

Feasibility of Privacy Preserving Minutiae-Based Fingerprint Matching

Julia Mader and Thomas Lorünser^a

AIT Austrian Institute of Technology, Vienna, Austria
{firstname.lastname}@ait.ac.at

Keywords: Multiparty Computation (MPC), Minutiae-Based Fingerprint Matching.

Abstract: While biometric data, such as fingerprints, are increasingly used for identification and authentication, their inability to be revoked once compromised raises privacy concerns. To mitigate these concerns, in this ongoing research, we explore the use of Multiparty Computation (MPC), which allows secure computations on encrypted data, as an option for privacy-preserving fingerprint matching. Despite MPC's known drawback of slowing down computation, recent advancements make it a viable option for real-world applications. Our research focuses on implementing and optimizing a minutiae-based fingerprint matching algorithm with MPC, addressing the challenge of maintaining privacy while ensuring reasonable computation times. We present our implementation using SourceAFIS optimized for MPC and evaluate its performance to assess if current protocols are ready for deployment in time critical scenarios. Preliminary results show promise, emphasizing our ongoing research to achieve a fully-fledged MPC implementation with high accuracy.

1 INTRODUCTION

Biometric data such as fingerprints, retina or facial characteristics are increasingly used as a promising replacement for conventional identification and authentication such as passwords or other identification tokens. However, while they are unique, permanent and difficult to forge, they cannot be revoked once leaked. Where passwords can be changed if compromised, biometric data cannot be altered. Furthermore, there are privacy concerns about the inevitable data collection, storage and potential misuse of personal and unique physical characteristics (Simoes et al., 2009). These disadvantages clearly demonstrate the importance of including protections against deliberate and accidental disclosure or misuse of biometric data. For this very reason, fingerprints are typically only scanned, stored and processed locally and on dedicated hardware, i.e., secure elements. However, this approach also limits application to local authentication and prevents from more advanced use cases.


An alternative approach to special hardware for privacy-preserving fingerprint matching is using multiparty computation (MPC), which allows for secure computations to be performed on encrypted data without the need for decryption, thus mitigating the security risks associated with exposing sensitive information during processing. To do so, MPC ensures in-

put privacy by allowing a set of parties to collectively compute a function over their private inputs without disclosing those inputs to any other party involved in the computation.

However, it should be noted that MPC comes with the drawback of significantly slowing down computation and therefore the matching process. While this is not a problem for matchers based on feature vector comparisons, as the matching process of computing the Euclidean distance can be also performed quickly in the encrypted domain, it poses a significant challenge for the more complex minutiae-based matchers. However, in recent years, there have been significant advancements in MPC protocol design and implementation, making it ready for real-world application. A different application was presented in (Lorünser et al., 2022). In particular, in this work we show, that the realization of a fully fledged privacy preserving minutiae-based matcher with practical efficiency is actually feasible.

1.1 Motivation and Use Cases

Our work is motivated by a novel use case identified together with the UN to improve border safety at airports which cannot rely on local matching but require remote matching capabilities (Strobl and Natali, 2022). For this solution we envision fingerprints being matched against No-Fly Lists at airport secu-

^a  <https://orcid.org/0000-0002-1829-4882>

rity checks. The idea is that in addition to verifying the traveler's passport data, their fingerprints are also compared to national and international No-Fly Lists. If a match is found, the traveler is not allowed to proceed their travels.

MPC is ideally suited for this scenarios, because it supports both objectives. Firstly, enables the confidential pooling of No-Fly Lists among these organizations that may not be willing to share them openly. Secondly, it ensures the privacy of regular travelers by keeping their biometrical data confidential. Only in the event of a match will the relevant data be disclosed to start further investigations, but never will fingerprints of regular travellers be disclosed.

More concretely, the data flow works as follows. First the individual No-Fly Lists of different organizations are secret shared (encrypted) and distributed among servers, e.g., three servers in this case. This corresponds to the pooling of input data in a way such that the individual servers do not learn the data they are holding, but only see encrypted data. Now, if a fingerprint is scanned, a so called template is generated and also encrypted (secret shared) and sent to the servers. Once the servers received the encrypted parts, they are able to obviously compute the result of the matching function in a joint MPC protocol and only reveal the result, a predicate in this case. If no match was found in the database, the individual at the security check preserves his privacy, because the server does not learn anything about his identity. In case of a match an alarm is triggered and further handling of the suspect can be triggered.

However, this is also a highly time-critical application and because MPC substantially slows down computation it is interesting to ask, if it is possible to obviously compute matching results in reasonable time. The waiting time for travelers should not be extended due to the additional check. The objective of our research is to implement and optimize a fully featured fingerprint matching algorithm with MPC to assess whether current protocols are ready for deployment in the given use case.

1.2 Research Goals and Methodology

Our research focuses on achieving an optimized implementation of multiparty computation of a previously existing minutiae-based fingerprint matching algorithm with state-of-the-art matching accuracy. Additionally, as a baseline for comparison we also implemented and evaluated a popular feature vector matcher (Jain et al., 1999) with MPC, as proposed in (Eerikson, 2020). However, it was already clear at the beginning, that the matching accuracy of the

FingerCode will not be good enough, which was confirmed in tests on plaintext data. However, in our general comparison of matchers we found that the open source minutiae-based matcher SourceAFIS (Važan, 2004) could compete with commercial matchers in terms of accuracy and showed outstanding performance, shown in Figure 2. Therefore, we selected SourceAFIS (Važan, 2004) for the implementation in MPC. The goal was to optimize it for the implementation with MPC and evaluate the results in terms of speed and accuracy.

The MPC implementation is developed using an open-source package called MPyC: Python Package for Secure Multiparty Computation (Schoenmakers, 2018). This package offers a range of cryptographic functionalities necessary for secure computation. Additionally, the work comprises an estimation of the execution time utilizing the MPC framework MP-SPDZ (Keller, 2020). Benchmarks of the frameworks can be found in (Lorünser and Wohner, 2020).

The algorithms are evaluated using fingerprint databases from the FVC2000: Fingerprint Verification Competition (Maio et al., 2002). Most notable characteristics used for the evaluation are the Fail Rate, False-Match-Rate (FMR), False-Non-Match-Rate (FNMR) and Equal-Error-rate (EER), where FMR and FNMR are the same. These rates provide valuable insights into the accuracy of the fingerprint matching algorithms.

For the purpose of this research, the implementation will be considered successful if it can complete the matching process within a time frame of less than 10 seconds.

1.3 Related Work

The best known performance of encrypted fingerprint matching was achieved by feature vector-based matchers, thanks to their straightforward structure and encryption friendly nature. In fact, in this approach it is enough to compute the Euclidean distance between two vectors of reasonable length. (Barni et al., 2010) as well as (Eerikson, 2020) have used this approach for their research. We actually based our FingerCode implementation on the latter utilizing their code for the feature extraction and only implementing their comparison from scratch. However, as already stated, matching accuracy of this techniques is very poor.

(Fälämaş et al., 2021) recently presented their research on privacy preserving password and iris authentication with mpc and secret sharing using feature vector comparison for the iris authentication. While this method of comparing feature vectors works well

for iris as well as facial recognition (Adnan et al., 2020), most fingerprint matchers are still minutiae-based matchers, because they perform better than feature-based fingerprint matchers.

There have also been some implementations of privacy-preserving minutiae based matchers. (Blanton and Gasti, 2011) encrypted FingerCode and an iris code as well as a minutiae-based fingerprint matcher using garbled circuits and homomorphic encryption. While they did not include a review of their matchers accuracy, we can draw certain conclusions from the structure of the matcher. Since the comparison of two minutiae is not rotation or translation invariant and calculated by computing their Euclidean distance, the accuracy of the matcher cannot surpass that of a feature vector-based one. (Shahandashti et al., 2012) also encrypt a minutiae-based fingerprint matcher using homomorphic encryption, once again using a very simplistic fingerprint matcher.

1.4 Contribution and Outline

To the best of our knowledge, we are the first to study the possibility for a fully fledged MPC implementation of a minutiae-based matcher with high accuracy comparable to commercial plaintext matchers. The effort is ongoing research and we present our approach as well as preliminary results. The remainder of the paper is organized as follows. In Section 2 we present the minutia matcher SourceAFIS and our ideas to optimize the algorithm for the implementation with MPC. In Section 3 we discuss preliminary evaluation of the privacy preserving implementation. In Section 4 we summarize our findings and present our future work.

2 PRIVACY PRESERVING MINUTIA MATCHING

2.1 Overview

The primary focus of this work is the implementation of a privacy preserving minutiae-based fingerprint matcher, specifically a MPC version of SourceAFIS (Važan, 2004) using the MPyC framework (Schoenmakers, 2018). The used algorithm was not implemented from scratch but partly taken from Robert Važan’s SourceAFIS implementation in Java. Especially the feature extraction could be used as is, since it is intended to run locally on the scanner side and not in the encrypted back-end. However, the core part for the matching functionality had to be completely

refactored and rewritten to achieve reasonable runtimes with MPC, albeit we did not deviate from the high level concept of the matcher regarding the computation of matching scores.

The goal of our research is to generate an understanding of the complications involved in implementing these complex algorithms using MPC and to provide optimizations as well as estimates for best achievable runtimes with current technology.

2.2 Feature Vector Matcher

Matchers based on feature vector comparisons work as follows: from the raw biometric information, e.g., a fingerprint scan, a feature vector is extracted which can then be compared to other feature vectors by calculating their standard deviation.

In that case, features represent key characteristics of biometric information and similarity is given by feature values being close to each other. A matching score is then simply given by the distance between two vectors, e.g., in the l_2 -norm. This makes these matchers very easy to implement and fast, even in the encrypted domain. These matchers also work exceptionally well on iris or face recognition, however, when it comes to fingerprint matching there accuracy is very low and far below any useful threshold as shown in our evaluation results in Figure 2. Compared to minutiae-based matchers they suffer from extremely high error rates in all possible working conditions and are not used in practice. Nevertheless, to have a baseline for benchmarking the minutiae based matcher, we also implemented and evaluated a feature vector matcher.

For this, we used FingerCode (Jain et al., 1999), a filterbank-based matcher, that uses a set of filters to analyse the texture information of the fingerprint image. In the paper Jain et al. summarized the three main steps as (i) determination of the reference frame of the fingerprint image, (ii) filtering the image in eight different directions using a bank of Gabor filters and (iii) computing the standard deviation of grey values around the reference point. The results are then stored in a feature vector, the so-called FingerCode, which can be easily compared to the FingerCode of other fingerprints by computing the Euclidean distance. This concludes a computationally cheap way to represent and match fingerprints.

2.3 Minutiae Matcher (SourceAFIS)

SourceAFIS is a commercially used and open-source minutiae-based fingerprint matcher by Robert Važan (Važan, 2004). The algorithm can be divided into two

parts: feature extraction and fingerprint matching.

During the feature extraction, SourceAFIS reads in an image of a scanned fingerprint and extracts their minutiae to store them in the so-called template. The minutiae are saved with their type, namely ending or bifurcation, their respective position as x and y coordinates and their direction as a 32-bit direction angle of the minutiae. As these features are not rotation and translation invariant, the template not only stores minutiae, but also computes and stores edges, which are connections between two minutiae. For each minutia the algorithm calculates a fixed number of edges, which are described by their length and two angles relative to the reference minutia and the neighboring minutia respectively. These parameters do not change when the edge is moved or rotated, making the algorithm rotation and translation invariant.

During the second part – the fingerprint matching – the algorithm evaluates whether two fingerprint templates, which are called probe and candidate, belong to the same finger. To do this, the algorithm first generates an special edge hash table for the probe in a pre-processing step to speed-up later search for similar candidate edges.

The candidate's part of the algorithm comprises three phases: (i) enumerate, (ii) crawl and (iii) score.

In detail, during (i) the algorithm's goal is to find root pairs available in both fingerprints. For this, it iterates through the minutiae of the candidate and calculates an edge to every other minutiae of the candidate and their respective value of the edge hash. This hashvalue is then compared to the probe's hash table. If there is an edge with the same hashvalue, the corresponding probe edge is further compared in detail to the calculated candidate edge. Are the edges similar enough a root minutia was found and a root pair consisting of the probe's edge and the candidate' edge is created.

Starting from the first root pair, in (ii) the algorithm then tries to build a spanning tree through all minutiae. For this the algorithm compares the surrounding edges of a root pair in probe and candidate. If one edge is similar enough, the ending minutiae gets added to the tree on both fingerprints and its surrounding edges will once again be compared to each other. This procedure is repeated until all minutiae are checked. An example how a spanning tree of matching fingerprints could look like is shown in Figure 1. In contrast to the previous step, where edges were compared to each other, this comparison takes the edge surroundings and placement into account, as edges can look very similar but be at different locations on the fingerprint.

After generating a spanning tree the algorithm

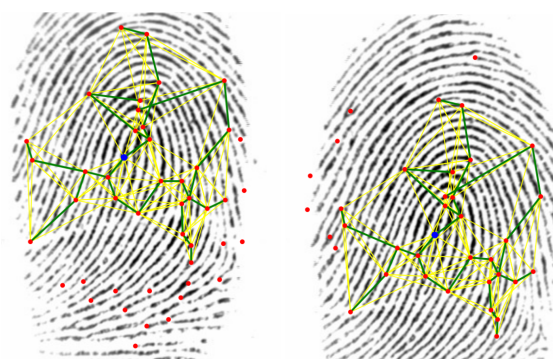


Figure 1: Spanning Tree of Matching Fingerprint Pair with minutiae marked in red, root pair blue, pairing tree green and graph of supporting edges yellow (Važan, 2004).

computes a match probability score in (iii). In order to achieve this, the algorithm counts matched features, scores how close they match, and sums the the scores up. The result is a representation of the matching probability, i.e., an indication if the resulting tree is a match or just a coincidence.

The generation of the tree and the scoring are repeated for every root pair to ensure the the best possible representation of the fingerprint. The best tree and probability score is returned as the similarity, which can then be compared to a threshold to declare the fingerprints matching or not.

2.4 Privacy Preserving SourceAFIS

As the feature extraction can be computed locally and without MPC, it can be incorporated unchanged from the original code to generate the fingerprint templates consisting of the fingerprint's minutiae and edges needed for the matching process. These templates can subsequently be fed into the MPC implementation for further fingerprint matching.

Regarding the encryption with MPC, the decision was made that all minutiae-describing details, including the coordinates, orientation, edge lengths and angles, have to be encrypted at all times. However, we choose that the minutiae and edge id can be handled unencrypted, because they do not contain sensitive information. By encrypting the minutiae information while handling minutiae and edge id unencrypted, a good balance between privacy and computational efficiency is achieved. This approach ensures that the most critical and private fingerprint data is protected while allowing some freedom for an efficient MPC implementation.

Moreover, it is stated in the SourceAFIS documentation (Važan, 2004) that the minutiae numbers are permuted pseudorandomly during the feature extraction. Although the ordering is consistent, which

ensures that running the algorithm with same image twice results in the same fingerprint template, thus guaranteeing no cryptographic security, the permutation adds an extra layer of protection, making it challenging for potential attackers to find sensitive data from the minutiae and edge numbering alone. Anyway, we also plan to add an additional random permutation step also for the candidate templates stored in the database, to prevent from any linkage of templates between different runs.

2.5 Optimizing SourceAFIS for MPC

A naive MPC implementation would just translate the existing algorithm into the secure domain, however, the existing implementation makes heavy use of MPC-unfriendly operators and data structures as well as programming patterns, which would render the approach infeasible. In particular, the algorithm extensively re-computes values, which are computationally expensive with MPyC per se, and applies the mentioned hash tables which would require oblivious indexing in Python dictionaries. In essence, we needed a different approach for the implementation with MPC and specific optimizations for the MPyC software framework and its protocols. In the following we describe our preliminary results on the presented main steps of SourceAFIS, which corresponding functions are called (i) enumerate, (ii) crawl and (iii) score and can be implemented with MPyC independently of each other.

(I) Enumeration Step

The first thing we changed from the original implementation is the redundancy at certain parts of the algorithm. Specifically, edges were computed during both feature extraction and enumeration, and again in the crawl phase, leading to unnecessary repetition. While acceptable in environments with negligible execution times, this became inefficient with MPC due to its substantial computational costs. Consequently, within our implementation, the strategy involves reusing results and leveraging pre-computed values.

The original implementation stores the hash as key-value pairs, where the key is the calculated hash value and the value includes the associated edge. Implementing this hash in MPyC poses challenges as the key has to remain secret while indexing the list. While the MPyC module `seclist` would allow secret indexing through encrypted scalar multiplication, the indexing cannot be performed since the keys do not exist seamlessly, making it impossible to incorporate this hash list with MPyC.

These considerations taken into account, we decided to utilize the edges generated in the feature extraction to create a look-up table that indicates which edges match. This look-up table can then be used for both the localization of the roots and the formation of the tree.

Specifically for the enumerate function, this meant that the re-computation of edges, the calculation of the hash value, the search of edges with the same hash value and the comparison of edges, when finding matching edges in the probe's hash, can be replaced by the 1:1 comparison of all edges, resulting in the output of a match array. To compare these edges the algorithm checks whether the difference between the lengths and angles of the probe and candidate edge fall within an acceptable error tolerance.

The match array can then be decrypted and is subsequently available in unencrypted form for further processing and as a lookup table for the fingerprint template's roots. This is permissible because the values in the array only represent the edge numbers, which are permuted, thus making the association of information challenging for a potential attacker.

Extensive testing revealed that while the implementation is correct, its independent use remains impractical due to substantial execution time challenges. For instance, comparing 164 edges takes 104s, averaging 640ms per edge. Extrapolating to an average 1 : 1 fingerprint comparison with around 16,000 edge comparisons, the execution time for the first part alone would be 3 hours. Consequently, the algorithm underwent further modifications to shorten the execution time.

Reflecting on the MPC implementation of the FingerCode algorithm, it can be observed that the most efficient time optimization lies in the combination of operations, thereby eliminating the need to re-share the values several times. Thus, instead of utilizing MPyC functions for performing operations in each step of the edge comparison, the compared lengths and angles are stored in lists, allowing for simultaneous operations over the whole list. At this point, AND and OR operations are replaced by element-wise vector multiplication and addition respectively.

These code modifications yield significant improvements, as the execution time of length comparison was reduced from 18s to 0.62s. This corresponds to a resulting time of 3.4% of the original time. The angle comparisons are implemented similarly, leading to the complete function needing 4.4s for 164 edge comparisons, which is an execution time of 26.7ms per edge comparison. This extrapolates to 7 minutes for 16000 edge comparisons, resulting in 4.2% of the original execution time.

(II) Crawl Step

Continuing the encryption approach of the enumerate function, the aim of the implementation is to utilize the resulting array of root pairs from the enumerate function, which represents matching edges, to construct the best possible spanning tree. Considering the previous decision to use edge and minutia numbers unencrypted, along with the utilization of the root array, this enables an unencrypted computation of the crawl function.

Since the function can be computed without encryption, the new implementation closely resembles the original implementation. The primary difference lies in the collection and comparison of the surrounding edges. Instead of re-matching the collected probe and candidate edges to each other, which would require encryption and increase the execution time, the new implementation utilizes the root array as a look up table to determine if the edges match and subsequently can be added to the spanning tree.

(III) Scoring Step

As previously mentioned, the scoring function computes the match probability score of the probe and candidate fingerprint image. Similar to the crawl function, the look-up table can be employed here as well. Furthermore, certain operations that do not involve sensitive data can be computed without encryption. Consequently, only a few comparison operations need to be encrypted. As this part of the implementation has not been completed yet, no performance results can be presented. For all additional evaluations, the unencrypted program was utilized.

3 PRELIMINARY EVALUATION

3.1 Setup

The software development and evaluation were conducted using rather standard hardware, namely a Dell Latitude 7490 notebook with an Intel i5-8350U CPU running on 1.70GHz with 4 cores and 16GB of RAM. The Python version 3.8.10 was used for the development.

The fingerprint databases used for the evaluation of the fingerprint matcher were created for the FVC2000: Fingerprint Verification Competition (Maio et al., 2002) and can be accessed online. There are four databases consisting of 80 fingerprint images, each database collected by different means. The databases 1 and 2 were collected by one optical and

one capacitive sensor; both were small-size and low-cost sensors, no attention was paid to correct finger centering and the sensor platens were not systematically cleaned. Database 3 was collected by a higher quality optical sensor with the sensor platens getting cleaned between images and database 4 consists of synthetically generated fingerprint images. The available databases include 80 images, consisting of 10 fingers with eight images each.

3.2 FingerCode Implementation

The algorithm for the feature vector extraction can be executed locally, thus encryption with MPC is only needed for the fingerprint matching. As for all feature vector based matchers the fingerprint matching consists of computing the euclidean distance between two feature vectors representing the biometrics. To compute the values in their encrypted form via Multiparty Computation, we were able to use various MPyC functions. The package provides heavily optimized operations for matrices and vectors, ensuring efficient execution of these operations within the MPyC framework. Our tests showed that by using the vector subtraction the average execution time for the subtraction can be lowered by 75%, while using the Schur product reduces the execution time for the multiplication to an eighth.

This results in an execution time for an 1 : 1 fingerprint comparison with 3 parties of 45ms. Compared to the time needed for the not encrypted FingerCode match of 1 μ s this amounts to a slowdown by 45000 times by using MPyC.

While the execution time of 45ms would work nicely for our envisioned scenario, the downside of feature vector based matchers is their accuracy. For the implemented matcher the feature extraction fails for 60-94% of the fingerprint images depending on the used fingerprint database. The main reasons are (i) that the core is too far on the edge of the image, such that the tessellated sectors around it are already outside of the picture, hence the image cannot be filtered and the FingerCode not created and (ii) that the quality of the fingerprint images is too bad and the core cannot be correctly identified.

This can be prevented by scanning fingerprints with greater precision and ensuring that the cores are centered in the scan. However, even when taking those fingerprint images not into account, the EER of the FingerCode matcher lies between 18% and 30%. This accuracy of the algorithm is extremely poor and does not fulfill any quality criteria in practical applications.

3.3 SourceAFIS Benchmarks

By combining the locally executed feature extraction implemented in Java with the three steps of fingerprint matching implemented and executed using MPyC, a complete fingerprint matching algorithm is obtained based on the open-source project SourceAFIS (Važan, 2004). Each database contains 80 fingerprint images that are matched against each other, resulting in a total of 6400 matching attempts per database.

With a FAIL rate of 2.5% and an EER of 7.9% on average, the matcher achieved very good results for an open-source project, especially compared to the results of the FingerCode matcher, as shown in Figure 2. The FNMR could be lowered by decreasing the threshold. However, the potential for improvement is already severely limited, as the EER is at a threshold of 2 or 4, depending on the database.

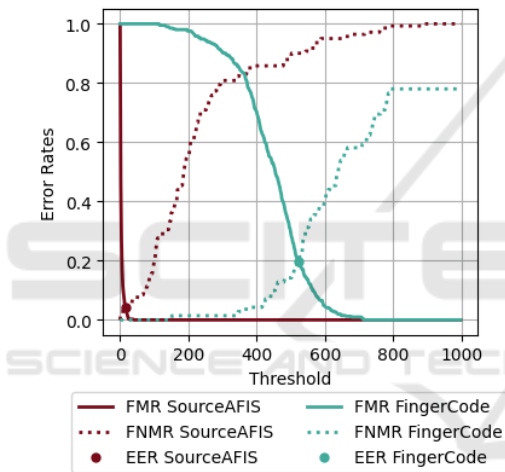


Figure 2: Comparison of Error Rates of minutiae-based matcher SourceAFIS and feature vector-based matcher FingerCode to measure accuracy.

While there are no apparent weaknesses in the functionality of the MPyC implementation, its primary disadvantage lies in the execution time. In the original implementation, the matching process, excluding the feature extraction, requires approximately 45ms to execute one 1 : 1 matching. However, in the case of the MPyC implementation, this process takes approximately 7 minutes, with crawl and scoring taking 0.2s and the edge comparison the rest. Despite achieving a significant improvement compared to the initial MPyC implementation with an execution time of around 3 hours, the resulting time still exceeds the requirements for the intended use-case. Thus, although the execution time has been reduced, further optimizations may be necessary to meet the desired performance targets.

3.4 Estimation for MP-SPDZ

Realizing that the current implementation of SourceAFIS with MPyC does not meet the requirements for the intended application, the author investigates alternative frameworks to potentially improve execution time. In this pursuit of execution time optimization for the fingerprint matcher, the MP-SPDZ framework is selected. MP-SPDZ (Keller, 2020) is a Multiparty Computation framework that may offer performance enhancements over the MPyC framework, which was utilized in the initial implementation.

To be able to provide a meaningful comparison of the two frameworks, a key component of the matcher is implemented in MP-SPDZ. This operation is representative of the computational complexity and performance characteristics of the full matcher.

Through the extensive evaluation of the fingerprint matcher it was determined that the most time-consuming part is the enumerate function trying to find all matching root pairs of two fingerprint images. For the MPC implementation the enumerate function was implemented as the comparison of all probe edges to all candidate edges, generating a root array that highlights the matching pairs. Generally, the full comparison of two edges consists of six separate comparisons of the lower and upper bounds of the length and the two angles, respectively. Considering that the fingerprint templates of the used databases consist of 10 to 100 minutiae depending on their quality, the algorithm comprises 50,000 to 5,000,000 comparisons.

The MP-SPDZ framework includes various protocols and settings. For comparison purposes, only different secret sharing schemes were chosen, all with a semi-honest setting. The protocol rep-field is on the basis of replicated secret sharing, while atlas and shamir are based on Shamir's secret sharing. The results can be found in Table 1.

Table 1: Execution time depending on the number of edge comparisons in MP-SPDZ.

comparisons	rep-field	atlas	shamir
10,000	0.06s	0.56s	0.31s
100,000	0.51s	5.21s	2.69s
1,000,000	6.5s	61.76s	38.65s

Comparing the tested protocols, the most efficient execution time is achieved utilizing replicated secret sharing. When factoring in the time required for the remainder of the matching algorithm, along with some computational overhead, the execution time for a 1:1 fingerprint comparison remains under 7 seconds. This represents a significant improvement, being ap-

proximately 1/60th of the time required for the MPyC implementation. Based on this estimation, it is recommended to utilize MP-SPDZ for future implementations and optimizations.

4 CONCLUSIONS

In our work we present the preliminary evaluation of two privacy preserving fingerprint matchers. The first is FingerCode, a matcher based on feature vector comparison. As expected the matcher was simple to implement with MPC and achieved a fast execution time of 45ms per 1:1 fingerprint comparison. However, the matcher showed poor accuracy results and is thus not applicable for real-world deployment.

While researchers are actively exploring machine learning-based FingerCode extraction and more accurate results can be expected in the future, the majority of matchers currently favor minutiae-based approaches due to their superior accuracy. This is why we focused mainly on the encryption of minutiae-based fingerprint matchers in our research.

The second matcher we implemented was SourceAFIS, a minutiae-based fingerprint matcher with very good matching accuracy. Our preliminary evaluation estimated that we are not only able to implement a working minutiae-based fingerprint matcher with MPC but also improve the execution time from 3 hours to 7 seconds.

In the future we plan optimizations regarding parallelization and exploitation of techniques to speedup the MPC implementation. We are also working on a full implementation in MP-SPDZ to fully leverage the potential of our optimizations. Finally, we are studying the leakage of our trade-off in detail as well as additional measures to further reduce it, e.g., by edge and vertex permutations.

ACKNOWLEDGEMENTS

The work was partially funded by the Austrian Security Research Programme KIRAS of the Federal Ministry of Finance via grant agreement no. 905287 ("PASSENGER") and from the European Union's Horizon Europe Programme via grant agreement no. 101073821 ("SUNRISE"). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

- Adnan, S., Ali, F., and Abdulmunem, A. A. (2020). Facial feature extraction for face recognition. In *Journal of Physics: Conference Series*, volume 1664, page 012050. IOP Publishing.
- Barni, M., Bianchi, T., Catalano, D., et al. (2010). Privacy-preserving fingercode authentication. In *Proceedings of the 12th ACM workshop on Multimedia and security*, pages 231–240.
- Blanton, M. and Gasti, P. (2011). Secure and efficient protocols for iris and fingerprint identification. In *Computer Security—ESORICS 2011: 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings 16*, pages 190–209. Springer.
- Eerikson, H. (2020). Privacy preserving fingerprint identification. Bachelor's Thesis.
- Fălămaș, D.-E., Marton, K., and Suciu, A. (2021). Assessment of two privacy preserving authentication methods using secure multiparty computation based on secret sharing. *Symmetry*, 13(5):894.
- Jain, A., Prabhakar, S., Hong, L., and Pankanti, S. (1999). Fingercode: a filterbank for fingerprint representation and matching. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 187–193.
- Keller, M. (2020). Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1575–1590.
- Lorünser, T. and Wohner, F. (2020). Performance comparison of two generic mpc-frameworks with symmetric ciphers. In *ICETE (2)*, pages 587–594.
- Lorünser, T., Wohner, F., and Krenn, S. (2022). A verifiable multiparty computation solver for the linear assignment problem: And applications to air traffic management. In *Proceedings of the 2022 on Cloud Computing Security Workshop, CCSW'22*, page 41–51, New York, NY, USA. Association for Computing Machinery.
- Maio, D., Maltoni, D., Cappelli, R., et al. (2002). Fvc2000: Fingerprint verification competition. *IEEE transactions on pattern analysis and machine intelligence*, 24(3):402–412.
- Schoenmakers, B. (2018). Mpyc secure multiparty computation in python. Last accessed 01-07-2023.
- Shahandashti, S. F., Safavi-Naini, R., and Ogunbona, P. (2012). Private fingerprint matching. In *Australasian Conference on Information Security and Privacy*, pages 426–433. Springer.
- Simoens, K., Tuyls, P., and Preneel, B. (2009). Privacy weaknesses in biometric sketches. In *2009 30th IEEE Symposium on Security and Privacy*, pages 188–203.
- Strobl, B. and Natali, M. (2022). Enhancing biometric data security by design. *ERCIM NEWS*, 131:25–26.
- Važan, R. (2004). Sourceafis fingerprint matcher. Last accessed 09-11-2023.