White Paper: AON-PRISMA
All-or-Nothing Private Similarity Matching

Florian Kerschbaum, Hongyang Zhang, John Premkumar,
Xinda Li, Faezeh Ebrahimianghazani, Lucas Gamez
University of Waterloo

Koray Karabina, Prini Kotian
NRC

Version 1.0

1 Introduction

The data for socially or economically relevant analyses is often distributed across several entities. For example, health care data and socioeconomic data are often held by different entities, such as hospitals, insurance agencies and tax services. Before it can be used to run a joint analysis, it must be linked across those entities. Even within one entity, there may exist sub-entities, e.g., different customers, that require a separation of data. Personal data is protected by privacy legislation, such as PIPEDA, HIPAA, CCPA, or GDPR. It is therefore necessary to deploy state-of-the-art data protection technologies. Furthermore, it is important to balance the data subject's perception of privacy and the quality of services offered.

Record linkage is offered by many trusted third parties, e.g. Statistics Canada[1], Ontario Health Study[2], or Population Data BC[3]. However, these entities carry a substantial risk handling this data and are required to deploy strong state-of-the-art data protection technologies.

The ideal protection would be to encrypt the data before sending it to the matching entity. Yet, that's non-trivial, since standard encryption destroys any similarity between data. Data entries often subtly differ due to data entry errors or missing data fields. Hence, it is important to be able to match not only identical but also similar data entries. Functional encryption allows a similarity function to be evaluated over encrypted data but is too slow for practical use. Partial encryption of data elements leaks partial matching results that have regularly been shown to break the encryption. In this white paper we describe our open-source software AON-PRISMA based on a novel cryptographic algorithm for similarity matching that is both secure and efficient. AON-PRISMA has three key properties:

- *Secure*: AON-PRISMA supports all-or-nothing disclosure. This means if two data entries differ more than a tunable similarity threshold, no information is revealed about them, including their distance. This also means that it is not possible to construct a new encrypted data entry from two or more encrypted data entries that can be used to match against other records.
- *Efficient*: AON-PRISMA only uses secret-sharing and symmetric encryption and no public-key encryption. It is hence much faster than functional or homomorphic encryption scheme. We also have optimized the field operations in our implementation.
- *Accurate*: AON-PRISMA uses preprocessing of data entries using machine learning. By fine-tuning existing models, e.g., large language models, we can leverage the power of representation learning. We use contrastive learning that keeps distance information to encode data entries before matching them encrypted.

[1] https://www150.statcan.gc.ca/n1/edu/power-pouvoir/ch3/5214780-eng.htm
[2] https://www.ontariohealthstudy.ca/for-researchers/data-linkage-opportunities/
[3] https://www.popdata.bc.ca/datalinkage

2 Challenges

2.1 Security: All-or-nothing disclosure

Standard encryption, such AES or RSA, can be ruled out for similarity matching since it destroys any similarity between data entries. This property is necessary since partial encryption leaks sensitive information. Consider the following trivial encryption scheme: For a bitstring, encrypt zeros as "A" and ones as "B". Two similar strings 010111 and 010110 would become ABABBB and ABABBA. This preserves similarity and one can perform similarity matching on the encrypted data, but it is easy to see that this encryption scheme offers little security.

For AON-PRISMA, we have developed a new encryption algorithm. This new encryption scheme allows comparing encrypted data entries, but also offers all-or-nothing disclosure. This means that if two data entries are not similar, no information about those data entries can be inferred. Encrypted records remain encrypted, even when matched.

Furthermore, AON-PRISMA's encrypts the entire data entry. Consider two bitstrings 000111 and 111000 matched against 000000. None of the two bitstrings by itself is similar to the matched string, but taking the first three bits of the first string and the last three bits of the second string produces a perfect match. Hence, a malicious entity with those two strings could produce a match despite not having a matching data entry. AON-PRISMA's encryption algorithm comes with a security proof that rules out these attacks.

2.2 Efficiency: Error-correcting codes and symmetric encryption

AON-PRISMA's encryption algorithm encodes each data entry using an error-correcting code, such that the combination of two encrypted data entries can be decoded if they are similar. The decoding uses efficient, polynomial-time algorithms, such as Berlekamp-Welch and list decoding. Symmetric encryption ensures that only similar records can be decoded.

Both – error-correcting codes and symmetric encryption – are fast operations, much faster than functional or homomorphic encryption. We compare the running time of AON-PRISMA's encryption algorithm to efficient, "practical" function-hiding inner-product encryption. AON-PRISMA is orders of magnitude faster.

We have optimized AON-PRISMA's encryption algorithm to operate over smaller field sizes that fit into CPU registers and take advantage of improved lookup tables, without sacrificing security. Such algorithmic optimizations provide unmatched speed up compared to naïve implementations. For comparing data sets, the time to match two records is critical, since the total running time scales linearly in the time for a single matching, i.e., matching two records 2 times faster reduces the total running time by a factor of 2. AON-PRISMA also supports private blocking, e.g., locality-sensitive hash functions, which reduces the number of necessary comparisons from all pairs to a much smaller subset.

2.3 Accuracy: Machine learning-based encoding

AON-PRISMA uses an extended version of the Hamming distance to match records. Each data entry encoding is not composed of bits, but integers which are then compared element-wise and their number of matches comprises the similarity. For similarity matching the threshold to determine a match can be set at encryption time and is tunable for different data domains.

A data entry input is encoded using a Transformer neural network into a fixed-length vector. The goal is to map the input to a feature space, such that the mapping preserve similarity between data entries. AON-PRISMA uses contrastive learning to achieve this encoding. This type of training makes AON-PRISMA very flexible. Depending on the training data set, AON-PRISMA's encoding can capture syntactic, e.g., spelling errors, missing field, etc., and semantic, e.g., different vocabulary, domains, etc., differences. AON-PRISMA can also leverage existing model snapshots using fine-tuning. This allows to leverage large language models for natural language matching using state-of-the-art performance.

3 Encryption Algorithm

In this section we describe our encryption algorithm. Next to the use of efficient primitives, such as symmetric encryption and error-correcting codes, we optimized our implementation to operate over small field sizes to significantly reduce the constants factors in running time. Before we describe our algorithms, we present a formal definition of our algorithms and their security and correctness properties.

3.1 Definitions

Let λ be the security parameter that bounds a computational adversary. A secure approximate equality operator consists of three, possibly probabilistic polynomial-time algorithms.

1. $K \leftarrow \mathsf{KeyGen}(1^\lambda)$: generates a (symmetric) key K using the security parameter λ.
2. $\vec{c} \leftarrow \mathsf{Encode}(K, \vec{x}, t)$: generates a transformed ciphertext c for vector $\vec{x}$ and threshold t.
3. $\top/\bot \leftarrow \mathsf{Compare}(\vec{c}_1, \vec{c}_2)$: outputs "equal" ($\top$) or "not equal" ($\bot$) given two transformed vectors $\vec{c}_1$ and $\vec{c}_2$.

We say a secure approximate equality operator is correct, if

$$\begin{aligned}
&\forall \lambda, \vec{x}, \vec{y}, t, \\
&K \leftarrow \mathsf{KeyGen}(1^\lambda), \\
&\vec{c}_1 \leftarrow \mathsf{Encode}(K, \vec{x}, t), \\
&\vec{c}_2 \leftarrow \mathsf{Encode}(K, \vec{y}, t), \\
&d(\vec{x}, \vec{y}) \geq t \Longrightarrow \mathsf{Compare}(\vec{c}_1, \vec{c}_2) = \top, \\
&Pr[d(\vec{x}, \vec{y}) < t \wedge \mathsf{Compare}(\vec{c}_1, \vec{c}_2) = \top] = \mathsf{negl}(\lambda).
\end{aligned}$$

Let $\mathcal{L}(\vec{x}, \vec{y})$ be the information about $\vec{x}$ and $\vec{y}$ leaked by executing a secure approximate equality operator. We denote computational indistinguishability of two ensembles E_1 and E_2 as $E_1 \stackrel{c}{\equiv} E_2$. We say an approximate equality operator is $\mathcal{L}$-secure, if there exists a simulator $\mathsf{Sim}(\mathcal{L}(\vec{x}, \vec{y}))$ such that

$$\begin{aligned}
&\forall \vec{x}, \vec{y}, t, \\
&K \leftarrow \mathsf{KeyGen}(1^\lambda), \\
&\vec{c}_1 \leftarrow \mathsf{Encode}(K, \vec{x}, t), \\
&\vec{c}_2 \leftarrow \mathsf{Encode}(K, \vec{y}, t), \\
&\mathsf{Sim}(\mathcal{L}(\vec{x}, \vec{y})) \stackrel{c}{\equiv} \mathsf{Compare}(\vec{c}_1, \vec{c}_2).
\end{aligned}$$

Let x_i be the i-th entry of vector $\vec{x}$. Let $H_{K_i}(\cdot)$ be a keyed, one-way transformation function, e.g., a message authentication code. Let n be the length of vectors $\vec{x}$ and $\vec{y}$. We say an approximate equality operator is *relaxed* secure, if it is $\mathcal{L}^\star$-secure for

$$\mathcal{L}^\star(\vec{x}, \vec{y}) = \begin{cases} \top,\ H_{K_i}(x_i) - H_{K_i}(y_i),\ \forall i | x_i \neq y_i, & \text{if } d(\vec{x}, \vec{y}) > t; \\ \bot, & \text{otherwise.} \end{cases}$$

3.2 Base Algorithm (Over Large Fields)

Let $\mathbb{I}(x, y)$ be the equality function, i.e., $\mathbb{I}(x, y) = 1$ if $x = y$ and 0 otherwise. Our secure approximate equality operator implements the following similarity function d.

$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{n} \mathbb{I}(x_i, y_i).$$

We implement our secure approximate equality operator as follows

- $\mathsf{KeyGen}(1^\lambda)$: Execute $K \leftarrow \mathsf{Keygen}_{MAC}(1^\lambda)$ and output K.

- $\mathsf{Encode}(K, \vec{x}, t)$: Let m and m' be messages. Create n codewords from a linear code, e.g., Shamir's secret shares for Reed-Solomon codes, $\sigma_i = SS_{2t-n,m}(i)$ and $\sigma'_i = SS_{2t-n,m'}(i)$ for $1 \leq i \leq n$. Output the vectors $\sigma_{x,i} = \sigma_i + MAC_{MAC_K(i)}(x_i)$ and $\sigma'_{x,i} = \sigma'_i - MAC_{MAC_K(i)}(x_i)$.
- $\mathsf{Compare}(\vec{\sigma}_{\vec{x}}, \vec{\sigma}'_{\vec{y}})$: Compute $\rho_i = \sigma_{x,i} + \sigma'_{y,i}$. Reconstruct $m + m'$ from ρ_i. If the reconstruction is successful, output $\top$ and optionally $m + m'$, else $\perp$.

3.3 Small Field Implementation

In small fields we can no longer rely on adversary's inability to guess the MAC of an input element x_i. We therefore need to introduce some redundancy and confusion into the algorithm. Let α, β be (repetition) parameters. Let π_K be a random permutation of $[1, \beta n]$ uniformly chosen based on K. We can implement our secure approximate equality operator over small fields as follows

- $\mathsf{KeyGen}(1^\lambda)$: Execute $K \leftarrow \mathsf{Keygen}_{MAC}(1^\lambda)$ and output K.
- $\mathsf{Encode}(K, \vec{x}, t)$: We repeat the following procedure for $1 \leq j \leq \alpha$. Let m_j and m'_j be messages. Create βn codewords from a linear code, e.g., Shamir's secret shares for Reed-Solomon codes, $\sigma_{j,i} = SS_{2\beta t-\beta n,m_j}(i)$ and $\sigma'_{j,i} = SS_{2\beta t-\beta n,m'_j}(i)$ for $1 \leq i \leq \beta n$. Output vector $\sigma_{\vec{x},j,i} = \sigma_{j,i} + MAC_{MAC_K(j,i)}(x_{\lceil \pi_K(i)/\beta \rceil})$ and $\sigma'_{\vec{x},j,i} = \sigma'_i - MAC_{MAC_K(j,i)}(x_{\lceil \pi_K(i)/\beta \rceil})$.
- $\mathsf{Compare}(\vec{\sigma}_{\vec{x},1}, \vec{\sigma'}_{\vec{y},1}, \ldots, \vec{\sigma}_{\vec{x},\alpha}, \vec{\sigma'}_{\vec{y},\alpha})$: For $1 \leq j \leq \alpha$,
  - Compute $\rho_{j,i} = \sigma_{\vec{x},j,i} + \sigma'_{\vec{y},j,i}$. Reconstruct $m_j + m'_j$ from $\{\rho_{j,i}\}_{i=1}^{\beta n}$. If the reconstruction is not successful, output $\perp$ and halt.
- Output $\top$ and optionally $m_1 + m'_1, \ldots, m_\alpha + m'_\alpha$.

3.4 Worst Case Guessing Attack

In small fields we can no longer rely on adversary's inability to guess the MAC of an input element. We therefore need to consider the adversary's capability to guess such a MAC in the worst case.

Let p be the probability of the event E that $\vec{x}$ and $\vec{y}$ differ in one element above the interpolation threshold. We start by assuming $\beta = 1$ and the π is the identity function. Then, if E and the attacker knows m and m', the attacker can try all N values of all $\sigma_{\vec{x},1,i}$ and $\sigma_{\vec{y},1,i}$. The attacker can target a specific index i or try all possible i. If x_i and y_i are the same (but the vector differs at some other index), the attacker obtains a freshly drawn random number. If x_i and y_i differ, the attacker obtains $MAC_{MAC_K(1,i)}(x_i) - MAC_{MAC_K(1,i)}(y_i)$ which is statistically different from a freshly drawn random number, since it is the same for each event E. Given sufficiently many attempts, the attacker can recover a distinctive histogram over multiple events E.

If $\beta > 1$ and π is a random permutation, then the attacker needs to guess β indices which range over $\binom{\beta n}{\beta}$ value tuples. He can iterate over $\beta - 1$ indices fixing them to every possible value tuple, since there is always an β-th value that will recover $m_j + m'_j$, and compute the value for the β-th index using Lagrange interpolation.

3.5 Computationally Efficient Guessing Attack Using Berlekamp-Welch

The worst-case guessing attacks works, even if the number of matching values in the vectors is below the secret sharing threshold. However, the attack is more efficient if there are more matching values and one can use efficient reconstruction attacks to speed up the attack.

Let p be the probability of the event E that $\vec{x}$ and $\vec{y}$ differ in one element above the correction threshold of the Berlekamp-Welch algorithm (considering that elements may be repeated β times). If E, the attacker can try all N^β values of β chosen elements $\sigma_{\vec{x},i}$ and $\sigma_{\vec{y},i}$, and run the Berlekamp-Welch algorithm to see if it reconstructs. If the chosen index i is indeed an index where $\vec{x}$ and $\vec{y}$ differ, the attacker obtains $MAC_{MAC_K(j,i)}(x_{\lceil \pi_K(i)/\beta \rceil}) - MAC_{MAC_K(j,i)}(y_{\lceil \pi_K(i)/\beta \rceil})$.

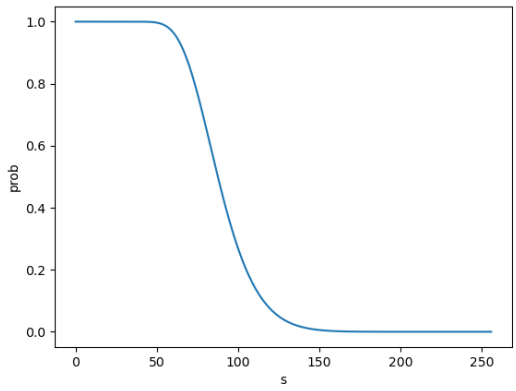

Figure 1: $p(E_1)$ v.s. s curve.

4 Locality Sensitive Hashing

We can implement a locality-sensitive hashing (LSH) algorithm to reduce the number of equality operations computed by the server. Let r be the number of indices selected and k be the repetition parameter of the LSH algorithm. Let b be the number of bins. Each client performs the following:

- Uniformly samples r values from the encoding vector x and get $(x_1, ..., x_r)$.
- Compute $h_i = H_K(x_1||...||x_r) \bmod b$.
- Repeat the above steps for k times and get a vector of hashes $H_x = [h_1, ..., h_k]$. The client sends the vector to the server along with the record.

On the server side, for two records x and y that require comparison, if $\exists i, H_x[i] = H_y[i]$, the server returns the output of our secure approximate matching algorithm. Otherwise, it returns false.

4.1 Analysis

Let t be the decision threshold and s be the number of non-matching indices (extended hamming distance) of two vectors x and y.

Define E_0 be the event that the selected index subset contains at least one of the non-matching indices. Then $p(E_0) = 1 - \frac{\binom{n-s}{r}}{\binom{n}{r}}$. We assume if E_0, then $h_{ix} \neq h_{iy}$ (may need to consider hash collision later). Define E_1 be the event that at least one of k hashes of x and y are the same at the same index. That is, $\exists i, H_x[i] = H_y[i]$. Then, $p(E_1) = 1 - p(E_0)^k$.

In our algorithm, the server performs matching if E_1 happens. Thus, our goal is to achieve the following by tuning r and k: when $s \leq t$, $p(E_1)$ is large (minimize false negatives), and when $s > t$, $p(E_1)$ is small (minimize the wasting of computation time). This can be done by defining a score function and using a grid search. A possible criterion could be the area under curve (AUC) when $s \leq t$ and $s > t$.

Figure 1 is a sample plot of the $p(E_1)$ v.s. s curve. Here, we take $n = 256, t = 92, k = 50, r = 10$.

4.2 Security

We implement differentially private padding to prevent leaking whether a specific element was part of a party's input set. The privacy parameter ϵ is tunable and the number of padding elements per bin is expected to be linear in ϵ. Loosely speaking, the server will not be able to determine whether an element is part of any party's input set except with probability bounded by ϵ. We use the Laplace mechanism to satisfy differentially private padding. The Laplace Protocol works by inserting (only inserting and not removing) a carefully chosen number of dummy records into each bin of the blocking strategy such that the bin sizes are differentially private. While candidate matches may contain dummy

records, they do not contribute to the output set of matches, because the dummy records do not match any record. These candidate matches are then securely matched using the rest of our algorithm.

5 Machine Learning-based Encoding

AON-PRISMA uses an extended version of the Hamming distance to match records in the feature space. To map the input space to the feature space such that similar data (under some distance metrics) in the input space are close in terms of extended Hamming distance in the feature space, one needs to learn an encoder. In this section, we introduce a machine learning technique, contrastive learning, to achieve this goal.

To fine-tune the model on the public dataset, we use the loss that encourages the distance of matched data pair to be small while enlarging the distance of unmatched data pair in the encoding space. This is achieved by minimizing the contrastive loss w.r.t. θ for each batch:

$$L_{\text{contrastive}}(s_i, s_j, \theta) = \mathbf{1}[y_i = y_j] \cdot \|\theta(s_i) - \theta(s_j)\|_2^2 + \mathbf{1}[y_i \neq y_j] \cdot [\max(0, \epsilon - \|\theta(s_i) - \theta(s_j)\|_2)]^2,$$

where θ is the encoder to be fine-tuned, $((s_i, y_i), (s_j, y_j))$ is the data pair, ϵ is the margin hyper-parameter, and $\mathbf{1}[\cdot]$ is the indicator function which is equal to 1 if the condition in the bracket holds, and is equal to 0 otherwise. After fune-tuning, we apply AON-PRISMA to the output vectors of the encoder. The encoder we use depends on specific applications. For text data where semantic similarity should be captured, we use pre-trained RoBERTa as our backbone model.

6 Applications

6.1 Record Linkage

In this section, we consider the record linkage application where the goal is to identify which records belong to the same identity. We tested AON-PRISMA on three record linkage datasets. Each dataset contains two sub-tables and a ground-truth table for true links. For each dataset, we split both sub-tables by 60%, 20%, and 20% for training, validation, and testing purposes. To measure the imbalance level of a dataset, we define *ratio* r to be # of non-match pairs / # of match pairs in the dataset.

- Amazon-google dataset: The Amazon-google dataset has 1,363 and 3,226 records each. The total number of data pairs in the validation dataset is $272 \times 645 = 175,440$. The ratio of the validation dataset is roughly 3,579.
- Febrl4: The Freely Extensible Biomedical Record Linkage dataset is generated with 5,000 original records and 5,000 duplicates, with one duplicate per original. The total number of data pairs in the validation dataset is 1,000k. The ratio of the validation dataset is roughly 5,000.
- Abt-buy: The Abt-buy dataset contains 1,081 entities from abt.com and 1,092 entities from buy.com. The total number of data pairs in the validation dataset is around 47k. The ratio of the validation dataset is roughly 960.

6.1.1 Accuracy

For the Amazon-google dataset and $n = 256$, the distribution of the extended hamming distance of each data pair (after encoding) is shown in Figure 2:

The results of the experiment for all datasets are shown below in a table ($n = 256$).

Dataset	Model	r (model training)	Th value	Test acc	Test f-score
Amazon-google	Roberta	1,000	118	99.98	73.85
Febrl4	Roberta	1,000	122	99.99	97.16
Abt-buy	Roberta	1,000	106	99.98	88.46

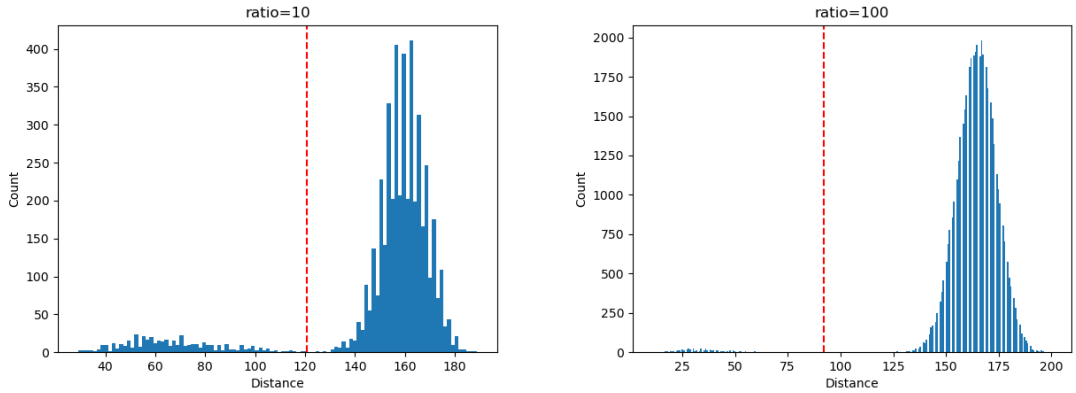

Figure 2: The distribution of the extended hamming distance of each data pair (after encoding) for Amazon-google dataset.

6.1.2 Running Time

The computational time and overall utility of end-to-end evaluation are shown below in a table. Current runs are performed on the Intel Xeon Platinum 8368 cpu (152 cores). The LSH column shows the parameters for the locality-sensitive hashing if used.

Dataset	n	N	ℓ	Poly degree	LSH (k, r)	F-score	Total time	# of matchings
Amazon-google	256	2^{16}	2	46	(10, 3)	70.37	29m9s	110k
	256	2^{16}	2	46	-	73.85	39m46s	175k
Febrl4	256	2^{16}	2	20	(10, 4)	96.25	2h28m	529k
	256	2^{16}	2	20	-	97.16	3h58m	1,000k
Abt-buy	256	2^{16}	2	24	(10, 2)	90.19	9m46s	30k
	256	2^{16}	2	24	-	90.19	12m45s	47k
Amazon-google	256	2^{32}	1	-	-	-	24m39s	-
	256	2^{32}	1	-	-	-	32m57s	-
Febrl4	256	2^{32}	1	-	-	-	1h46m	-
	256	2^{32}	1	-	-		2h39m	-
Abt-buy	256	2^{32}	1	-	-	-	7m51s	-
	256	2^{32}	1	-	-	-	9m19s	-
Amazon-google	256	plaintext	1	-	-	-	2m42s	-
Febrl4	256	plaintext	1	-	-	-	18m29s	-
Abt-buy	256	plaintext	1	-	-	-	1m25s	-

6.2 Biometric Template Protection (Keystroke dynamics data)

In this section, we present how we apply AON-PRISMA to match a type of biometric data, keystroke dynamics data, accurately and privately. Keystroke dynamics is the analysis of typing rhythms to discriminate among users. It can be used for biometric verification and authentication purposes. In the real world, password-based authentication is the most common and effective method to prevent unauthorized access. However, user-created passwords may be easily compromised by adversaries if they do not follow certain password-creating rules. Moreover, adversaries may exploit compromised passwords to break into accounts that have similar passwords. Keystroke dynamics was proposed as a biometric identifier to differentiate users in order to mitigate these security threats. With keystroke dynamics, imposter attempts to authenticate using the true password could be detected and rejected if the typing behaviour deviates significantly from that of the genuine user. Since the user's keystroke dynamics data is private and sensitive, we apply AON-PRISMA to ensure the matching is performed on ciphertexts, and no private information is revealed.

In a typical authentication round, the server decides whether to accept the login attempt of a user based on the submitted typing behaviour data and the user-specific template data stored on the server. For each user's data, a similarity score is computed on the server by comparing the data with the corresponding user's template. Then the server determines whether the user is genuine based on

a decision threshold.

6.3 Server methods

We assume the server stores a few template records which could be used to authenticate a given record for each client. The following proposed methods are based on the Extended Hamming distance, so they could be applied to compute the score on the server for a given record.

- Mean: The user template is stored as the mean of a set of genuine user data. In the test phase, the score is calculated as the extended hamming distance between the test vector and the mean vector.
- Filtered mean: In the training phase, the mean and std of the training vector are calculated. Any training vector element that is not within mean $\pm$ 3 $\times$ std is removed, and a more robust mean vector is computed without these outliers. The testing phase is the same as Mean.
- Median: Similar to Mean, but the element-wise median is calculated.
- Filtered median: Similar to Filtered mean, but the element-wise median is calculated.
- Nearest-neighbour: In the training phase, the server saves the list of training vectors. In the testing phase, the score is calculated as the extended hamming distance from the test vector to the nearest training vector.

6.4 Client encoders

We could either use the identity encoder or train a custom encoder for each client. For the identity encoder, the client does not encode the record at all, but quantization and encryption still apply. For the custom encoder, we train a machine learning-based encoder that encodes clients' records to fixed-length encodings. We use contrastive learning to train an encoder for each user. We construct positive pairs by taking two genuine records from the user and negative pairs by taking a genuine user's record and an imposter's record.

6.5 Results

We perform experiments on the CMU Keystroke Dynamics Benchmark Dataset. We split each user's keystroke data into non-overlapping training, validation and testing sets. The validation and testing dataset each contains 100 genuine and 500 impostor data records.

The experiment results are displayed in Table 1. Table 2 contains the baseline results. To produce baseline results for comparison, we do not quantize the record vector and use l_1 distance to measure distances. The identity encoder is used in this case. The F-score is calculated as the averaged F-score across every user.

Table 1: Results for keystroke data

Encoder type/Server method	Mean	Filtered Mean	Median	Filtered Median	Nearest-neighbour
Identity encoder	61.53	66.37	69.72	70.15	80.19
Custom encoder	79.79	79.67	79.97	80.33	82.75

Table 2: Baseline results for keystroke data

Approach	F-score
Mean	68.74
Mean + filtered	73.47
Median	75.60
Median + filtered	75.79
Nearest-neighbour	78.72