sharing based scalar product (step 1 of Protocol 5) for the joint result of the two computing parties $\hat{z}_\theta$ in Equation 6,

$$\begin{aligned} \hat{z}_\theta &= \hat{z}_{\theta 0} + \hat{z}_{\theta 1} = (r_{\theta 0} + r_{\theta 1}) + (d_{\boldsymbol{x}_0} + d_{\boldsymbol{x}_1}) + (d_{\boldsymbol{y}_1} + d_{\boldsymbol{y}_1}) + f_{cp} \cdot \textstyle\sum^l \\ &\quad [\Delta_{\boldsymbol{x}}\Delta_{\boldsymbol{y}} - (\Delta_{\boldsymbol{x}}\delta_{\boldsymbol{y}_0} + \Delta_{\boldsymbol{x}}\delta_{\boldsymbol{y}_1}) - (\Delta_{\boldsymbol{y}}\delta_{\boldsymbol{x}_0} + \Delta_{\boldsymbol{y}}\delta_{\boldsymbol{x}_1}) + (\delta_{\boldsymbol{xy}_0} + \delta_{\boldsymbol{xy}_1})] \\ &= r_\theta + d_{\boldsymbol{x}} + d_{\boldsymbol{y}} + f_{cp} \cdot \textstyle\sum^l [\Delta_{\boldsymbol{x}}\Delta_{\boldsymbol{y}} - \Delta_{\boldsymbol{x}}\delta_{\boldsymbol{y}} - \Delta_{\boldsymbol{y}}\delta_{\boldsymbol{x}} + \delta_{\boldsymbol{xy}}] \\ &= r - \theta + d_{\boldsymbol{x}} + d_{\boldsymbol{y}} + f_{cp} \cdot \textstyle\sum^l [\Delta_{\boldsymbol{x}}\Delta_{\boldsymbol{y}} - \Delta_{\boldsymbol{x}}\delta_{\boldsymbol{y}} - \Delta_{\boldsymbol{y}}\delta_{\boldsymbol{x}} + \delta_{\boldsymbol{x}}\delta_{\boldsymbol{y}}] \\ &= r - \theta + d_{\boldsymbol{x}} + d_{\boldsymbol{y}} + f_{cp} \cdot \textstyle\sum^l (\Delta_{\boldsymbol{x}} - \delta_{\boldsymbol{x}}) \cdot (\Delta_{\boldsymbol{y}} - \delta_{\boldsymbol{y}}) \\ &= r - \theta + f_{local}(\boldsymbol{x}) + f_{local}(\boldsymbol{y}) + f_{cp} \cdot \textstyle\sum^l (\boldsymbol{x}^{(i)} \cdot \boldsymbol{y}^{(i)}) \\ &= r - \theta + f_{dist}(\boldsymbol{x}, \boldsymbol{y}) = z_\theta + r \end{aligned} \tag{6}$$

where we group all the SS shares and reconstruct their original values, replace r_θ and $\delta_{\boldsymbol{xy}}$ by the corresponding values (from definitions in protocol 1), group the Π-shares of $\boldsymbol{x}$ and $\boldsymbol{y}$ to later reconstruct their values, and finally make use of Equation 4.

With the public input $\hat{z}$ sorted out, we analyze the Interval Containment evaluation with output reconstruction in Equation 7,

$$\begin{aligned} o = o_1 + o_2 &= \text{FSS.Eval}^{IC}(0, \boldsymbol{k}^{IC}_0, \hat{z}_\theta) + \text{FSS.Eval}^{IC}(1, \boldsymbol{k}^{IC}_1, \hat{z}_\theta) \\ &= \text{FSS.Eval}^{IC}(0, \text{FSS.Gen}^{IC}(\lambda, n, r)^{(0)}, z_\theta + r) \\ &\quad + \text{FSS.Eval}^{IC}(1, \text{FSS.Gen}^{IC}(\lambda, n, r)^{(1)}, z_\theta + r) \\ &= 1_{z_\theta \in \mathbb{Z}^*_{n^+}} = 1_{0 \leqslant z - \theta} = 1_{f_{dist}(\boldsymbol{x},\boldsymbol{y}) \geqslant \theta} \end{aligned} \tag{7}$$

where we resort to Theorem 3 of [11] to argue that the two protocols (FSS.Gen$^{IC}(\lambda, n, r)$, FSS.Eval$^{IC}(j, \boldsymbol{k}^{IC}_j, \hat{z}_\theta)$) constitute an FSS gate[4] correctly realizing $f(z_\theta) = 1_{p \leq z_\theta \leq q}$. Then, following Definition 2 (Correctness) of [11], we can argue that $\Pr[\text{FSS.Eval}^{IC}(0, \boldsymbol{k}^{IC}_0, \hat{z}_\theta) + \text{FSS.Eval}^{IC}(1, \boldsymbol{k}^{IC}_1, \hat{z}_\theta) = 1_{0 \leq z_\theta \leq 2^{n-1}-1}] = 1$, thus equating the output of the FSS gate to $1_{z_\theta \in \mathbb{Z}^*_{n^+}}$, the unit step function. □

3.7 Implementation

We implement our solution in a standalone Python library with efficient C++ blocks by virtue of Cython. Our code is available at https://github.com/ibarrond/funshade. We use a Miyaguchi-Preneel one-way compression function with an AES block cipher for our PRG construction, an extended variant of Matyas-Meyer-Oseas function used in previous works [54]. We concatenate several fixed key block ciphers to achieve the desired output length.

We timed the execution of Funshade.Eval in a single computing party to 900μs with one single core (Processor AMD Ryzen 5 PRO 3500U, 2100 Mhz, 4 Cores available), and around 550μs when using two cores to speed up the Interval Containment evaluation (one DCF per core). This indicates that the communication latency (e.g., 10ms for LAN, 70ms for WAN) would be the main bottleneck in a real-world deployment for 1:1 distance calculations, and there would be a wide margin to compensate communication with computation in 1:N or M:N scenarios (e.g., biometric identification). We also vary the vector sizes ranging from $l = 64$ to $l = 65536$, with negligible impact to the Funshade.Eval time, indicating that the main bottleneck in terms of computation is located in the evaluation of DCFs.

Additionally, we test our solution with randomized input vectors for all distance metrics, verifying the 100% correctness as long as natural overflows ($z > 2^{n-1} - 1$ or $z < -2^{n-1}$) are avoided.

4 PREVIOUS WORK

Distance metric evaluations, specially for Hamming Distance and Scalar Products, range among the most typical applications of privacy-preserving computation techniques. Consequently, a wide variety of previous work in MPC, FHE and FE have dealt with some form of it.

The Multi Party Computation field includes a plethora of works covering distance metric evaluations. All the frameworks for privacy preserving neural networks cover scalar-product-based matrix multiplications often followed by ReLU activations [8, 26, 43, 52, 59], covering a mixture of Garbled Circuits, Secret Sharing and their conversions. Secure hamming distance evaluation has motivated

[4]There are several notation elements to adapt in order to align with [11]. Our mask r is written as r^{in} in Figure 3 of [11] depicting the FSS IC gate. We set the parameters $p = 0$ and $q = 2^{n-1} - 1$ to define the interval containing all positive integers. $1_{p \leq z_\theta \leq q} = g_{IC,n,p,q}(z_\theta)$ is a function that belongs (per definition of IC gate in Section 4 of [11]) to the family of functions $\mathcal{G}^{IC}_{n,p,q}$ referenced in Theorem 3 of [11].

Table 2: Benchmark of theoretical costs on evaluating a scalar product and comparison to threshold between two vectors of l $n-bit$ integers

Work	Type	#Rounds of communication	#ring elements in communication	Correctness	Online Computation Blocks
AriaNN [54]	2PC SS: Arith., FSS	2 (1+1)	$4l+4$	N	SS scalar product, FSS Comparison (1 DCF)
Boyle et. al.[11]	2PC SS: Arith., FSS	2 (1+1)	$4l+4$	Y	SS scalar product, FSS IC gate (2 DCF)
ABY[29]	2PC SS: Boolean&Arith, GC	3 (1+2+0)	$\gg 6l$	Y	SS scalar product, Arith. to Yao conversion, GC evaluation
ABY2.0[49]	2PC ΠSS: Boolean&Arith.	5 (1+1+3)	$\gg 2$	Y	ΠSS scalar product, Arith. to Boolean conversion, BitExtraction
GSHADE[15] (only scalar prod.)	2PC OT	2	$> 2l$	Y	correlated OTs.
CryptFlow2[51]	2PC SS: Arith., OT	5	$> (128+14)l$	Y	Linear layer (1-dim weights), dReLU
Falcon[59]	3PC Replicated SS: Arith.	8 (1+7)	> 6	Y	MatMult with 1-dim matrices, Private Compare
FUNSHADE (OURS)	**2PC** **ΠSS: Arith., FSS**	**1**	**2**	**Y**	**ΠSS scalar product,** **FSS IC gate (2 DCF)**

work such as [16] based on Oblivious Transfer, with its generalization to multiple metrics in [15]. Mixed-mode protocols have also tackled distance evaluations [29, 48, 49]. However, the majority of these solutions incur in a considerable communication cost to perform comparison. More recently, solutions based on FSS [11, 14, 54] have shown promising results, leading to this work.

In the field of Homomorphic Encryption, the biometrics use-case has led to a variety of approaches, including [4, 45] for hamming distance or [61] for scalar product. However, these approaches do not include comparison to a threshold, and often rely on costly cryptographic primitives that make them slow.

Since the advent of Functional Encryption [9], scalar product and hamming distance have been the most suitable candidates to study. Inner Product Encryption (IPE) started off with selective security in [1], already envisioning biometric use-cases, and reaching full security with [28] and [57]. [42] applied FE to biometric authentication with hamming distance and to nearest-neighbor search on encrypted data; [44] employs IPE for hamming-weight based matchings of real-world iris templates. [41] and [39] are the latest iterations of privacy-preserving scalar product techniques based on FE, demonstrating performances in the order of hundreds of *ms* for vectors of 128 values. While FE does not require an extra operation after the "evaluation" to retrieve the result, these schemes scale polynomially with the input vector length (thus are unsuitable for very large vectors), and their computation does not include comparison to a threshold. To include it, one must resort to techniques such as Threshold Predicate Encryption [62].

There also exist techniques in the literature not resorting to these three main fields, such as [63] with a custom scheme, or [55] with Identity Based Encryption.

We compare the online phase performance of our solution with that of selected previous works in Table 1. FUNSHADE is the first work in the 2PC setting requiring one single round of communication to evaluate $1_{\boldsymbol{x}^T\boldsymbol{y}>\theta}$ while also presenting the lowest communication size of 2 ring elements. An additional side-by-side comparison with AriaNN [54] is provided in Appendix A.

On the importance of the threshold comparison in privacy-preserving distance metrics. The security provided by our construction, and that of all privacy-preserving techniques in general (MPC, FHE, FE), does not prevent the reconstructed outputs $o = f(\boldsymbol{x}, \boldsymbol{y})$ from revealing information about the inputs $\boldsymbol{x}, \boldsymbol{y}$. Indeed, P_{res} can leverage on his knowledge about the function being computed and attempt to extract information about the inputs from the outputs by inverting the function being computed $Leak(\boldsymbol{x}, \boldsymbol{y}) \leftarrow Leak(f^{-1}(o))$. Labeled as "input leakage" in previous works [39], this leakage affects the practical privacy of real-world deployments of privacy-preserving solutions. Applications using distance metric calculations as one of many building blocks (e.g., Machine Learning) might be more naturally protected thanks to the complexity of the function (beware! black-box model extraction attacks are real [58]), yet applications requiring only one distance metric evaluation (e.g., biometric matching, CSAM detection) are much more sensitive to this leakage, since these distance metrics are linear functions and thus easily invertible.

While solutions exist to add controlled noise to the input (e.g., Differential Privacy in [18]), the most straightforward method to reduce this leakage is to output the least information possible. For applications like biometric matching and CSAM detection, one-bit outputs suffice to determine whether there is a match or not, and hence performing the comparison in a privacy-preserving manner reduces considerably the input leakage of the construction. As such, FHE and FE-based solutions without privacy-preserving threshold comparison are more risky to apply in real-world scenarios than threshold-enabled solutions that MPC (ours included) offers out of the shelf.

5 CONCLUSIONS

In this work we presented Funshade, a novel 2PC privacy preserving solution of various distance metrics (e.g., Hamming distance, Scalar Product) followed by threshold comparison. We build our protocols upon ΠSS, a version of arithmetic secret sharing optimized for the secure evaluation of scalar products, and functional secret sharing with 100% correctness for comparison. Thanks to this, Funshade proposes the first solution in the 2PC literature requiring one single round of communication in the online phase while outperforming all previous works in online communication size (two ring elements), all while relying on lightweight cryptographic primitives. We implement our solution from scratch in a standalone Python/C++ library, and test it to record a runtime of less than 1ms per computing party excluding communication costs.

REFERENCES

[1] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. 2015. Simple functional encryption schemes for inner products. In *IACR International Workshop on Public Key Cryptography*. Springer, USA, 733–751.

[2] Shashank Agrawal and David J Wu. 2017. Functional encryption: deterministic to randomized functions from simple assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, France, 30–61.

[3] Manuel Barbosa, Dario Catalano, Azam Soleimanian, and Bogdan Warinschi. 2019. Efficient function-hiding functional encryption: From inner-products to orthogonality. In *Cryptographers' Track at the RSA Conference*. Springer, USA, 127–148.

[4] Mauro Barni, Tiziano Bianchi, Dario Catalano, Mario Di Raimondo, Ruggero Donida Labati, Pierluigi Failla, Dario Fiore, Riccardo Lazzeretti, Vincenzo Piuri, Alessandro Piva, et al. 2010. A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates. In *2010 Fourth IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS)*. IEEE, USA, 1–7.

[5] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*. Springer, Germany, 420–432.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 2019. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. ACM, New York, NY, USA, 351–371.

[7] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. 2020. Secure large-scale genome-wide association studies using homomorphic encryption. *Proceedings of the National Academy of Sciences of the United States of America* 117, 21 (26 May 2020), 11608–11613. https://doi.org/10.1073/pnas.1918257117

[8] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. 2020. MP2ML: A mixed-protocol machine learning framework for private inference. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. Association for Computing Machinery, Ireland, 1–10.

[9] Dan Boneh, Amit Sahai, and Brent Waters. 2011. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*. Springer, USA, 253–273.

[10] Florian Bourse. 2017. *Functional encryption for inner-product evaluations*. Ph. D. Dissertation. PSL Research University.

[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, October 17–21, 2021, Proceedings, Part II*. Springer, Croatia, 871–900.

[12] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function secret sharing. In *EUROCRYPT*. Springer, Bulgaria, 337–367.

[13] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Austria, 1292–1303.

[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2019. Secure Computation with Preprocessing via Function Secret Sharing. In *17th International Conference on Theory of Cryptography, TCC 2019*. Springer, Germany, 341–371.

[15] Julien Bringer, Herve Chabanne, Melanie Favre, Alain Patey, Thomas Schneider, and Michael Zohner. 2014. GSHADE: Faster privacy-preserving distance computation and biometric identification. In *Proceedings of the 2nd ACM workshop on Information hiding and multimedia security*. ACM, Austria, 187–198.

[16] Julien Bringer, Hervé Chabanne, and Alain Patey. 2013. Shade: Secure hamming distance computation from oblivious transfer. In *International Conference on Financial Cryptography and Data Security*. Springer, Japan, 164–176.

[17] Ran Canetti. 2000. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY* 13, 1 (2000), 143–202.

[18] Mahawaga Arachchige Pathum Chamikara, Peter Bertok, Ibrahim Khalil, Dongxi Liu, and Seyit Camtepe. 2020. Privacy preserving face recognition utilizing differential privacy. *Computers & Security* 97 (2020), 101951.

[19] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. 2019. EzPC: programmable and efficient secure two-party computation for machine learning. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Sweden, 496–511.

[20] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. 2019. ASTRA: high throughput 3pc over rings with application to secure prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*. ACM, UK, 81–92.

[21] David Chaum, Claude Crépeau, and Ivan Damgard. 1988. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, USA, 11–19.

[22] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast Private Set Intersection from Homomorphic Encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1243–1255. https://doi.org/10.1145/3133956.3134061

[23] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*. Springer International Publishing, China, 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[24] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. 2019. Numerical method for comparison on homomorphically encrypted numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, Japan, 415–445.

[25] European Commission. 2022. Proposal for a regulation laying down rules to prevent and Combat Child Sexual abuse. https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=COM:2022:209:FIN

[26] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2020. *Fantastic four: Honest-majority four-party secure computation with malicious security*. Technical Report. Cryptology ePrint Archive, Report 2020/1330, 2020. https://eprint. iacr. org ....

[27] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. 2017. The TinyTable protocol for 2-party secure computation, or: gate-scrambling revisited. In *Annual International Cryptology Conference*. Springer, USA, 167–187.

[28] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. 2016. Functional encryption for inner product with full function privacy. In *Public-Key Cryptography–PKC 2016*. Springer, Taiwan, 164–195.

[29] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In *22nd Annual Network and Distributed System Security Symposium, NDSS*. Usenix, San Diego, CA, USA, 15.

[30] Ling Du, Anthony TS Ho, and Runmin Cong. 2020. Perceptual hashing for image authentication: A survey. *Signal Processing: Image Communication* 81 (2020), 115713.

[31] David Evans, Yan Huang, Jonathan Katz, and Lior Malka. 2011. Efficient privacy-preserving biometric identification. In *Proceedings of the 17th conference Network and Distributed System Security Symposium, NDSS*, Vol. 68. Usenix, USA, 90–98.

[32] 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.

[33] J Fan and F Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive* 2012, 144 (2012), 29. https://eprint.iacr.org/2012/144

[34] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. 2017. Privacy-Preserving Distributed Linear Regression on High-Dimensional Data. *Proc. Priv. Enhancing Technol.* 2017, 4 (2017), 345–364.

[35] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Vol. 20. Stanford, USA.

[36] Babak Poorebrahim Gilkalaye, Ajita Rattani, and Reza Derakhshani. 2019. Euclidean-distance based fuzzy commitment scheme for biometric template security. In *2019 7th International Workshop on Biometrics and Forensics (IWBF)*. IEEE, USA, 1–6.

[37] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, UK.

[38] 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. ACM, USA, 307–328.

[39] Alberto Ibarrondo, Hervé Chabanne, and Melek Önen. 2021. Practical Privacy-Preserving Face Identification based on Function-Hiding Functional Encryption. In *International Conference on Cryptology and Network Security*. Springer, Austria, 63–71.

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

[41] Seong-Yun Jeon and Mun-Kyu Lee. 2021. Acceleration of Inner-Pairing Product Operation for Secure Biometric Verification. *Sensors* 21, 8 (2021), 2859.

[42] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J Wu. 2018. Function-hiding inner product encryption is practical. In *International Conference on Security and Cryptography for Networks*. Springer, Italy, 544–562.

[43] 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, USA, 336–353.

[44] Joohee Lee, Dongwoo Kim, Duhyeong Kim, Yongsoo Song, Junbum Shin, and Jung Hee Cheon. 2018. Instant privacy-preserving biometric authentication for hamming distance. *Cryptology ePrint Archive* 2018, 1214 (2018), 28.

[45] Ying Luo, S Cheung Sen-ching, and Shuiming Ye. 2009. Anonymous biometric access control based on homomorphic encryption. In *2009 IEEE International Conference on Multimedia and Expo*. IEEE, USA, 1046–1049.

[46] T Soni Madhulatha. 2012. An overview on clustering methods. *arXiv preprint* 2012, 1205.1117 (2012), 7.

[47] David Marr and Ellen Hildreth. 1980. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences* 207, 1167 (1980), 187–217.

[48] 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*. ACM, Canada, 35–52.

[49] 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)*. USENIX Association, USA, 2165–2182.

[50] Michael O Rabin. 2005. How to exchange secrets with oblivious transfer. *Cryptology ePrint Archive* 2005, 187 (2005), 27.

[51] 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*. ACM, USA, 325–342.

[52] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, Korea, 707–721.

[53] Zhang Rui and Zheng Yan. 2018. A survey on biometric authentication: Toward secure and privacy-preserving identification. *IEEE access* 7 (2018), 5994–6009.

[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* 1 (2022), 291–316.

[55] Amit Sahai and Brent Waters. 2005. Fuzzy identity-based encryption. In *EUROCRYPT*. Springer, Germany, 457–473.

[56] Adi Shamir. 1979. How to share a secret. *Comm. of the ACM* 22, 11 (1979), 612–613.

[57] Junichi Tomida, Masayuki Abe, and Tatsuaki Okamoto. 2016. Efficient functional encryption for inner-product values with full-hiding security. In *International Conference on Information Security*. Springer, USA, 408–425.

[58] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. USENIX Association, Canada, 601–618.

[59] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2020. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *arXiv preprint arXiv:2004.02229* 2020, 2004.02229 (2020), 1–21.

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

[61] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. 2013. Packed homomorphic encryption based on ideal lattices and its application to biometrics. In *International Conference on Availability, Reliability, and Security*. Springer, Germany, 55–74.

[62] Kai Zhou and Jian Ren. 2018. PassBio: Privacy-preserving user-centric biometric authentication. *IEEE Transactions on Info. Forensics and Security* 13, 12 (2018), 3050–3063.

[63] Youwen Zhu and Tsuyoshi Takagi. 2015. Efficient scalar product protocol and its privacy–preserving application. *International Journal of Electronic Security and Digital Forensics* 7, 1 (2015), 1–19.

A DIAGRAM OF FUNSHADE PRIMITIVES VS ARIANN PRIMITIVES

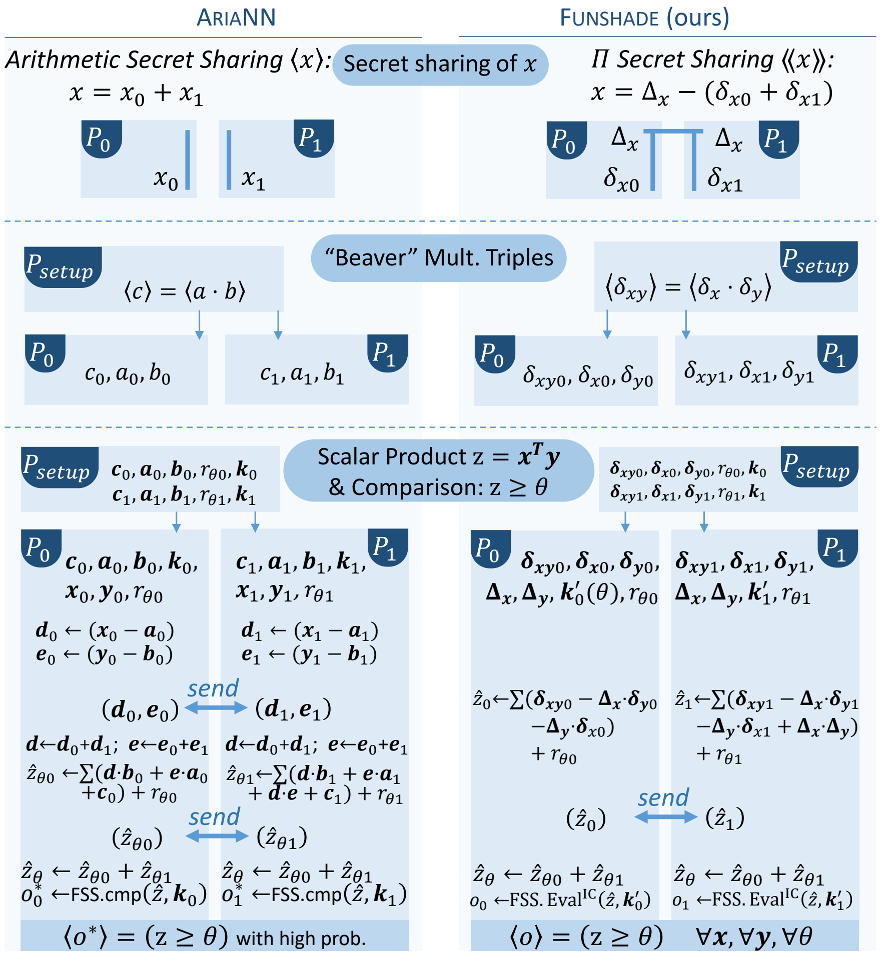

Figure 3: Side-by-side comparison between AriaNN and Funshade (ours)