Private Information Retrieval applied to Biometric Verification

Martijn P. de Vries
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
m.p.devries@student.utwente.nl

April 2022

Contents

Abstract

Biometric verification is convenient, since you always have your 'key' with you, so there is no risk of forgetting or losing it. However, it also poses a security risk, since biometric characteristics are hard to keep secret and can contain privacy sensitive medical information [1]. To keep this secure biometric template protection can be applied in various ways, including feature transformation and biometric cryptosystems. We propose using an oblivious data structure, namely private information retrieval, to retrieve a biometric template and similarity scores in a biometric verification system. The results show that retrieving the template in most cases is too slow for practical use, but retrieving the similarity scores shows promise for small template sizes.

1 Introduction

Biometric verification is a form of identity verification that uses the principle of 'something you *are*', as opposed to 'something you *have*' (e.g. a key) or 'something you *know*' (e.g. a password). This is achieved through biometric recognition that automates the recognition of individuals based on their biological and behavioural characteristics, such as fingerprints or gaits.

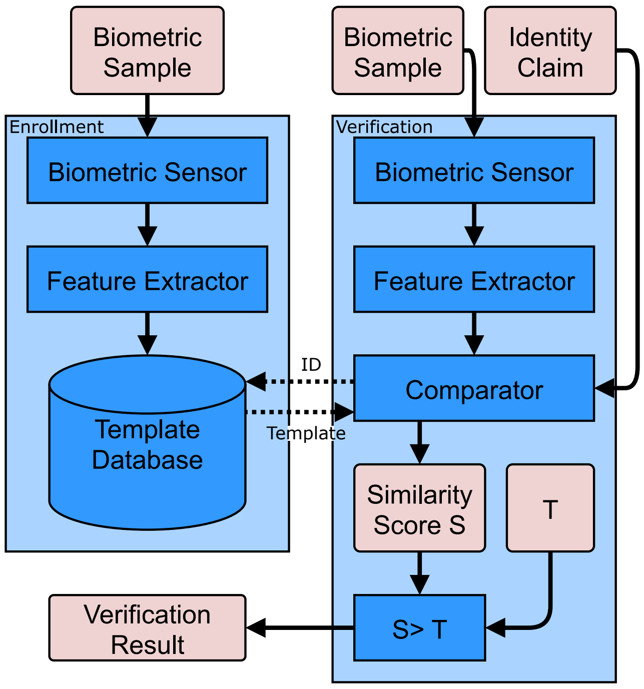


Figure 1: A diagram of a biometric verification system. The identity claim is accepted when the similarity score S exceeds the threshold T, and rejected otherwise.

In general, a biometric verification system comprises two phases. First, there is the *enrollment phase*, where a biometric sensor (also referred to as client) captures the user's biometric sample. A feature extractor extracts features from the user's raw biometric data in the form of a feature vector. This feature vector represents a template, that is stored in a template database on a server under a unique identification key, where it can later be retrieved and used as a biometric reference. Second, there is the *verification phase*. The first two steps of the verification phase are the same as the enrollment phase, producing a feature vector called probe from a live biometric sample. In the verification phase the user makes an identity claim and the client fetches the corresponding template from the server's database. The probe is compared to the template and a similarity score is computed. The identity claim is accepted if the similarity score S exceeds the threshold T, and rejected otherwise. A diagram of such a system is shown in Figure 1.

Biometric verification is appealing, since it is convenient to use and there is no risk of forgetting or losing something. On the other hand, biometric characteristics are hard to keep secret and impossible to replace. For example, a person's fingerprints are left everywhere and can easily be obtained, but are impossible to replace once lost. Additionally, some biometric characteristics contain medical information of individuals that is privacy sensitive and should be kept secret [1]. The inherent disadvantages of biometric characteristics also threaten the security of biometric verification systems, since templates and probes leak sensitive information related to individuals and thus must be protected.

To remedy this, biometric template protection can be applied, as shown in the research by Jain et al. [2]. Jain et al. broadly classify template protection schemes into two main categories, namely, feature transformation and biometric cryptosystem. Another category is biometric recognition in the encrypted domain, which is used in Peeters et al. [3]. Peeters et al. propose a protocol that is designed to prevent the server from finding out the user's biometric information and the result of the verification. It also prevents the user from accessing the biometric information of other users stored in the server's database. It does so by leveraging the ElGamal homomorphic encryption scheme [4] and likelihood-ratio-based biometric verification [5] to achieve secure biometric verification in the semi-honest model.

Homomorphic encryption allows arithmetic operations to be performed on ciphertexts. When a ciphertext is decrypted, the result will be the same as if the arithmetic operations had been applied to the plaintext. A biometric comparison can be performed in the encrypted domain using homomorphic encryption to keep the probe and template secret. To apply homomorphic encryption Peeters et al. propose using the precomputed log-likelihood ratio classifier. This method generates a lookup table with similarity scores for each feature in the feature vector. The probe and template select a column and row, respectively, and the sum of all selected scores is checked against the threshold to determine an accept or reject.

Protected table lookup can also be used with other cryptographic systems such as oblivious data structures. Two common oblivious data structures are Oblivious RAM (ORAM) and Private Information Retrieval (PIR). ORAM was originally proposed by Goldreich and Ostrovsky [6]. It provides a way for a user with a database stored on (multiple) untrusted remote server(s) to hide their access pattern to their database. Similarly, a PIR protocol allows a user to retrieve information from a database on a remote server without revealing what item has been retrieved. PIR is the more fitting option, since ORAM was designed for a single client with a private database, whereas PIR was designed with multiple clients and a public database in mind. In the case of biometric verification there are usually multiple clients that require access to the same database.

PIR was first introduced by Chor et al. [7][8], who used a multi-server approach. The downside of storing the database on multiple servers is that the amount of storage needed is multiplied by the amount of servers used. On the other hand, a single server approach requires less storage, but in return is more costly computation-wise. One such single server PIR protocols was proposed by Dong and Chen [9].

With the properties of the precomputed log-likelihood ratio classifier and PIR in mind we ask ourselves the question:

> What are the consequences of applying private information retrieval to biometric verification with the precomputed log-likelihood ratio classifier?

To answer this question, we identified two possible applications of PIR to biometric verification with the precomputed log-likelihood ratio classifier. First, we present a method to hide the user's identity from the server by using PIR to retrieve the user's template. Second, we present a method to retrieve the similarity scores relevant to the probe directly from the template using PIR.

The rest of the paper is structured as follows. In section 2 we provide more background information on the concepts introduced above. In section 3 we look at work related to this paper. With section 4 we show the designs we came up with and in section 5 we explain the experiments we conducted with implementations following these designs and show their results. Next, in section 6 we discuss the results and finally, in section 7 we present our conclusion.

2 Background

In this section we will provide additional background information on homomorphic encryption, the precomputed log-likelihood ratio classifier, and PIR, which were introduced in the previous section. We start with a short section on notation used in the paper.

2.1 Notation

For the rest of the paper we will be using the following notation. Single values are denoted with a regular letter x, vectors are denoted with an arrow over a letter $\vec{x}$, and bit strings (a vector where each value is either a 0 or a 1) are denoted with a bold letter $\mathbf{x}$. The sum of multiple values, like the sum of a vector, is denoted with a capital letter X. Finally, we denote matrices with a bold capital letter $\mathbf{X}$. Other notations are introduced as their respective concepts are explained.

2.2 Homomorphic Encryption

Homomorphic encryption (HE) allows arithmetic operations to be performed on ciphertexts. When a ciphertext is decrypted, the result will be the same as if the arithmetic operations had been applied to the plaintext. A fully homomorphic encryption (FHE) scheme can perform an arbitrary amount of operations on a ciphertext of multiple types. Other types of HE, such as somewhat homomorphic encryption (SHE) and partially homomorphic encryption (PHE), are more restricted and can only perform a limited amount of operations of certain types. Therefore, FHE offers the most functionality out of all HE types.

Peeters et al.'s protocol uses ElGamal, a PHE scheme. Its security is based on the Decisional Diffie-Hellman (DDH) assumption [10]. A PHE algorithm can only evaluate one type of operation, e.g. addition or multiplication. ElGamal is multiplicative homomorphic, but can be made to be additive homomorphic. ElGamal is built around a cyclic group G of order p and generator g. ElGamal can be made to be additive homomorphic by encoding the message m as an exponent of the generator g^m: $[\![g^m]\!][\![g^{m'}]\!] = [\![g^{m+m'}]\!]$. However, it can only be decrypted for a small message space, i.e. $m \in \{0, 1\}$.

Dong and Chen's protocol uses BGV [11], which is an FHE scheme. BGV's security is based on the ring-LWE (RLWE) [12] problem. Let $\Phi_m(x)$ be the

m-th cyclotomic polynomial with degree $\phi(m)$, where $\phi(\cdot)$ represents Euler's totient function. We then have a polynomial ring $\mathbb{A} = \mathbb{Z}[x]/\Phi_m(x)$. The ciphertext space of BGV then consists of polynomials over $\mathbb{A}_q = \mathbb{A}/q\mathbb{A}$, which are elements in $\mathbb{A}$ reduced modulo q, where q is an odd integer. Similarly, the plaintext space consists of the ring $\mathbb{A}_2 = \mathbb{A}/2\mathbb{A}$, which are binary polynomials of degree up to $\phi(m) - 1$.

In the rest of this paper we will denote homomorphic operations the following way. Homomorphic addition with $\boxplus$, homomorphic multiplication with $\boxtimes$, and homomorphic rotation with $\lhd$ (shift left) and $\rhd$ (shift right).

2.3 Precomputed log-likelihood ratio classifier

A biometric verification system determines whether a user's biometric signature is a match by checking a similarity score against a threshold. The similarity score is calculated from a biometric comparison. Performing this calculation under encryption is expensive, therefore Peeters et al. [3] suggest using precomputed log-likelihood ratio classifiers. This section gives a short summary on how Peeters et al.'s biometric verification system works, for more information, refer to their paper [3].

The precomputed log-likelihood ratio classifier has a lookup table for each feature f. It is denoted by $\mathcal{T}_{b,f}$ and has a size of $2^b \times 2^b$. It contains all possible similarity scores $s_{x,y}$ of a single, quantized feature, where x and y denote the observation during enrollment and the observation during verification, respectively.

$$\mathcal{T}_{b,f} = \begin{pmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,2^b-1} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,2^b-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{2^b-1,0} & s_{2^b-1,1} & \cdots & s_{2^b-1,2^b-1} \end{pmatrix} \tag{1}$$

For every feature f the x-th row of its lookup table ($\mathcal{T}_{b,f}^{(x)}$) contains all possible similarity scores for that enrollment observation. Any verification observation y will then select the y-th column and thus the individual score $s_{x,y}$ for that feature. Each feature has its own lookup table, so for a biometric sample described by a k-dimensional feature vector, k lookup tables $\mathcal{T}_{b,f_0}, \mathcal{T}_{b,f_1}, \ldots, \mathcal{T}_{b,f_{k-1}}$ can be constructed. The template is formed by taking the row of each lookup table that corresponds to the enrollment observation for each feature. For each row in the template there exists a column that corresponds to the verification observation of that feature. The sum of the individual scores yields the final score that is then compared against a biometric threshold.

2.3.1 Template encryption

As mentioned in 2.2, Peeters et al. have opted to use ElGamal to keep the probe and template private. ElGamal can be used in additive homomorphic mode by using a small message space. This property is used in the verification protocol to determine an accept or reject.

Another property of ElGamal is that it can be used in a threshold variant, which was first shown by Desmedt and Frankel [13]. The threshold variant allows users to perform a partial decryption by splitting the secret key into two secret shares $sk = sk_1 + sk_2$. No single partial key can fully decrypt the message and a partially decrypted ciphertext is indistinguishable from a full encryption. For notation we use double brackets to denote an encrypted value $[\![\cdot]\!]$ and single brackets to denote a partially decrypted ciphertext $[\cdot]$. For a secret key sk with 2 secret shares sk_1 and sk_2 a message m is encrypted by applying the encryption function E with the joint public key pk, we get $E_{pk}(m) = [\![m]\!]$. The message can be partially decrypted using the function D with secret share sk_1, we get $D_{sk_1}([\![m]\!]) = [m]$. The full decryption uses the same function D with the remaining secret share sk_2, we get $D_{sk_2}(D_{sk_1}([\![m]\!])) = m$.

In Peeters et al.'s protocol the message to be encrypted is the user's template and the secret key is shared among a client and a server. By applying homomorphic operations the template is transformed into a similarity score and checked against a threshold, the result of which the server decrypts partially. The client then applies the final decryption, revealing the result of the verification.

2.3.2 Biometric verification protocol

In Peeters et al.'s protocol, there are two parties, the *sensor device* and the *verification service*. The sensor device captures the user's biometric data and receives the end result of the comparison. The verification service holds the templates and does the threshold comparison.

Recall from the introduction that a biometric verification system consists of two phases, the enrollment phase and the verification phase. During the enrollment phase of Peeters et al.'s protocol, a feature vector $\vec{r}$ is sampled, where each element r_i selects the r_i-th row of the i-th feature's lookup table. These rows are organized in a list to generate the template $\boldsymbol{T}_u$ for user u:

$$\boldsymbol{T}_u = \left((\mathcal{T}_{b,f_0}^{(r_0)})^\top | (\mathcal{T}_{b,f_1}^{(r_1)})^\top | \ldots | (\mathcal{T}_{b,f_{k-1}}^{r_{(k-1)}})^\top \right)^\top \tag{2}$$

The sensor device encrypts the template $\boldsymbol{T}_u$ with the joint public key of the threshold cryptosystem ElGamal, resulting in $[\![\boldsymbol{T}_u]\!]$. They then send it to the verification service, which will store it in their database with the user's identity u as the unique identifier.

Then the verification phase begins, which is sketched in Figure 2. The protocol starts with the sensor device calling *Capture()* to get the quantized feature vector $\vec{p}$ and sending an identity claim u to the verification service. The verification service uses u to find the corresponding encrypted template $[\![\boldsymbol{T}_u]\!]$ and sends it to the sensor device.

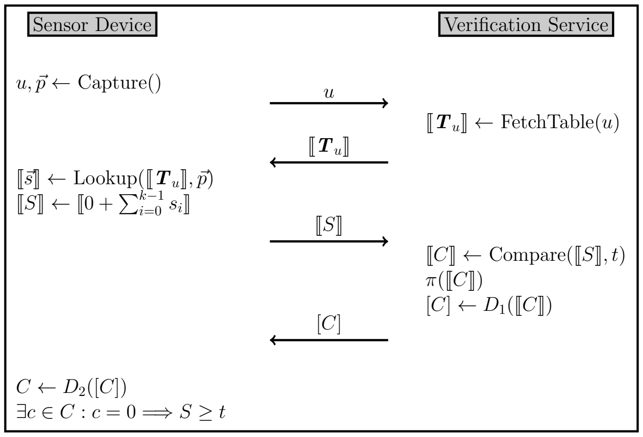

Sensor Device
Verification Service
$u, \vec{p} \leftarrow$ Capture()
u
$[\![\boldsymbol{T}_u]\!] \leftarrow$ FetchTable(u)
$[\![\boldsymbol{T}_u]\!]$
$[\![\vec{s}]\!] \leftarrow$ Lookup($[\![\boldsymbol{T}_u]\!], \vec{p}$)
$[\![S]\!] \leftarrow [\![0 + \sum_{i=0}^{k-1} s_i]\!]$
$[\![S]\!]$
$[\![C]\!] \leftarrow$ Compare($[\![S]\!], t$)
$\pi([\![C]\!])$
$[C] \leftarrow D_1([\![C]\!])$
$[C]$
$C \leftarrow D_2([C])$
$\exists c \in C : c = 0 \Longrightarrow S \geq t$

Figure 2: Peeters et al.'s biometric verification protocol [3].

The template $[\![\boldsymbol{T}_u]\!]$ consists of k rows and the probe $\vec{p}$ consists of k values, one per feature. For each feature f_i, with $0 \leq i < k$, the sensor device selects the p_i-th column of the row $[\![\boldsymbol{T}_u]\!]_i$, essentially performing a lookup of $\mathcal{T}_{b,f_i}$ with r_i as row and p_i as column. This results in a vector of encrypted similarity scores $[\![\vec{s}]\!]$. The scores are summed under encryption using ElGamal's homomorphic property to get $[\![0+\sum_{i=0}^{k-1} s_i]\!] = [\![S]\!]$. The 0 is added to randomise the outcome of the sum. This prevents an attacker who has the encrypted template from guessing the elements which yield the same value as $[\![S]\!]$, since that would leak the probe.

$[\![S]\!]$ is sent to the verification service, which compares it to a threshold t. Unfortunately, the comparison operation does not exist in homomorphic encryption, but checking for equality is possible $[\![S - t]\!] = [\![0]\!]$. Due to the quantization steps in creating the biometric template, the score distribution $\mathbb{S}$ becomes finite and therefore there exists a maximum value for the scores, $\max(\mathbb{S})$. By calculating $[\![S - t - i]\!]$ for all $0 \leq i \leq \max(\mathbb{S}) - t$ it is possible to check whether the value of S lies between the threshold and the maximum score, essentially checking whether S is greater or equal to t.

So instead of comparing $[\![S]\!]$ to the threshold, a result set $[\![C]\!]$ is calculated as $[\![C]\!] = \{[\![r(S - t - i)]\!] | \forall 0 \leq i \leq \max(S) - t, r \in_R [1, |G|]\}$, where r is a random value used as a multiplicative blind to hide the scores. The order of the result set is then scrambled using a random permutation $\pi([\![C]\!])$.

The verification service then partially decrypts the result set and sends it to the sensor device as $[C]$. The sensor device fully decrypts it and can check whether the result set contains a 0. The claim is accepted if, and only if $S \geq t$; otherwise it is rejected.

2.4 Private information retrieval

PIR was first introduced by Chor et al. [7], [8]. A PIR protocol allows a user to retrieve data from a database on a server without the server finding out what has been retrieved. In the trivial case, a client can download the entire database and retrieve the information it needs without the server finding out anything. Obviously, for large databases this approach is impractical as the communication complexity is $\mathcal{O}(n)$ for a database of size n.

At first, PIR was studied in the multi-server setting. The database was copied to databases on multiple non-colluding servers and they would jointly answer the queries of the client. The security of multi-server PIR schemes relies on no single server seeing all queries made by the client. The obvious disadvantage of such schemes is the need for extra storage space compared to a single server. Additionally, the servers need to be non-colluding, increasing the amount of involved parties.

Using only a single server avoids those issues, but this approach has problems of its own. To achieve information theoretic PIR in the single-server setting $\Omega(n)$ bits are required for a database of size n, which is already achieved with the trivial case. However, Kushilevitz and Ostrovsky [14] found that this is not the case for computational PIR schemes. They showed that single-server computational PIR schemes can achieve sublinear communication complexity.

2.5 Dong and Chen

A fast, single server PIR scheme was proposed by Dong and Chen [9] (from this point onward, we will refer to this protocol as DC-PIR). This scheme focuses on achieving a low server-side computation complexity, instead of the more traditional low communication complexity. According to Sion and Carbunar [15] nontrivial single-server PIR protocols may often have low communication complexity, but are slower than the trivial solution, due to high server-side computation complexity. Dong and Chen claim that their protocol is fast in comparison to other schemes and focuses on server computation instead of communication, which can lead to a greater overall gain in run-time.

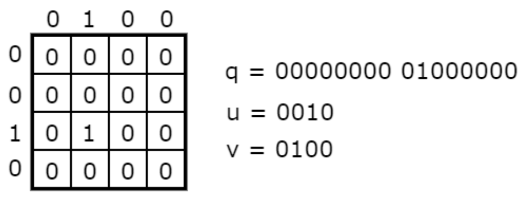
DC-PIR uses the following concept. Let $\mathbf{x}$ be an n-bit integer database on the server of which the client wants to retrieve the i^{th} bit $\mathbf{x}_i$. The server picks an integer $t < n$ and arranges their database into an $n' \times t$ matrix $\mathbf{X}$, where $n' = \lceil \frac{n}{t} \rceil$. $\mathbf{x}_i$ has now become $\mathbf{X}_{jk}$ for some j and k in the matrix, where j represents a specific row in the matrix $\mathbf{X}$, and k a specific column. The client only needs to retrieve the j-th row to find $\mathbf{X}_{jk}$. They do this by creating an n'-bit query string $\mathbf{q} = \mathbf{q}_1\mathbf{q}_2 \ldots \mathbf{q}_{n'}$, such that all bits are 0 except $\mathbf{q}_j$. The query string is sent to the server, which calculates the inner product of $\mathbf{q}$ and $\mathbf{X}$ as $\mathbf{q}_1 \cdot \mathbf{X}_1 + \mathbf{q}_2 \cdot \mathbf{X}_2 + \cdots + \mathbf{q}_{n'} \cdot \mathbf{X}_{n'}$. The resulting inner product equals $\mathbf{X}_j$ (the j^{th} row of the matrix), since only $\mathbf{q}_j$ is 1. The server sends the result back to the client, who checks the k^{th} bit of the result (recall that this bit is $\mathbf{X}_{jk}$), which is the i-th bit of the database $\mathbf{x}$.

2.5.1 Fully homomorphic encryption

Recall from 2.2 that Dong and Chen use BGV [11] to keep the query and retrieved data private during the protocol. The BGV scheme allows packing plaintexts and batching homomorphic computation, which was first observed by Smart and Vercauteren [16]. Packing plaintexts can be done, because the plaintext space $\mathbb{A}_2$ can be partitioned into a vector of plaintext slots. We can factor $\Phi_m(x)$ modulo 2 into l irreducible factors, where each factor has a degree of $d = \phi(m)/l$. We get a mapping $\pi : \mathbb{F}^l_{2^d} \rightarrow \mathbb{A}_2$, which packs l elements in field $\mathbb{F}_{2^d}$ into a single element $\mathbb{A}_2$. This element can be encrypted with BGV, since it is in the BGV plaintext space. To unpack the plaintext we apply the inverse mapping $\pi^{-1} : \mathbb{A}_2 \rightarrow \mathbb{F}^l_{2^d}$. Homomorphic operations can be applied to a packed ciphertext, which will perform the operations on the entire plaintext vector in an SIMD (single instruction multiple data) fashion.

2.5.2 Tree-based compression

To retrieve data privately from the server, DC-PIR creates a query string that is encrypted with BGV. However, applying BGV inflates the communication complexity, therefore, DC-PIR proposes a tree-based compression scheme to reduce the length of the query string. The basic idea of the compression scheme is to fold a query string, consisting of all 0's and a single 1, into a matrix and extract a row and column, which indicate the position of the 1. Let $\mathbf{q}$ be a query string of length $n' = 2^\zeta$, with $\zeta \in \mathbb{Z}^+$, consisting of only 0's except the bit at index j, which is set to 1 (Figure 3 shows an example with $n' = 16$ and $j = 9$). To fold it, a $d_1 \times d_2$ matrix $\mathbf{M}$ is created, which is filled with the query string $\mathbf{q}$ starting from the top leftmost cell and wrapping at the end of each row. The matrix now consists of all 0's and a single 1 at $\mathbf{M}_{\alpha\beta}$. Two strings $\mathbf{u}$ and $\mathbf{v}$ of length d_1 and d_2, respectively, are obtained such that in $\mathbf{u}$ the bit at index α is 1 and in $\mathbf{v}$ the bit at index β is 1. To unfold, a matrix $\mathbf{M}'$ is created and filled using $\mathbf{u}$ and $\mathbf{v}$ such that for each $1 \leq a \leq d_1$, $1 \leq b \leq d_2$, $\mathbf{M}'_{ab} = \mathbf{u}_a \cdot \mathbf{v}_b$. The original query string $\mathbf{q}$ is recovered by concatenating the rows of $\mathbf{M}'$.



$$\begin{array}{c|cccc} & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 3: Example of the folding algorithm (from [9])

The strings $\mathbf{u}$ and $\mathbf{v}$ are also strings with a single bit set to 1, so they can be folded the same way $\mathbf{q}$ was. Folding a query string of length n' repeatedly compresses it into $\log n'$ strings, each 2-bit long. To be deterministic, a tree structure is defined that directs how to fold and unfold a string recursively, hence tree-based compression scheme.

This folding tree is a binary tree, such that each non-leaf node has exactly two children. Each node stores a number that determines the length of the string to be folded or unfolded at that node. To generate such a tree, we provide the length of the query string, store it at the root and split it into two integer shares, with a difference in value of at most 1. We use these shares to recursively generate the node's children. If a share has a value of 2, it becomes a leaf node. An example of a folding tree is shown in Figure 4.

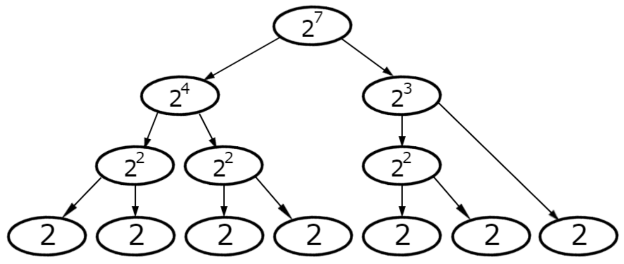


Figure 4: Example of a folding tree generated with a query string of length 2^7, the values show the length of the query string to be folded or unfolded at that node.

2.5.3 Protocol description

We now have all the building blocks to describe DC-PIR. For correctness and security proofs refer to the full paper [9]. Recall we can pack l elements into a single ciphertext and process it in an SIMD fashion. We can use this to run l instances of the basic protocol (second paragraph of subsection 2.5) simultaneously. The database can be represented as an $n' \times l$ matrix with d-bit binary vectors as elements, where d comes from the BGV FHE scheme (see section 2.5.1). The client creates a query string $\mathbf{q}$ of length n', which they fold into $\mathbf{s}$, reducing the length to $2 \log n'$. The client can then pack $\mathbf{s}$ into a single ciphertext and send it to the server. This works, because we can always find BGV parameters, such that $2 \log n' \leq l$. The protocol, where a client wants to know the i-th bit of the server's n-bit database $\mathbf{x}$, then consists of the following 4 algorithms:

1. **Init(x):** The client generates the BGV key pair (pk, sk). Given $\phi(m)$ and the number of plaintext slots l, the server represents its n-bit database $\mathbf{x}$ as an $n' \times l$ matrix $\mathbf{X}$, where $n' = \lceil \frac{n}{\phi(m)} \rceil$. Each element $\mathbf{X}_i \in \mathbf{X}$ is a d-bit binary vector, where $d = \frac{\phi(m)}{l}$.

2. **QGen(i, n', l):** The client converts i into (α, β, γ), where $\mathbf{x}_i$ is the γ-th bit in the element at $\mathbf{X}_{\alpha\beta}$. The client creates a query string $\mathbf{q}$, with all bits set to 0, except the α-bit, which is set to 1. The client generates a folding tree with input n' and folds $\mathbf{q}$ into $\mathbf{s}$. They pad $\mathbf{s}$ with 0's to get a length of l and circularly shift $\mathbf{s}$ to the right to get $\mathbf{s}' = \mathbf{s} \gg (\beta - 1)$. Now, the β-th bit in $\mathbf{s}'$ is the first bit in $\mathbf{s}$. The client packs $\mathbf{s}'$ and encrypts it to get $\mathfrak{s} = \mathrm{E}_{pk}(\pi(\mathbf{s}'))$ and sends it to the server.

3. **RGen($\mathfrak{s}$):** The server creates a vector $\mathbf{c}$ of $2 \log n'$ ciphertexts where $\mathbf{c}_1 = \mathfrak{s}$ and for each $2 \leq k \leq 2 \log n'$, $\mathbf{c}_k = \mathfrak{s} \triangleleft (k-1)$. The server then generates a folding tree with input n' and homomorphically unfolds $\mathbf{c}$ into $\mathbf{c}'$. The vector $\mathbf{c}'$ contains n' ciphertexts. The server uses $\mathbf{c}'$ and the packed columns of the matrix $\mathbf{X}$ to homomorphically compute the inner product $\mathfrak{r} = (\mathbf{c}'_1 \boxtimes \pi(\mathbf{X}_1)) \boxplus (\mathbf{c}'_2 \boxtimes \pi(\mathbf{X}_2)) \boxplus \cdots \boxplus (\mathbf{c}'_{n'} \boxtimes \pi(\mathbf{X}_{n'}))$. The response $\mathfrak{r}$ is then sent to the client.

4. **RExt($\mathfrak{r}$):** Finally, the client decrypts $\mathfrak{r}$ and obtains a vector with l elements. The γ-th bit in the β-th element in the vector is the bit $\mathbf{x}_i$ they want to retrieve.

In Figure 5 we see a small example of DC-PIR. The parameters for the example are $n = 32$, $\phi(m) = 8$, $n' = 4$, $l = 4$, $d = 2$, $\alpha = 3$, $\beta = 2$, and $\gamma = 1$. The server's database is organized as a 4×4 matrix with 2-bit elements and the client wants to retrieve the first bit in the element at $\mathbf{X}_{3,2}$.

3 Related Work

In this section we summarize related research done on biometric verification and oblivious data structures.

3.1 Biometric verification systems

Trauring was the first to publish research on automated biometric recognition in 1963 with his article on fingerprint matching [17], closely followed by Pruzansky [18] on voice recognition. Afterwards, research on automated recognition of other biometric traits was published, such as for signature by Mauceri [19] and for face by Bledsoe [20]. Ernst [21] patented a system for hand geometry and in 1993 Daugman wrote a paper on a biometric system based on the iris [22]. Later research focused on finding new techniques and improving existing ones. Bazen and Veldhuis [5], for example, showed that for multi-user verification the use of the likelihood ratio is optimal in terms of average error rates. This was later used by Peeters et al. [3] to design their system, which focuses on high accuracy, high performance, and privacy protection. Later, Bassit et al. [23] extended this work to include an additional attacker model.

Biometric verification systems are vulnerable to a variety of attacks, for example, spoof attacks, linkability attacks, and identity creep. With a spoof attack an attacker uses a counterfeit biometric trait that is not obtained from a live person to fool the system [24]. An example of a spoofed biometric trait is a photograph of a person's face or a gummy finger. Research has been dedicated to developing liveness detection techniques to prevent spoofing as shown in Nixon et al. [25]. A linkability attack is when an attacker links users cross applications based on their biometric data [26]. Finally, identity creep occurs when an attacker makes repeated attempts to take on the identity of a legitimate user of the system and succeeds due to a false match [26].

Biometric template security is a critical step in minimizing the security and privacy risks associated with biometric systems, according to Jain and Nandakumar [24]. This is where oblivious data structures can be useful, since they focus on protecting access behavior and/or their users.

3.2 Oblivious data structures

Kushilevitz and Ostrovsky [14] first showed that single-server PIR schemes with sublinear communication complexity were possible. Later, various such schemes were

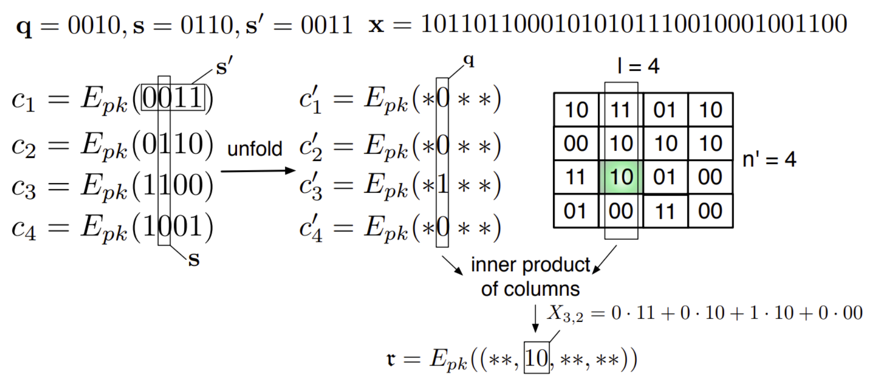


Figure 5: Example of DC-PIR (* denotes a bit we do not care about) (from [9])

proposed by e.g. Cachin et al. [27], Kushilevitz and Ostrovsky [28], Gentry and Ramzan [29], and Lipmaa [30]. Sion and Carbunar [15] found that nontrivial single-server PIR schemes like these may often have low communication complexity, but are slow due to high server-side computation. Some other solutions take this to heart, like the scheme we use in this research, Dong and Chen [9], and Corrigan-Gibbs and Kogan [31], which uses an offline/online model. This means they split the protocol in two phases. The linear-time server-side computation is performed in a query-independent offline phase, so the subsequent online phase can be completed in sublinear time. Next to PIR there is exists another oblivious data structure, namely ORAM.

ORAM was originally proposed by Goldreich and Ostrovsky [6] and it allows a user to store data on a remote server and access it without revealing the access pattern. This is done by arranging the data in such a way that the user never touches the same piece more than once. It was initially proposed for the client-server setting, where the data stored on the server is owned by a single client. The server does not need to perform any (heavy) computation, but the communication complexity becomes quite high. Additional schemes have been proposed that lower the communication complexity but increase the client memory, such as Path ORAM by Stefanov et al. [32].

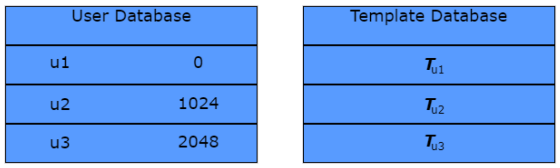
Gordon et al. [33] has adapted ORAM to create RAM-based secure computation (RAM-SC). It uses ORAM techniques to perform secure computation, in which two or more parties evaluate a function together using secret input from the parties. No party should learn anything about the data or access pattern, only the result of the computation is revealed. Zahur et al. [34] implemented a version, which uses Goldreich and Ostrovsky's original square-root ORAM scheme for secure multi-party computation.

4 Design

We identified two possible applications for PIR to biometric verification with the precomputed log-likelihood ratio classifier. First, we use PIR to retrieve the user's template from the database, effectively hiding the user's identity as described in section 4.1. Second, we retrieve the similarity scores of the user's probe and template combination with PIR to protect the template as described in section 4.2.

4.1 Design 1: Retrieving the full template

Applying PIR to retrieve the user's full template ensures their identity is kept secret. In Peeters et al.'s protocol (see section 2.3.2) the sensor device sends an identity claim u to the verification service to retrieve the corresponding template. By applying PIR in this step, the template can be retrieved without revealing the claim to the verification service. This is important in a biometric verification setting, since a user's identity can be linked to their biometric template [35].



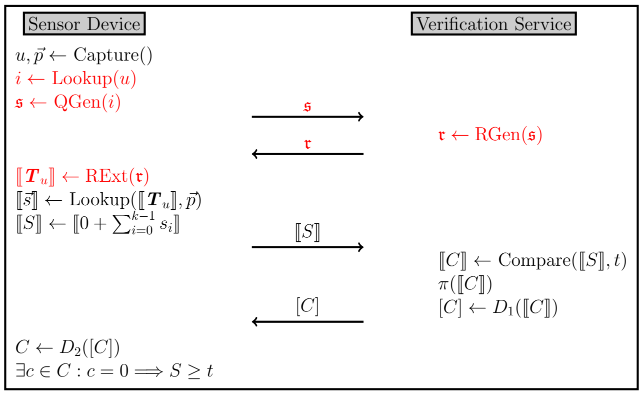
User Database	
u1	0
u2	1024
u3	2048

Template Database
T_{u1}
T_{u2}
T_{u3}

Figure 6: Design 1 uses a user database to store the location of each user's template in the template database. In this toy example each template is 1024 bits long.

First, we explain our design on an intuitive level; then, we give a more formal explanation. We start with the enrollment phase, which is the same as in Peeters et al.'s protocol (see section 2.3.2), where the sensor device generates a template $\boldsymbol{T}_u$, encrypts it, and sends it to the verification service, which stores it in the template database. Additionally, the verification service keeps a second, smaller database, which we call the user database as shown in Figure 6. It keeps track of the starting bit of each user's template in the template database and is thus used to determine which bits the sensor device must retrieve with PIR.

The verification phase starts with the sensor device capturing the user's biometric probe. They then access the verification service's user database to determine the index of the template's first bit. The user database can either be downloaded in its entirety or be accessed with PIR, depending on whether the focus lies on reducing computation or communication complexity. The user database is very small in comparison to the template database. The sensor device uses the template's index to generate the DC-PIR query and sends it to the verification service. The verification service then generates a response and returns it to the sensor device. Finally, the sensor device extracts the user's template from the response. The rest of the protocol is the same as Peeters et al.'s protocol discussed in section 2.3.2.


Sensor Device
Verification Service
$u, \vec{p} \leftarrow \text{Capture}()$
$i \leftarrow \text{Lookup}(u)$
$\mathfrak{s} \leftarrow \text{QGen}(i)$
$\mathfrak{s}$
$\mathfrak{r} \leftarrow \text{RGen}(\mathfrak{s})$
$\mathfrak{r}$
$[\![\boldsymbol{T}_u]\!] \leftarrow \text{RExt}(\mathfrak{r})$
$[\![\vec{s}]\!] \leftarrow \text{Lookup}([\![\boldsymbol{T}_u]\!], \vec{p})$
$[\![S]\!] \leftarrow [\![0 + \sum_{i=0}^{k-1} s_i]\!]$
$[\![S]\!]$
$[\![C]\!] \leftarrow \text{Compare}([\![S]\!], t)$
$\pi([\![C]\!])$
$[C] \leftarrow D_1([\![C]\!])$
$[C]$
$C \leftarrow D_2([C])$
$\exists c \in C : c = 0 \implies S \geq t$
Figure 7: Verification phase of design 1. The red text shows where this design differs from Peeters. et al's verification protocol (see Figure 2).

We now present the verification protocol more formally using the notation introduced in section 2.1. We begin the verification phase (Figure 7) with the sensor device calling *Capture()* to get the probe $\vec{p}$ and the identity claim u. The sensor device then calls *Lookup(u)* on the user database to determine the index i of the first bit in the template in the template database. The sensor device calls *QGen(i)* to get $\mathfrak{s}$, which they send to the verification service. The verification service calls *RGen($\mathfrak{s}$)* to get $\mathfrak{r}$, which they send back to the sensor device. The sensor device calls *RExit($\mathfrak{r}$)* to get the template $[\![\boldsymbol{T}_u]\!]$, finishing up DC-PIR. The rest of the protocol is the same as in Peeters et al.'s protocol (see section 2.3.2).

4.2 Design 2: Retrieving the similarity scores

We have also applied PIR to retrieving the similarity scores from the user's template directly. This method addresses the user's template being sent to the sensor device. There is no system in place to preserve integrity, so the homomorphic property of the encrypted template can be leveraged to attempt to generate synthetic biometric data to get a false acceptance. Therefore, in our approach, the verification service does not send the user's template to the sensor device. Instead, the template acts as the database for the PIR protocol, so each column corresponding to the user's probe can be retrieved without the verification service finding out which column was retrieved. We also use a double additive blinding instead of HE to keep the template private. The advantage of a blinding instead of HE is that the result is much smaller in size. Computational complexity in PIR scales with the database size and we use the template as database, hence blinding instead of HE significantly reduces the computational costs of the PIR scheme. The double blinding consists of a client blind applied during enrollment, and a server blind, which is different for each run of the verification. This prevents an attacker from running the protocol multiple times to get the entire template. The additive blinding also affects how the rest of the protocol is performed.

Again, we first explain our design on an intuitive level after which we give a more formal explanation. In the enrollment phase the sensor device generates a template and a client blind and adds them together. The blinded template is then sent to the verification service. Then in the verification phase the sensor captures the user's probe and generates the DC-PIR query with it. The result is sent to the verification service, which generates a server blind and adds it to the template. Afterwards, we use the template as database and generate a DC-PIR response to the sensor device's query. The response is sent to the sensor, where we apply the final DC-PIR step to get a blinded similarity score. We sum the score and add a random integer as additional blinding to the template blindings. The random integer is also added to the summed client blind and encrypted using an FHE scheme inherently used by DC-PIR, so no additional scheme is needed. Both the blinded similarity score and the encrypted blind are sent to the verification service. Here we do a threshold comparison under blinding, similar to the comparison under encryption in Peeters et al.'s protocol (see section

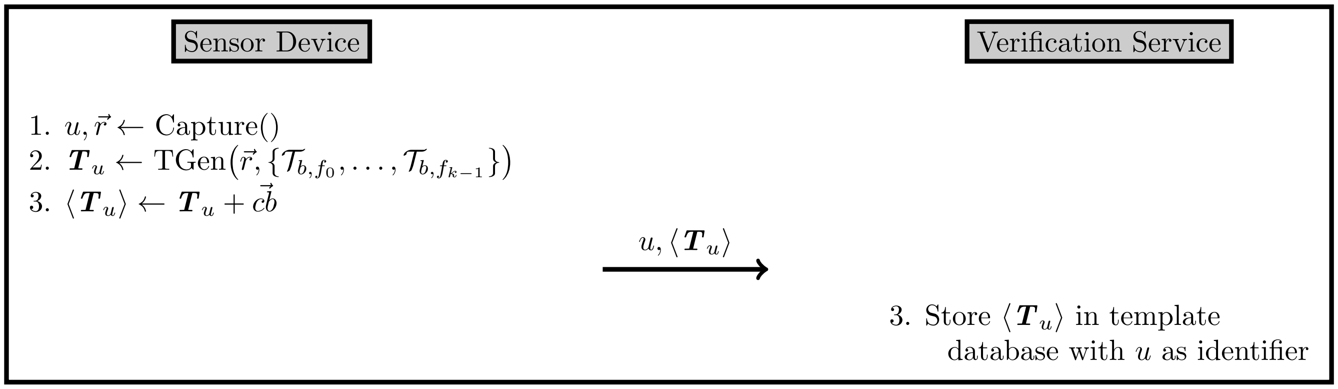

Sensor Device
Verification Service
1. $u, \vec{r} \leftarrow$ Capture()
2. $\boldsymbol{T}_u \leftarrow \mathrm{TGen}\big(\vec{r}, \{\mathcal{T}_{b,f_0}, \dots, \mathcal{T}_{b,f_{k-1}}\}\big)$
3. $\langle \boldsymbol{T}_u \rangle \leftarrow \boldsymbol{T}_u + \vec{cb}$
$u, \langle \boldsymbol{T}_u \rangle$
3. Store $\langle \boldsymbol{T}_u \rangle$ in template database with u as identifier

Figure 8: Enrollment phase of design 2

2.3.2). During the comparison a multiplicative blind vector is applied to the result vector as well. We then homomorphically add the summed server blind to the encrypted client blind and homomorphically multiply it with the multiplicative blind vector. We get a vector of encrypted blindings, each corresponding to a value in the result vector. We apply the same permutation to both the encrypted blind vector and result vector. Both are then sent to the sensor device. The sensor device decrypts the blind vector and subtracts it from the result vector. If the result vector then contains a zero, the verification is accepted, otherwise it is rejected.

We now present the protocol more formally using the notation introduced in section 2.1. An overview of the enrollment protocol is depicted in Figure 8 and of the verification protocol in Figure 9. The enrollment phase starts by generating a template $\boldsymbol{T}_u$. However, instead of encrypting it with ElGamal, we now generate a blinding for the template. For each similarity score in the template we generate a random integer and add it to the score to get the single blinded template $\langle \boldsymbol{T}_u \rangle$ (we use single angle brackets to denote a single blinded value $\langle \cdot \rangle$ and double angle brackets for a double blinded value $\langle\!\langle \cdot \rangle\!\rangle$). We gather the random integers together to form the client blind vector $\vec{cb}$, which we can sum together to get the client blind $CB = \mathrm{sum}(\vec{cb})$. The template is then sent to the verification service along with the identity u, where they are both stored in the template database.

In the verification phase (Figure 9) we start with the sensor calling *Capture()* to obtain u and $\vec{p}$. The sensor device then begins DC-PIR by calling *QGen($\vec{p}$)* to produce $\mathfrak{s}$, which is sent to the verification service along with u. The verification service then generates a server blind vector $\vec{sb}$, similar to the client blind vector $\vec{cb}$ in the enrollment phase. However, for every row in the template a random integer value is generated that is used for each similarity score in that row. So $\vec{sb}$ consists of k (= number of features) random integers, where $\vec{cb}$ consists of $k \cdot 2^b$ random integers. We then add $SB = sum(\vec{sb})$ to the template to produce a doubly blinded template $\langle\!\langle \boldsymbol{T}_u \rangle\!\rangle$, which we use as the database for *RGen($\mathfrak{s}$)* to produce $\mathfrak{r}$, which is sent back to the sensor device. The sensor device now finishes the PIR protocol by calling *RExt($\mathfrak{r}$)*, obtaining $\langle\!\langle \vec{s} \rangle\!\rangle$.

The sensor device now sums $\langle\!\langle \vec{s} \rangle\!\rangle$ and adds a random integer r to get the blinded value $\langle\!\langle S \rangle\!\rangle$ (r ensures the verification service cannot guess which similarity scores were accessed from the template). This value now consists of the summed similarity scores, the summed client blind, the summed server blind, and the random integer r; $\langle\!\langle S \rangle\!\rangle = S + CB + SB + r$. The sensor device then adds r to CB and encrypts it using a FHE scheme $|CB| \leftarrow Enc(CB + r)$. The sensor device then sends it to the verification service, along with $\langle\!\langle S \rangle\!\rangle$.

To check the similarity score against the threshold t, we need a vector $\vec{\theta}$ that contains every integer between t and the maximum value the summed similarity score can be, which we call $\max(\mathbb{S})$; $\vec{\theta} = \{t, t{+}1, \dots, \max(\mathbb{S})\}$, with order ℓ. We also generate a vector of random, non-zero integers $\vec{a}$ of order ℓ as additional, multiplicative blinds. Then, we compute a vector $\vec{c}$, where for each $i \in [0, \ell - 1]$ we have $\langle\!\langle c_i \rangle\!\rangle \leftarrow a_i\,(\langle\!\langle S \rangle\!\rangle - \theta_i)$. The verification service then homomorphically adds the server blind SB to the encrypted client blind to get $|CB + SB| \leftarrow |CB| \boxplus SB$. We then apply a permutation π to $\vec{a}$ and $\langle\!\langle \vec{c} \rangle\!\rangle$, to avoid leaking information on the similarity score. After applying the permutation we homomorphically multiply $\vec{a}$ with $|CB + SB|$ and send the result to the sensor device, along with $\langle\!\langle \vec{c} \rangle\!\rangle$.

The sensor device decrypts the encrypted blinds to get $\vec{a}(CB + SB)$, which we subtract from $\langle\!\langle \vec{c} \rangle\!\rangle$ to get the result set $\vec{c}$. If there exists a 0 in the result set, we can conclude that $S \geq t$, otherwise $S < t$.

Security: Intuitively, the verification protocol is secure, since the communicated data is blinded with CB and SB. Neither one of the two parties can undo all

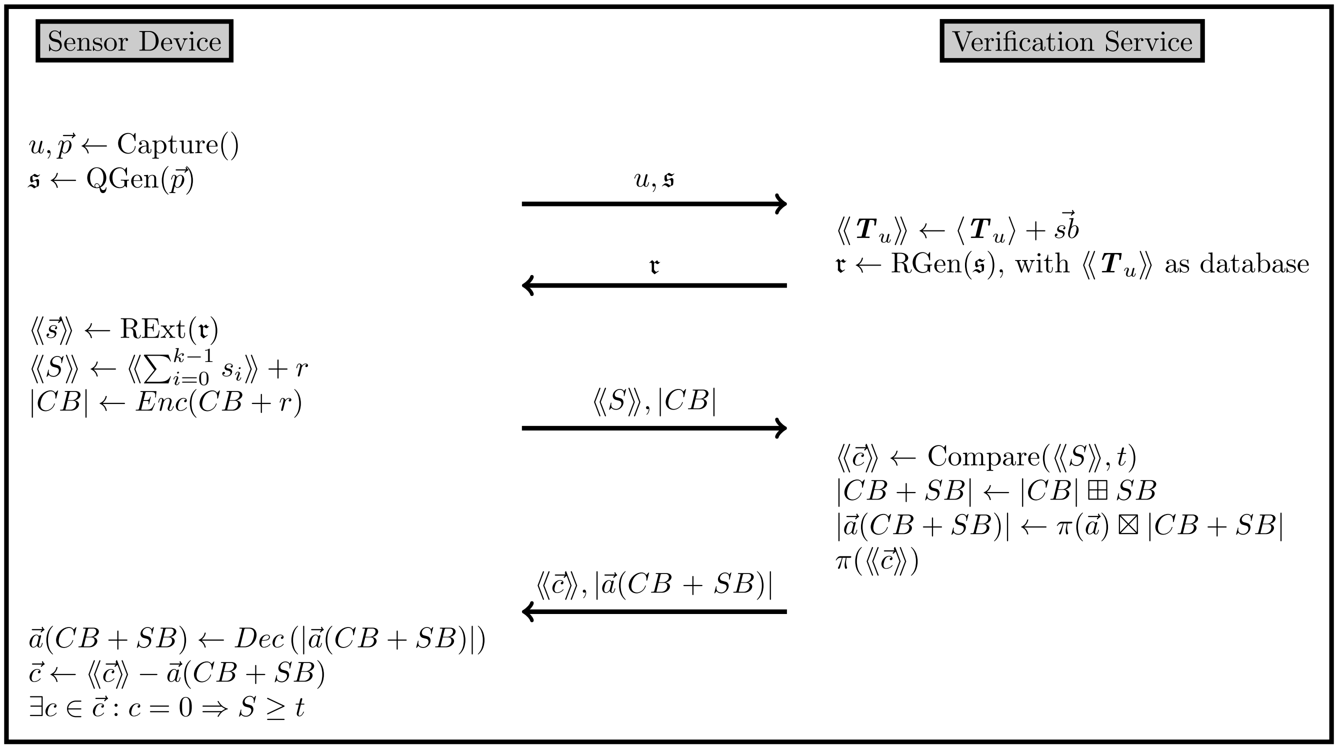

Sensor Device
Verification Service
$u, \vec{p} \leftarrow \text{Capture}()$
$\mathfrak{s} \leftarrow \text{QGen}(\vec{p})$
$u, \mathfrak{s}$
$\langle\!\langle \boldsymbol{T}_u \rangle\!\rangle \leftarrow \langle \boldsymbol{T}_u \rangle + \vec{sb}$
$\mathfrak{r} \leftarrow \text{RGen}(\mathfrak{s})$, with $\langle\!\langle \boldsymbol{T}_u \rangle\!\rangle$ as database
$\mathfrak{r}$
$\langle\!\langle \vec{s} \rangle\!\rangle \leftarrow \text{RExt}(\mathfrak{r})$
$\langle\!\langle S \rangle\!\rangle \leftarrow \langle\!\langle \sum_{i=0}^{k-1} s_i \rangle\!\rangle + r$
$|CB| \leftarrow Enc(CB + r)$
$\langle\!\langle S \rangle\!\rangle, |CB|$
$\langle\!\langle \vec{c} \rangle\!\rangle \leftarrow \text{Compare}(\langle\!\langle S \rangle\!\rangle, t)$
$|CB + SB| \leftarrow |CB| \boxplus SB$
$|\vec{a}(CB + SB)| \leftarrow \pi(\vec{a}) \boxtimes |CB + SB|$
$\pi(\langle\!\langle \vec{c} \rangle\!\rangle)$
$\langle\!\langle \vec{c} \rangle\!\rangle, |\vec{a}(CB + SB)|$
$\vec{a}(CB + SB) \leftarrow Dec\left(|\vec{a}(CB + SB)|\right)$
$\vec{c} \leftarrow \langle\!\langle \vec{c} \rangle\!\rangle - \vec{a}(CB + SB)$
$\exists c \in \vec{c} : c = 0 \Rightarrow S \geq t$

Figure 9: Verification phase of design 2

blindings on the data single-handedly. The blindings themselves are encrypted by the sensor device using BGV, so the verification service cannot use them to undo the blindings on $\langle\!\langle S \rangle\!\rangle$. In addition, a random permutation π hides the index of the comparison result in $\langle\!\langle \vec{c} \rangle\!\rangle$ and $\vec{a}$ randomizes all its non-zero values, so the sensor device cannot determine the value of S.

Correctness: For each $i \in [0, \ell - 1]$ we can write $\langle\!\langle \vec{c} \rangle\!\rangle$ as $\langle\!\langle c_i \rangle\!\rangle = a_i(S + SB + CB - \theta_i) = a_i(S - \theta_i) + a_i(CB + SB)$, if we then subtract $a_i(CB + SB)$ we are left with $a_i(S - \theta_i)$. If $a_i(S - \theta_i) = 0$, then $S - \theta_i = 0$, which means that $S \in \vec{\theta}$, and therefore $S \geq t$.

5 Experiments and Results

We implemented the designs and conducted all experiments on a desktop with an Intel Core i5-8400 2.8 GHz CPU and 16 GB RAM running Ubuntu 21.04. We first implemented DC-PIR to test its computational and communication complexity. Afterwards, we implemented design 2, retrieving similarity scores directly from the template.

5.1 Implementation

We implemented our designs on C++ using the open source lattice crypto software library Palisade[1], as it provides an API for BGV. Palisade uses the value of the ring dimension $\phi(m)$ for the plaintext vector space l. In DC-PIR these values are separate, because if $\phi(m)$ and l are the same, the matrix element size $d = \frac{\phi(m)}{l}$ becomes 1. This greatly reduces the amount of bits that can be retrieved from the database at once. Luckily, we can artificially use a different l as DC-PIR parameter, which we call l', and fit multiple DC-PIR queries in a single ciphertext, similar to how Dong and Chen use the SIMD property in the original protocol. We create multiple queries of size l' and pack them into a single ciphertext. The queries do not have to be sequential.

We incorporated optimizations suggested by DC-PIR as well. First, we have tree pruning, where we prune the compression tree, which decreases the multiplication depth of BGV and therefore reduces communication cost. Second, we use multithreading during unfolding and calculating the inner product, which greatly reduces server-side computation time.

[1]`https://palisade-crypto.org`

Database Size → **# Bits ↓**	$\mathbf{2^{18}}$	$\mathbf{2^{19}}$	$\mathbf{2^{20}}$	$\mathbf{2^{21}}$	$\mathbf{2^{22}}$	$\mathbf{2^{23}}$	$\mathbf{2^{24}}$	$\mathbf{2^{25}}$
128	*0,103*	*0,1892*	*0,3817*	*0,735*	*1,4557*	*1,7973*	*3,3093*	*6,4698*
256	0,206	0,3784	0,7634	1,47	2,9114	3,5946	6,6186	12,9396
640	0,515	0,946	1,9085	3,675	7,2785	8,9865	16,5465	32,349
1024	0,824	1,5136	3,0536	5,88	11,6456	14,3784	26,4744	51,7584
3456 (BMDB)	2,781	5,1084	10,3059	19,845	39,3039	48,5271	89,3511	174,6846
13568 (FRGC)	10,918	20,0552	40,4602	77,91	154,3042	190,5138	350,7858	685,7988
18816 (PUT)	15,141	27,8124	56,1099	108,045	213,9879	264,2031	486,4671	951,0606

Table 1: Time (in seconds) needed to retrieve a number of bits from a database of a certain size. All rows from the second row onward are extrapolated from the values of the first row.

5.2 Results

5.2.1 DC-PIR

We measured the average time it takes to complete a single run of DC-PIR where we retrieve 128 bits at once for different database sizes. We used $\phi(m) = 8192$, $l' = 512$, and $d = 16$. We extrapolated the time it would take to retrieve more bits, in multiples of 128. The results are shown in table 1. The last three rows represent template sizes of three biometric datasets. These are BMDB [36] of 36 features and a lookup table of 16×16, FRGC [37] with 46 features and a lookup table of 49×49, and PUT [38] with 49 features and a lookup table of 64×64. Each value consists of 6 bits, so the final template sizes are 3456 bits for BMDB, 13524 bits for FRGC, and 18816 bits for PUT.

We also measured the size of the communicated data for a run of DC-PIR with a database of a certain size. We used Palisade's serialization feature to output the query and response in separate files and measured their sizes in bytes. The results are shown in table 2. They are mostly the same size, except for the response for the larger three database sizes, which might be due to the way Palisade serializes its ciphertexts.

Database size	Query	Response
2^{20} (1 MB)	387 KB	387 KB
2^{21} (2 MB)	387 KB	387 KB
2^{22} (4 MB)	387 KB	387 KB
2^{23} (8 MB)	387 KB	264 KB
2^{24} (16 MB)	387 KB	264 KB
2^{25} (32 MB)	387 KB	264 KB

Table 2: Communication cost for different database sizes (in bits)

5.2.2 Second design

For the second design we did three tests using the template sizes from the biometric databases mentioned in section 5.2.1. We used different DC-PIR parameters for each template size to optimize runtime. We used 8 bits per template element and $d = 8$ as the base for the other DC-PIR parameters as well. The parameters and runtime are shown in table 3. DC-PIR runs make up most of the runtime with the rest of the protocol taking less than 100 milliseconds for each database type. Taking $l' = k$ is in general a good starting point for the DC-PIR parameters. However, in some cases it does not work for practical reasons, such as with FRGC, where $l' \neq k$.

6 Discussion

In this section we will interpret and discuss the results from section 5. We will start with design 1 and then move on to design 2. We also present a short discussion on future research at the end of this section.

6.1 Interpretation of results

6.1.1 Design 1

First, we will discuss design 1, DC-PIR applied to retrieving the full template. From table 1 we can see that DC-PIR is slow when used with large databases. Even with the BMDB database, which has a template size of 3456 bits, it takes almost 3 seconds to retrieve template from a database of 2^{18} bits ($\sim$ 75 templates). Doubling the database size approximately doubles the time to retrieve a template as well. If we then look at databases such as FRGC and PUT, which have larger templates, the amount of runs necessary to retrieve a single template also increases. This leads to even longer retrieval times, making design 1 for large databases infeasible in most cases.

Even databases that use small templates of 256 bits become slow when they contain a large amount of templates. From table 1 we see it takes almost 3 seconds to retrieve a 256-bit template from a database of size 2^{22} bits, which holds $2^{14} = 16384$ entries. This shows that design 1 is not yet fast enough to be used in

Database	Template size (bits)	k	$\phi(m)$	n'	l'	Time per DC-PIR run	# DC-PIR runs	Total time
PUT	25088	49	392	64	49	190,308 ms	13	2,574 s
FRGC	18816	46	294	64	36	186,769 ms	12	2,341 s
BMDB	4608	36	288	16	36	64,08 ms	9	0,677 s

Table 3: Time needed to perform a run of design 2 per database type. In this table we also show the following parameters, k as the number of features, $\phi(m)$ as the ring dimension, and n' and l' as the number of rows and columns in DC-PIR's database matrix, respectively.

use cases that require a large amount of entries (more than 10000). However, in use cases that require smaller amounts of entries it reaches more acceptable speeds. An example of such a use case is a door in a building that should only be accessible to authorized personnel. The number of authorized personnel should be relatively low (less than 100).

Communication cost is mostly the same for different database sizes, as shown in table 2. This is because each run uses the same ciphertext size, regardless of database size. The ciphertext size depends on the BGV parameters, which we keep the same for the different database sizes in this test. The bottom three responses are smaller in size than the rest, we are unsure why this is the case. We expected it to be the same as the other response files, since they also consist of a single ciphertext. We believe it might have something to do with the way Palisade serializes ciphertexts to file, but we cannot say for certain.

6.1.2 Design 2

Second, we discuss design 2, DC-PIR applied to retrieving the similarity scores from the template. In this design we used the PUT, FRGC, and BMDB database template sizes again for testing. In table 3 we can see that most of the runtime is taken up by DC-PIR runs, which need to be run multiple times to retrieve every similarity score from the template. The amount of runs depend on the amount of features in the template. The PUT database consists of 49 features, FRGC consists of 46, and BMDB consists of 36. This translates to the number of DC-PIR runs in the table. We know that the runtime of the DC-PIR runs depends on the size of the database, which is the template itself in this case. PUT and FRGC have template sizes of the same order (2^{15}), and so their runtime per DC-PIR run is very similar. These factors translate to the PUT and FRGC database tests having very similar total time, 2,574 and 2,341 seconds, respectively.

On the other hand, BMDB is much faster, since its template size is considerably smaller and it consists of less features. The full run takes less than a second to complete and its runtime is not dependant on how many templates are stored in the server's database, so it is scalable. This shows that design 2 is promising when combined with moderately sized templates. Design 2 relies on the template not being encrypted, but rather it uses an additive blind to protect the template's content. An additive blind does not inflate the template size (or does so minimally), so DC-PIR's runtime remains within acceptable ranges.

The biometric performance of design 2 remains the same as the biometric verification system it is based on. The design does not change the results, since the way the underlying biometric comparison is performed is not altered. The design only changes the way the (intermediate) results are protected and communicated.

6.2 Future research

As mentioned in the previous section, design 2 uses an additive blind to ensure the template sizes remain small. However, there is no formal proof of the security of this method, which is necessary to further flesh out the design. This is therefore a great place to start future research. Alternatively, instead of using blinding future research could look at the use of an HE scheme with a small ciphertext space, so it does not (significantly) slow down DC-PIR in comparison.

Research could also be focused on finding new techniques to obtain smaller template sizes, as this would decrease DC-PIR runtime in both designs. This could possibly be accomplished by using less features in a template. This has the added benefit that less features corresponds to less DC-PIR runs in design 2, decreasing its overall runtime. We did not look at the relative effect on design 2's runtime of using smaller templates versus using less features. It would be interesting to see which of the two is more beneficial for the overall runtime.

A final, obvious topic for future research is faster, single-server PIR schemes. The biggest bottleneck in both designs is DC-PIR, so finding a PIR scheme that has a notable runtime improvement would have great effect. In this paper we focused on single-server PIR, but future research could also look into the use of multi-server PIR schemes to see whether the pros outweigh the cons.

7 Conclusion

In this paper we looked at two methods of applying PIR to biometric verification. We first sought to hide the user's identity claim from the server by retrieving the user's biometric template with DC-PIR in design 1. We then used DC-PIR to retrieve only the relevant similarity scores from the user's template directly in design 2, never revealing the template itself. We found that design 1 is very slow with large databases, especially when the template itself is also large, since it requires multiple runs of DC-PIR. Smaller databases, especially with smaller templates as well, have much better runtimes and could be used in certain situations. Design 2 shows more promise with moderately sized templates, achieving a runtime of less than a second for the BMDB database test. However, design 2 is not entirely fleshed out yet, since the blind to keep the template secret has not been formally proven secure yet.

We have shown that PIR can be applied to biometric verification to some extent. Use cases with small databases and small template sizes can already benefit from this combination. Systems with moderately to large sized templates still need more research before they can be used in practice.

8 Acknowledgements

I would like to thank Florian Hahn for helping me shape this research paper and giving me direction when I was unsure of where to go next, his guidance taught me a lot about scientific research. I would also like to thank Amina Bassit for her insight in all matters biometric and her assistance with coming up with design 2, in particular. Their feedback in general was a great boon to the quality of this work. Lastly, I would like to show my gratitude to Raymond Veldhuis for taking the time to assess my work and galloping through the examination procedure, so I could finish my thesis in a timely manner.

References

[1] E. Mordini. *Biometrics, Human Body, and Medicine: A Controversial History*, pages 249–272. 01 2009.

[2] A.K. Jain, K. Nandakumar, and A. Nagar. Biometric template security. *EURASIP J. Adv. Signal Process*, 2008, January 2008.

[3] J.J. Peeters, A. Peter, and R.N.J. Veldhuis. Fast and accurate likelihood ratio based biometric verification in the encrypted domain, August 2016.

[4] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G.R. Blakley and D. Chaum, editors, *Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[5] A.M. Bazen and R.N.J. Veldhuis. Likelihood ratio-based biometric verification. *IEEE transactions on circuits and systems for video technology*, 14(1):86–94, January 2004. Imported from DIES.

[6] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[7] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, page 41, USA, 1995. IEEE Computer Society.

[8] B. Chor, E. Kushilevitz, and O. Goldreich. Private information retrieval. *J. ACM*, 45:965–981, 11 1998.

[9] C. Dong and L. Chen. A fast single server private information retrieval protocol with low communication cost. In M. Kutyłowski and J. Vaidya, editors, *Computer Security - ESORICS 2014*, pages 380–399, Cham, 2014. Springer International Publishing.

[10] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[12] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[13] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 307–315, New York, NY, 1990. Springer New York.

[14] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[15] R. Sion and B. Carbunar. On the computational practicality of private information retrieval. 01 2007.

[16] N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. In *Designs, Codes and Cryptography*, volume 71, pages 57–81, 2014.

[17] M. Trauring. Automatic comparison of finger-ridge patterns. volume 197, pages 938–940, 1963.

[18] S. Pruzansky. Pattern-matching procedure for automatic talker recognition. *The Journal of the Acoustical Society of America*, 35:354, 1963.

[19] A. J. Mauceri. Feasibility study of personnel identification by signature verification. Technical report, NORTH AMERICAN AVIATION INC DOWNEY CALIF SPACE AND INFORMATION SYSTEMS DIV, 1965.

[20] W. W. Bledsoe. Man-machine facial recognition. *Panoramic Research Inc., Palo Alto, CA*, 1966.

[21] R. H. Ernst. Hand id system, U.S. Patent 3576537, 1971.

[22] J. G Daugman. High confidence visual recognition of persons by a test of statistical independence. *IEEE transactions on pattern analysis and machine intelligence*, 15(11):1148–1161, 1993.

[23] A. Bassit, F. W. Hahn, J. J. Peeters, T. Kevenaar, R. N. J. Veldhuis, and A. Peter. Fast and accurate likelihood ratio based biometric verification secure against malicious adversaries. *IEEE transactions on information forensics and security*, 16:5045–5060, October 2021. Publisher Copyright: Author.

[24] A. K. Jain and K. Nandakumar. Biometric authentication: System security and user privacy. *Computer*, 45(11):87–92, 2012.

[25] K. A. Nixon, V. Aimale, and R. K. Rowe. Spoof detection schemes. 2008.

[26] A. K. Jain, K. Nandakumar, and A. Ross. 50 years of biometric research: Accomplishments, challenges, and opportunities. volume 79, pages 80–105, 2016.

[27] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, page 402–414, Berlin, Heidelberg, 1999. Springer-Verlag.

[28] E. Kushilevitz and R. Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In B. Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000. EUROCRYPT 2000. Lecture Notes in Computer Science*, volume 1807, pages 104–121. Springer Berlin Heidelberg, 2000.

[29] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming. ICALP 2005. Lecture Notes in Computer Science*, volume 3580, pages 803–815. Springer Berlin Heidelberg, 2005.

[30] H. Lipmaa. An oblivious transfer protocol with log-squared communication. In J. Zhou, J. Lopez, R.H. Deng, and F. Bao, editors, *Information Security. ISC 2005. Lecture Notes in Computer Science*, volume 3650, pages 314–328. Springer Berlin Heidelberg, 2005.

[31] H. Corrigan-Gibbs and D. Kogan. Private information retrieval with sublinear online time. In *39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings*, volume 12105 of *Lecture Notes in Computer Science*. Springer, 2020.

[32] E. Stefanov, M. Van Dijk, E. Shi, T.-H. Hubert Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: An extremely simple oblivious ram protocol. *J. ACM*, 65(4), April 2018.

[33] S.D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis. Secure two-party computation in sublinear (amortized) time. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 513–524, New York, NY, USA, 2012. Association for Computing Machinery.

[34] S. Zahur, X. Wang, M. Raykova, A. Gascón, J. Doerner, D. Evans, and J. Katz. Revisiting square-root oram: Efficient random access in multi-party computation. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 218–234, 2016.

[35] Q. Tang, J. Bringer, H. Chabanne, and D. Pointcheval. A formal study of the privacy concerns in biometric-based remote authentication schemes. In L. Chen, Y. Mu, and W. Susilo, editors, *Information Security Practice and Experience*, pages 56–70, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[36] Javier Ortega-Garcia, Julian Fierrez, Fernando Alonso-Fernandez, Javier Galbally, Manuel R. Freire, Joaquin Gonzalez-Rodriguez, Carmen Garcia-Mateo, Jose-Luis Alba-Castro, Elisardo Gonzalez-Agulla, Enrique Otero-Muras, Sonia Garcia-Salicetti, Lorene Allano, Bao Ly-Van, Bernadette Dorizzi, Josef Kittler, Thirimachos Bourlai, Norman Poh, Farzin Deravi, Ming N. R. Ng, Michael Fairhurst, Jean Hennebert, Andreas Humm, Massimo Tistarelli, Linda Brodo, Jonas Richiardi, Andrezj Drygajlo, Harald Ganster, Federico M. Sukno, Sri-Kaushik Pavani, Alejandro Frangi, Lale Akarun, and Arman Savran. The multiscenario multienvironment biosecure multimodal database (bmdb). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(6):1097–1111, 2010.

[37] P.J. Phillips, P.J. Flynn, T. Scruggs, K.W. Bowyer, Jin Chang, K. Hoffman, J. Marques, Jaesik Min, and W. Worek. Overview of the face recognition grand challenge. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 947–954 vol. 1, 2005.

[38] A. Kasiński, A. Florek, and A. Schmidt. The put face database. *Image Processing and Communications*, Vol. 13, no 3-4:59–64, 2008.