Information Security Research Institute

Biometrics in SplitKey using fuzzy extraction

Version 1.0

Johanna Maria Kirss

D-2-456 / 2022

Mailing address:
Cybernetica AS
Mäealuse 2/1
12618 Tallinn
Estonia

Biometrics in SplitKey using fuzzy extraction

Johanna Maria Kirss

Version 1.0

Abstract

The report details the progress made on a task in identity research in Cybernetica. The task was to evaluate the security levels of a biometrics-related cryptographic construction. In the report, some theoretical context is explained, as well as the research progress and possible open questions for improving the project.

Contents

1 Introduction

This document covers the research done by Johanna Maria Kirss and Peeter Laud into the feasibility of using biometrics, specifically face recognition, and fuzzy extraction in SplitKey. It is written as a way to get an introduction to the project. It is likely most useful to read it along with the original article and its code as well as the references in this document.

The article [ZCY21] describes a construction of fuzzy extractors for facial templates. We set out to implement it for ourselves in order to survey its possible security levels and determine appropriate parameters for us. The aim was to evaluate whether face recognition could be used in the place of PINs in SplitKey. However, in discussions with the identity research team, it was also suggested that biometrics could be used in key recovery, or that fuzzy extraction could be used for recognizing images other than faces, so the potential use was left somewhat open. Directly replacing PINs was questioned due to the irreplaceability of biometric data. While a single application was not determined, it still seemed fitting to all of the ways it could be used to find out more about the security of the scheme.

In section 2 of this report, we begin by explaining why fuzzy extraction is needed in the case of biometrics. We continue by giving the definitions of a fuzzy extractor and a secure sketch, introducing lattices and describing the lattice-based construction. Section 3 details the implementation process, explains the data we collected and gives some security results. We finish the report by listing some open questions and next goals for the project in section 4.

2 Theoretical background of lattice-based fuzzy extraction

2.1 Motivation: issues with storing biometric data

Let us first consider classical password storage. In a database meant for storing strings or numbers as passwords, the values are stored in a hash table. When saving a new password, the password is hashed with a random value called a salt, and the hash is stored, as well as the salt. Consequently, when a password is sent to the server, it hashes the password with the user's respective salt. If the hashes match, it accepts the password, but if not, it does not.

A person's biometric data is made up of their uniquely defining physical and behavioral characteristics – for example, their fingerprints, DNA, face or walking gait. Measuring such information can allow effective authentication and access control protocols, which might motivate its uses in Apple's FaceID, fingerprint readers on phones, and various biometric checks on border control points. However, biometric features are hard to change, making the information highly sensitive. Thus, securely storing biometric data is an important privacy question.

Images of people's faces and irises, scans of fingerprints and other immediate results of measuring someone's biometrics are both extremely sensitive and inconvenient to store and operate with. For that reason, biometric algorithms return a mathematical digest (a vector of numerical values) of someone's features instead. They do so in a way where digests from the same person are close to each other under some distance d; for fingerprint scans, that is often the Hamming distance, for face encodings, it is Euclidean distance. In short, the templates are likely from the same person if they are sufficiently close with regards to some distance.

Two templates of the same biometric data are rarely identical. We would fail if we tried to use the hash table method directly for storing biometric data because if two measurements of the same person's data are different, their hashes are as well, and so matching a new hash with a previous one will fail even if the querying person is the same. One solution could be to store the biometrics locally as Apple keeps its biometric templates in a Secure Enclave. If one wants to authenticate biometrically between a device and a server, though, it might need a different solution.

It might be useful to find a cryptographic mechanism to map measurements of the same biometric source to the same random value. The primitive should still ensure that measurements from different sources get mapped to different outputs. Then, such a random value can be sent to the server, and on the next query, the client measures their biometrics again, obtains the same random value, and sends it to the server, who accepts if the values are equal. This motivates the definition of a fuzzy extractor, detailed in the following section.

2.2 Definitions of fuzzy extractor and secure sketch

We give the definitions of a fuzzy extractor and a secure sketch as given in [DORS06], but they can also be found in [ZCY21]. For a more complete treatment of the topic, refer to the rest of the first paper.

Note that both definitions use the notation $d(w, w')$. In order to define a fuzzy extractor and a related primitive, secure sketch, it is necessary that the templates (collections of encoded biometric data from a measurement) have some sort of a **distance** defined between them. That is, it is necessary that for any two templates w and w', we can find some non-negative real number $d(w, w')$ that signifies how similar the templates are. Precisely speaking, we look at the inputs to the fuzzy extractor and secure sketch as coming from some metric space $\mathcal{M}$, where the choice of the metric also influences how we can construct the fuzzy extractor later.

We will also need the concept of **min-entropy**. Given a random variable X with a distribution P, the min-entropy of X is the negative logarithm of the most likely outcome, and we denote it as $\mathbf{H}_\infty(X)$. In mathematical notation,

$$\mathbf{H}_\infty(X) = -\log \max_{x \in X} p(x). \tag{1}$$

Since the min-entropy of X is only dependent on its distribution P, one might also call (1) the min-entropy of a distribution P and denote it as $\mathbf{H}_\infty(P)$.

We are now ready to define a fuzzy extractor and a secure sketch.

Definition 2.1. An $(\mathcal{M}, m, \ell, t, \epsilon)$-fuzzy extractor is a pair of polynomial-time randomized procedures, $(\mathsf{FE.Gen}, \mathsf{FE.Rep})$ with the following properties:

- $\mathsf{FE.Gen}(w) \mapsto (R, P)$: on input $w \in \mathcal{M}$, the generation algorithm outputs an extracted string $r \in \{0,1\}^\ell$ and a helper string $P \in \{0,1\}^*$.
- $\mathsf{FE.Rep}(w', P) \mapsto R$: takes an element $w' \in \mathcal{M}$ and a bit string $P \in \{0,1\}^*$ as inputs.

They satisfy the following two properties.

Correctness. For any templates $w, w' \in \mathcal{M}$, the replication algorithm has the property that

$$d(w, w') \leq t \implies \mathsf{FE.Rep}(w', P) = R \text{ where } (P, R) \leftarrow \mathsf{FE.Gen}(w).$$

If $d(w, w') > t$, then no guarantee is provided about the output of $\mathsf{FE.Rep}$.

Security. For any distribution W of templates over $\mathcal{M}$ s.t. $\mathbf{H}_\infty(W) \geq m$, R is indistinguishable from the uniform distribution over $\mathcal{M}$ conditioned on P.

The definition intuitively says the following. Firstly, the fuzzy extractor's Gen algorithm takes a template w, and generates a random-looking string R along with some public data P. Then, upon a different input w' that is close to the original input, the Rep

algorithm uses the public info P to reproduce the original string R. In conclusion, with the aid of some helper information, a fuzzy extractor constructs the same random string from sufficiently similar inputs.

One way to construct a fuzzy extractor is via a *secure sketch*. A secure sketch generates some sort of information (a "sketch") s about its input w. Then, when receiving a sufficiently similar input w' and the sketch s, it reconstructs the original input w. This makes the work of the fuzzy extractor quite simple: it can simply calculate the hash $H(w)$, get the sketch s of w from the secure sketch. When it receives a new biometric measurement w', it then uses s to reconstruct w, and calculates $H(w)$ again. All in all, the fuzzy extractor simply hashes certain values, and the secure sketch performs the reconstruction. We now give the definition of a secure sketch.

Definition 2.2. An $(\mathcal{M}, m, m', t)$-**secure sketch** (SS) consists of a pair of polynomial-time randomized procedures $(\mathsf{SS.Gen}, \mathsf{SS.Rec})$ described below.

- $\mathsf{SS.Gen}(w) \mapsto s$: The sketch generation algorithm that outputs a bitstring $s \in \{0,1\}^*$ (a "sketch") from a template $w \in \mathcal{M}$.
- $\mathsf{SS.Rec}(w', s)$: The recovery algorithm that takes a template $w' \in \mathcal{M}$ and a bitstring s as inputs.

They satisfy the following two properties.

Correctness. For any $w, w' \in \mathcal{M}$, it holds that

$$w = \mathsf{SS.Rec}(w', \mathsf{SS.Gen}(w)) \text{ if } d(w, w') \leq t.$$

If $d(w, w') > t$, then no guarantee is given about the output of $\mathsf{SS.Rec}$.

Privacy. For any distribution W over $\mathcal{M}$, it holds that

$$\mathbf{H}_\infty(W|\mathsf{SS.Gen}(W)) \geq m', \text{ if } \mathbf{H}_\infty(W) \geq m.$$

In the lattice construction, the sketch s is of the same type as the templates $w \in \mathcal{M}$, that is, a vector of numerical values.

The correctness and privacy notions here signify that for similar templates, the secure sketch reconstructs to the original template, and that the public info obtained from the secure sketch gives a known change (rise) in the predictability of the template. It would be beneficial that the change is small, i.e. that m' is not much smaller than m, since that would ensure that the sketch does not leak too much information about the template.

It turns out that a secure sketch implies a fuzzy extractor, but in order to show that, we need to define an average-case strong extractor.

Definition 2.3. Let $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^l$ be a polynomial time probabilistic function which uses r bits of randomness. We say that Ext is an efficient average-case (n, m, ℓ, ϵ)-strong extractor if for all pairs of random variables (W, I) such that W is an n-bit string satisfying $\mathbf{H}_\infty(W \mid I) \geq m$, we have $\mathrm{SD}((\mathsf{Ext}(W; X), X, I), (U_\ell, X, I)) \leq \epsilon$, where X is uniform on $\{0,1\}^r$.

The definition, as well as a lemma showing that universal hash functions are average-case strong extractors, can be found in [DORS06]. Incidentally, that paper also shows how a secure sketch can imply a fuzzy extractor:

Lemma 2.1. *Assume* $(\mathsf{SS.Gen}, \mathsf{SS.Rec})$ *is an* (M, m, m', t)*-secure sketch, and let* Ext *be an average-case* (n, m, ℓ, ϵ)*-strong extractor. Then the following pair* $(\mathsf{FE.Gen}, \mathsf{FE.Rep})$ *is an* $(M, m, \ell, t, \epsilon)$*-fuzzy extractor:*

- $\mathsf{FE.Gen}(w; r, x)$*: set* $P = (\mathsf{SS.Gen}(w; r), x)$*,* $R = \mathsf{Ext}(w; x)$*, and output* (R, P)*.*
- $\mathsf{FE.Rep}(w', (s, x))$*: recover* $w = \mathsf{SS.Rec}(w', s)$ *and output* $R = \mathsf{Ext}(w; x)$*.*

All in all, a secure sketch implies a fuzzy extractor, since if there is a mechanism to always get a specific template from sufficiently similar ones, then a fuzzy extractor needs only to hash that original template with a salt and send the hash to a server for storage.

Algorithm 1 $\mathsf{FF.Gen}(w)$ from $\mathsf{SS.Gen}(w)$

$r \leftarrow \{0,1\}^\lambda$
$s := \mathsf{SS.Gen}(w)$
$P := (s, r)$
$R := H(w, P)$
return (R, P)

Algorithm 2 $\mathsf{FF.Rep}(w', P)$ from $\mathsf{SS.Rec}(w')$

Parse P as (s, r)
$\tilde{w} := \mathsf{SS.Rec}(w')$
return $H(\tilde{w}, P) \in \{0,1\}^*$

Algorithms 1 and 2 in [ZCY21] (also algorithms 1 and 2 here) show this approach in constructing a fuzzy extractor.

2.3 Preliminaries on lattices

In cryptography, a lattice in $\mathbb{R}^n$ is the set of all integral linear combinations of any k linearly independent vectors in $\mathbb{R}^n$. That is, for linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_k \in \mathbb{R}^n$, a lattice is any set

$$\Lambda = \left\{ \sum_{i=1}^{k} x_i \mathbf{b}_i \mid x_i \in \mathbb{Z} \right\}.$$

Basic theory about lattices as well as a treatment of the closest vector problem, a problem implicitly relevant here, can be found in e.g. [MG02].

Three lattices relevant to this project were the D_8, E_8 and Leech lattices. The D_8 and E_8 lattices are defined as follows:

$$D_8 = \{(x_1, \ldots, x_8) \mid x_1, \ldots, x_8 \in \mathbb{Z} \wedge \sum_{i=1}^{8} x_i \equiv 0 \pmod 2\}$$

and

$$E_8 = D_8 \cup D_8 + \frac{\mathbf{1}}{\mathbf{2}},$$

where $\frac{\mathbf{1}}{\mathbf{2}} = (\frac{1}{2}, \ldots, \frac{1}{2})$ is an 8-dimensional vector with entries of only $\frac{1}{2}$. Intuitively, D_8 consists of such 8-dimensional integer vectors whose sum of entries is even. The Leech lattice, consisting of certain 24-dimensional vectors, has a more complex definition, which can be found in [CS86].

The process of mapping a real vector to its closest lattice point is called **decoding**. Algorithms for decoding to the lattices mentioned above can be found in [CS82] and [CS86], and in [ZCY21].

In this project, an important concept is **scaling** a lattice. If Λ is a lattice and $t > 0$ is a real number, then $t\Lambda$ is the lattice that consists of all points tl where $l \in \Lambda$. Essentially, multiplying by a positive real number can size the lattice up or down ("scale" it). A consequence of scaling a lattice up, for example, is that the lattice points are further apart – the distance between them is larger. This means that where in a lattice Λ some cluster of real vectors would get mapped to different points since the vectors are closest to different points, they might all get mapped to the same one in a scaled-up lattice $t\Lambda$ where $t \gg 1$.

One more idea will be necessary later: the **Voronoi cell** $V(t\Lambda)$ of the origin marks the real vectors that are closer to the origin $(0, \ldots, 0)$ than to any other point on the lattice $t\Lambda$. The context around this is elaborated more fully in [ZCY21].

2.4 A lattice-based construction of fuzzy extractors

A face recognition algorithm produces a feature vector of real numbers (also called a template or an embedding) out of an image of a face. In [ZCY21], a fuzzy extraction scheme adapted to facial templates is implemented using lattices. The intuitive idea behind this construction is that face recognition algorithms create feature vectors that are close to each other under Euclidean distance, and points close to each other in a Euclidean sense will be decoded to the same lattice point.

The secure sketch algorithms are shown in algorithms 3 and 4, and the fuzzy extractor is constructed out of that as shown above.

Algorithm 3 Generalized construction for $\mathsf{SS.Gen}(w)$

$c = \mathsf{decode}(w)$
$s = c - w$
return $s \in \mathbb{R}^n$

The security parameters for this construction are as follows:

Theorem 2.2. *The secure sketch* $(\mathsf{SS.Gen}, \mathsf{SS.Rec})$ *in the algorithms 3 and 4 is an* $(\mathcal{M}, m, m - \log|V(t\Lambda) \cap \mathbb{Z}^n|, t)$*-lattice-based secure sketch.*

The proof of this result and an abundance of details on lattice-based secure sketches are found in [ZCY21].

Algorithm 4 Generalized construction for $\mathsf{SS.Rec}(w', s)$

$c' = w' + s$
$c^* = \mathsf{decode}(c')$
$\tilde{w} = c^* - s$
return $\tilde{w} \in \mathcal{M}$

2.5 False acceptance and false rejection rates

The key idea in the following is that as we make the lattice larger – i.e. space the points out more – more and more templates get decoded to the same lattice point.

In an extreme case, if we decode to some lattice $t\Lambda$ where t is really large, all templates might get decoded to the same lattice point. In that case, the offset s returned by $\mathsf{SS.Gen}(w)$ leaks a lot of information about the template w; in fact, it uniquely describes the template. In that case, $\mathsf{SS.Rec}(w', s)$ will yield the template w for any other template, regardless of whether it comes from the same person or not. In the case of fuzzy extractors, this would mean that if the lattice we decode to is extremely large, any person can authenticate themselves as any other person if they know their public information $P = (s, r)$. Situations where a person successfully authenticates themselves as someone else is called a **false acceptance**.

On the other hand, though, if we decode to some $t\Lambda$ where t is really small, every template might map to a different lattice point. This means that the offset s leaks very little about the template, but also that if t is extremely small, even templates from the same person are mapped to different lattice points. This would mean that for some original template w of a person x, it's possible that $\mathsf{SS.Rec}(w', s)$ gives a different template from w, and the person cannot authenticate themselves. This situation is called a **false rejection**.

So if the lattice is too large, the security of the scheme evaporates under the high false acceptance rate but if it is too small, the high false rejection rate plummets the usability. In this situation, we should then find a scale factor of the lattice we use where both rates are acceptable to us. In addition, it is important to know the min-entropy of the templates when we know the public information P or sketch s. Estimating that has a more elaborate method, but is discussed quite thoroughly in [ZCY21], as well as in the implementation section of this report.

2.6 PINs in SplitKey

In SplitKey, a PIN is used to safeguard a share d_1' of the secret key d. The share d_1' is stored on the Client's device, encrypted by AES. In order to decrypt to the share, a PIN needs to be entered. If the PIN is the same as the one entered upon registration, the encryption opens to the share d_1', but if it is not the same, the encryption still opens, but not to the share d_1'.

With a fuzzy extractor, the value $R := H(w, P)$ would fill the role of the PIN. The analogous use would be that upon registration, a facial template w of the user is taken. When the user wishes to decrypt the share, it takes another facial template w'. Now, we would need the encryption to open to the correct share d_1' if the two templates w and w' come from the same person, i.e. are close enough. If the two templates are too far, though, the encryption should open to something different.

3 Implementation process and results

3.1 Goal of the implementation

The goal of implementing the scheme for ourselves was to evaluate whether or for what purpose the scheme could be used in SplitKey. We wished to test whether we get similar results from the scheme, and whether there are some scalings of a lattice such that the false acceptance rate, false rejection rate and entropy combination suit our use case.

3.2 Outline of the process

The methodology was as follows. First, we'd take a dataset's worth of people's face images. We would find the templates of all of them with a face recognition algorithm, and quantize the templates (map them to integers, done by multiplying with a large number and rounding). After that, we would run three tests with the data: false acceptance rate tests, false rejection rate tests and entropy estimation.

For false acceptance, we chose two random templates w_1, w_2 from different people. We would test whether $\mathsf{SS.Rec}(w_1, \mathsf{SS.Gen}(w_2)) = w_2$ and vice versa. If yes, then we would count that as a false acceptance because the sketch obtained from w_2 allowed another person's template to be reconstructed to w_2. We would repeat this test $10,000$ times and record the frequency of the false acceptances.

For false rejections, we created a folder containing files of people, that is, files that contained one person's different templates. For each file (e.g. a file AaronPeirsol.txt containing Aaron Peirsol's templates), we would pick the first template w_0 to be the initial one, and test for every other template w whether $\mathsf{SS.Rec}(w, \mathsf{SS.Gen}(w_0)) = w_0$. If not, we would count that as a false rejection since the templates are, in fact, from the same person, and the correctness of the secure sketch is not fulfilled.

We would try to find a scaling factor t of the lattice where the false rejection rate is acceptable, but the false acceptance rate is low enough. At that scaling factor, we would

estimate the minimum entropy of the templates conditioned on the public information s. The latter we did in a way described in [ZCY21], as follows.

For many different scaling factors t, we would find the frequency p_t with which templates were decoded to the "most popular" lattice point. The most popular lattice point l would be the lattice point to which the most templates were decoded to, and the frequency would then be the amount of templates decoded to it over the amount of all templates. In other words, $p_t = \frac{|\{x \in X | \mathrm{decode}(x) = l\}|}{|X|}$ if X is the set of all templates.

As t grows, p_t would grow as well, giving a graph similar to a cumulative distribution function. We would then fit different probability distributions' CDFs to the graph we obtained, and test whether our graph is sufficiently close to the CDFs under the estimated parameters. For that, we would first simply plot the p_t values and the graph of the CDF, and if it looked like a plausible match, we would conduct the Kolmogorov-Smirnov test [Mas51].

3.3 Code layout

The code is written in Python.

The first step was to get a face recognition algorithm to be able to convert images to templates. At first, we used face-recognition, but later, to be able to estimate the security the way it was done in the original article, we switched to ArcFace instead. We also quantized the templates, which required no additional libraries. To be able to test false acceptance and false rejection and entropy efficiently, we converted the Labeled Faces in the Wild dataset [HMBLM08] into template. The LFW dataset consists of about 13,000 photos of over 5,000 people. Later, to fit distributions to our graphs, we used scipy, and to plot the data, we used matplotlib.

After getting templates, the cryptographic primitives needed implementing. For that, we used pycryptodome's hash functions (as well as some of their random and serialization methods) and numpy. The implementation was not heavy in advanced crypto, so that was all that was needed.

The code is written in a object-oriented way where there are classes for a fuzzy extractor, a secure sketch, and a decoder. A decoder is a class that contains a generic decoding method, which allows to decode to different lattices like D_8, E_8 and the Leech lattice. The code also has client and server classes, which can simulate a toy authentication protocol as given in [ZCY21], and a camera class which obtains templates from webcam input.

3.4 On lattices and quantization

The Leech lattice decoding algorithm was not tested because its operating time was very long. For that reason, the lattice we ran tests on was E_8. We initially ran the tests with face-recognition, but then switched to ArcFace. The relevant, usable results come from the latter situation.

In order to operate with integers and not floats – something that is quite necessary for cryptographic purposes – the float vectors need to get mapped to integers. Since that step

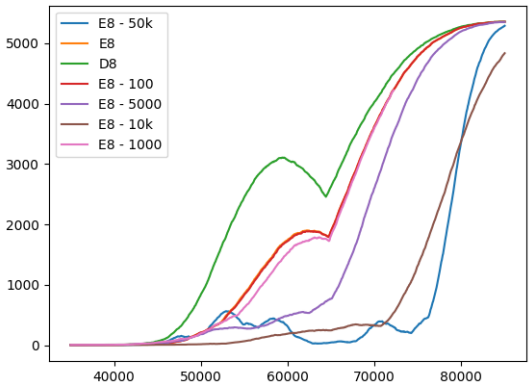

Figure 1: p_t values (before normalising) for different lattices when templates were obtained from face-recognition

was not clarified in the original paper, we initially did it in a way where we truncated the float to 5 decimal places and multiplied that number by 10^5 to get an integer. Having tested the FAR, FRR and min-entropy and received unsatisfying results, we emailed the authors of [ZCY21] to check what method they used.

The authors had instead multiplied all the vector components by 2^{20} and rounded the result. After updating my quantization algorithm, we tried the same tests again.

3.5 Results

At first, we ran the tests with face-recognition, on various different lattices. However, the frequency of decoding to the most common lattice point, p_t (the data we used to estimate min-entropy), did not grow monotonously with the scale factor. That is, as the lattice got scaled up, it was not necessarily the case that more and more templates would get decoded to the same lattice point. This meant the approach of viewing the p_t values as points on a CDF was not possible.

These results are illustrated on figure 1. The x-axis shows the scaling factors t of the lattices, and the y-axis the raw amount of templates that were decoded to the most common lattice point. The lattices used are D_8 and E_8 as defined above, and the notation $E_8 + C$ for some number C means that all templates were shifted by a vector $(C, \ldots, C)$ before decoding.

Then, we turned to using ArcFace, which made it possible to use the CDF approach to estimating entropy. Since the original paper found a very accurate match with the Johnson's S_U distribution, we tried the S_U and S_B distributions as well. These distributions are transformations of the normal distribution originally introduced in [Joh49].

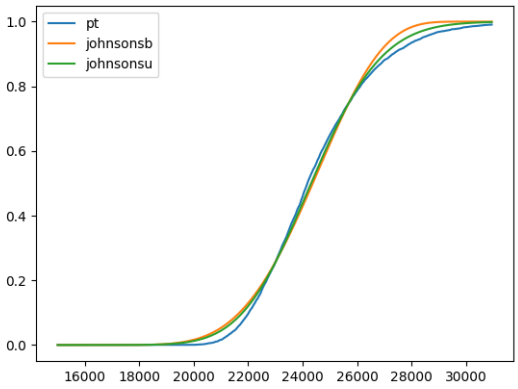

Figure 2: p_t values for the E_8 lattice, with ArcFace and templates quantized via truncation

The fitted distribution functions of the Johnson's S_U and Johnson's S_B distributions are shown on figure 2. When quantizing by truncating and multiplying, however, the FRR and FAR did not line up with the original article. For example, at a false rejection rate of 70%, the false acceptance rate was already 3% in my tests. At that rejection rate, the entropy was also low: only 12 bits. If we were to drop the false rejection rate even to 62.5%, the false acceptance rate would be 5% and the entropy 9 bits. Since the CDFs at these scaling factors are inaccurate (they have larger values than the p_t values we found), we used the empirical frequencies to find the entropy here.

We wanted to know if maybe the results could be improved by aligning the quantization method with the one the authors used. We emailed them about this and some other questions about the scheme, and implemented their quantization method of multiplying by 2^{20} and rounding. Here, the false acceptance rates, false rejection rates and entropy are similar. At an FRR of 61.25% and an FAR of 5%, the entropy of the scheme was 9 bits according to S_B and 7.5 bits according to S_U. At a 70% FRR, 3% FAR, the entropy was 5 bits according to both distributions. The graph of the new p_t values with the fitted S_U and S_B distributions is shown on figure 3.

The entropy estimation might be fixed by improving the fitting of CDFs, though. As can be seen on the graphs, none of the CDFs fit quite as well as the one in the original paper fits with their data. This means that especially the lower end of probabilities might not be modelled correctly. That is important, though, because the probability of decoding to the most popular lattice point has to be at least $1/|D|$, where D is the dataset size. However, as the lattice gets scaled down, the probability of decoding to the lattice point should still keep dropping – it just cannot be shown with a finite dataset. Thus, obtaining a CDF that fits the data better, choosing better parameters that can accurately model the lower end of probabilities is important in seeing whether higher entropy levels are possible. The same point is detailed in the original paper as well.

Nevertheless, even if we can fit the CDF to our data more accurately, the issue remains

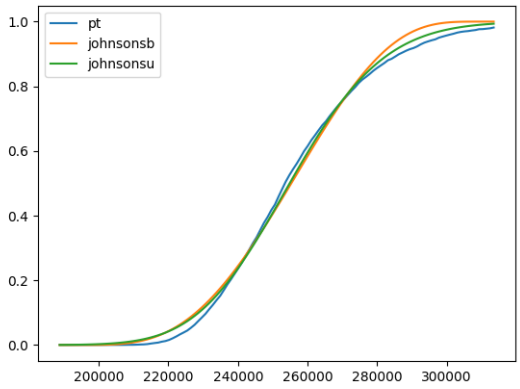

Figure 3: p_t values under ArcFace and new quantization

that at such low scaling parameters, the false rejection rate tends to be very high.
We have some guesses as to why the results are like that:

- The parameters of the CDFs obtained from scipy's curve_fit did not give a graph that is quite exactly aligned with the graph we obtained, which may alter entropy results, but not FAR and FRR.
- The methodology of testing false acceptance and false rejection might be different in the original article, which may cause different results.
- There may be errors in the decoding algorithm, although that seems a bit less likely.

The entropy estimates get better, the smaller the scaling factor is of the lattice. However, in these cases, the usability at sufficiently high levels of entropy was not acceptable.

4 Conclusion

4.1 Open questions and further directions

There are many avenues to keep improving the prototype. To name a few,

1. Most importantly, the security level of the scheme is currently not as high as is expected from the results of the article. The reason for this should be determined.
2. In case the client's device's camera is malicious, it should be useful to detect liveness from the facial template information the fuzzy extractor receives. In order to do that, it might be necessary to send many templates. In any case, research

into the relationship between the input image and the output template is needed to implement this.

3. The Leech lattice decoding, while implemented, is slow, which makes testing difficult. It might be useful to find a way to speed up the decoding or testing processes.

References

[CS82] J. Conway and N. Sloane. Fast quantizing and decoding and algorithms for lattice quantizers and codes. *IEEE Transactions on Information Theory*, 28(2):227–232, 1982.

[CS86] J. Conway and N. Sloane. Soft decoding techniques for codes and lattices, including the golay code and the leech lattice. *IEEE Transactions on Information Theory*, 32(1):41–50, 1986.

[DORS06] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *CoRR*, abs/cs/0602007, 2006.

[HMBLM08] Gary B. Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. In *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, Marseille, France, October 2008. Erik Learned-Miller and Andras Ferencz and Frédéric Jurie.

[Joh49] N. L. Johnson. Systems of frequency curves generated by methods of translation. *Biometrika*, 36(1/2):149–176, 1949.

[Mas51] Frank J. Massey. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[MG02] Daniele Micciancio and S. Goldwasser. *Complexity of Lattice Problems*. Kluwer Academic Publishers, USA, 2002.

[ZCY21] Kaiyi Zhang, Hongrui Cui, and Yu Yu. Facial template protection via lattice-based fuzzy extractors. Cryptology ePrint Archive, Paper 2021/1559, 2021. https://eprint.iacr.org/2021/1559.