# On the Use of Ordered Biometric Features for Secure Key Agreement

Dilara Akdoğan*, Duygu Karaoğlan Altop* and Albert Levi*
*Sabancı University, Istanbul, TURKEY
{dakdogan, duyguk, levi}@sabanciuniv.edu

***Abstract*—In this work, we propose a novel secure key agreement protocol, Secure Key Agreement using Pure Ordered Biometrics (SKA-POB), in which the cryptographic keys are generated using an ordered set of biometrics, without any other helper data. The proposed approach is realized using iris biometrics. Our protocol makes use of hash functions, and we propose a window-based comparison strategy and a window reset method. This way, performance is maximized without sacrificing security. SKA-POB protocol works in round manner, allowing to successfully terminate with key establishment as early as possible so that the complexity is reduced for both client and server sides. Additionally, we employ multi-criteria analyses for our proposed SKA-POB protocol and we provide verification results in terms of performance analysis together with randomness, distinctiveness and attack complexity through security analysis. Results show that highly random and secure keys can be generated with almost no error and with very low complexity.**

## I. Introduction

Biometrics and cryptography are combined such that a cryptographic key is strictly bound to the users' biometric traits, hence the users' identities. In the literature, there are many examples in which the biometrics are used for cryptographic key generations, which are analysed by [1], [2] and [3] in-depth. One of the important approaches on using biometric data in cryptographic protocols is to generate the cryptographic key directly from it. Biometric data can be categorized in two categories as ordered and unordered. Unordered data, such as fingerprints, can be combined in any order when used as cryptographic keys, while ordered data, such as iris, must be used in a specific order. This feature plays an important role in the design of the key agreement protocol to be developed in order to maintain both authentication performance and key quality [4]. In this work, we propose a secure key agreement protocol, named SKA-POB (Secure Key Agreement using Pure Ordered Biometrics), in which the client and the server agree on a symmetric cryptographic key generated directly from the captured ordered biometrics. No ancillary secret information other than the biometric data is used in the protocol during the key generation process. The protocol itself is based on one of our previous works [5], which is a secure key agreement protocol that uses biometrics with unordered set of features. However, due to the nature of ordered set of biometrics, in SKA-POB, we employed some extra features for both performance and security improvement. We introduced *window-based comparison* and *window reset* methodologies in order to improve the performance of the protocol without reducing its security and key quality. Besides, chaff blocks are introduced to increase the resistance against brute force, replay and impersonation attacks. These chaff blocks are generated such that they are indistinguishable from the original iris blocks. Our protocol works in rounds and in each round, a similarity score is calculated using the number of blocks found in common, according to which the user is either accepted or rejected. In this way, we allow SKA-POB to successfully terminate as early as possible.

Iris biometrics are used in applications requiring high security, with high distinctiveness, high permanence, high validation and low error rates, despite some difficulties in use [2]. As a proof of concept, we instantiate SKA-POB using iris, which is an ordered biometric. Our performance evaluation results show that Incorrect Key Generation Rate (IKGR) is $0\%$. Besides, we show that our protocol can withstand a high attack complexity ($2^{148}$) even with a cleverly constructed brute force attack. On the other hand, randomness of the generated keys is measured using entropy and the keys are found to be random at high ratios. Additionally, distinctiveness of the generated keys is evaluated by the proposed *key-block difference index* metric. In this respect, it is shown that on the average $74.61\%$ of the keys are generated by using different blocks. Given the complexity results, our protocol is computationally viable with today's technology.

The rest of this paper is organized as follows. In Section II, we introduce our proposed secure key agreement protocol (SKA-POB). Section III evaluates the performance of our proposed protocol and discusses its security analyses. Finally, in Section IV, we conclude the paper.

## II. Secure Key Agreement using Pure Ordered Biometrics (SKA-POB)

In this section, we describe our proposed SKA-POB protocol. SKA-POB uses iris biometrics and generates secure cryptographic keys directly from the iriscodes. Our key agreement protocol is divided into two phases: (i) enrollment, and (ii) verification. In the below subsections, each of these phases are explained in detail.

This work was supported in part by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under grant 114E557.
Dilara Akdoğan was supported by TÜBİTAK BİDEB 2228-A.
Duygu Karaoğlan Altop was supported by TÜBİTAK BİDEB 2211-C and Turkcell Academy Technology Leaders Graduate Scholarship Program.

### A. Enrollment Phase

At the enrollment stage, the user provides three different iris scans of the same eye. Then, from these iris scans, corresponding iriscodes are extracted. Each iris-code is 9600 bits long. Using these three iriscodes, most reliable bit values are selected. If a particular bit on a specific position has the same value in two out of three iriscodes, this value is taken as a reliable value. By picking the reliable values in each position of three iriscodes, the final 9600 bit iriscode is obtained. An iriscode is processed as blocks in our algorithm. Thus, each iriscode is divided into equal size blocks during enrollment. After some experiments, we decided to divide each iriscode into 25 blocks, so that each block is 384 bits long. These blocks are stored in the server.

### B. Verification Phase

In the verification stage, three different iris scans of the same eye are provided by the user. Iriscodes are extracted and most reliable bits are selected, as in the enrollment stage, which yields a 9600 bit iriscode divided into 25 blocks. For hiding the genuine blocks, fake blocks are generated randomly at the user side. In order to distribute fake blocks uniformly, a fake block generation constant, $X$, is defined. Before and after each genuine block, a uniformly distributed random number is generated in the interval $[0 \,..\, X]$, say $x$. After that, $x$ different random blocks are put in that place. As a result, uniformly distributed, approximately $|G_u| \times \frac{X}{2}$ fake blocks are inserted among the genuine blocks. These points, together with the user id, are sent to the server.

As shown in Figure 1, firstly genuine and fake blocks' list is transmitted to the server. The server, then, compares each received block with the users' stored blocks. Comparison is done using a *window-based strategy*. The server knows that there is at least zero and at most $X$ fake blocks between each genuine blocks. It means that the first genuine block can be one of the first $(X+1)$ elements. Besides, second genuine block can be among the $2^{nd}$ and $(2X+2)^{th}$ elements. In short, each element can be in a specific window. The server compares the user's genuine block with each block in the window. If the percentage of difference between compared blocks is below the predefined acceptance threshold, the block is considered as genuine. In addition, if a genuine block is found in a position, the next block should be in subsequent positions, since the iriscode is ordered.

Apart from the window-based strategy, we also describe a *window reset strategy*. The server can make sure that a block is definitely genuine, if the similarity score is high enough. For this reason, we define a *reset threshold*, $T_{reset}$. If the difference between two compared blocks is below $T_{reset}$, it is considered as a genuine block definitely, and the window is reset. In other words, window is reinitialized. Assume that we are searching for the $i^{th}$ genuine block. If we do not reset the window, $i^{th}$ block can be one of the blocks in the interval $[i \,..\, i \times (X+1)]$. If we definitely find the $i^{th}$ block in the $k^{th}$ index of the list sent by the user, we reset the window. If we reset the window on the $r^{th}$ genuine block, we find the genuine block on the $k^{th}$ index, and we are searching for the $i^{th}$ genuine block, then new window would be $[k+(i-r) \,..\, k+(i-r) \times (X+1)]$. For instance, if the first block is definitely on the $5^{th}$ index ($r:1$, $i:2$, $k:5$), the second block can be between $6^{th}$ and $(X+6)^{th}$ indexes. In other words, we do not need to search for the second element until the $(2X+2)^{th}$ element.

After finding the genuine blocks, a similarity score is calculated [6]. If the similarity score is above the predefined acceptance threshold, $T_{sim}$, the protocol continues with the key generation phase. Otherwise, the user is rejected and the protocol ends. In the key generation phase, the server concatenates all common found blocks sent by the user and hashes the final concatenation. The hash result is the key, $K_{su}$, to be used in the communication. The server calculates an HMAC value on a predefined message using $K_{su}$ as the key. After that, the server sends the HMAC value and the number of common blocks used in the key generation, $|G_s^{'}|$, to the user.

At the user side, every possible candidate key is generated using $|G_s^{'}|$ many genuine blocks. If the HMAC value is verified for a candidate key, user stops and sends a positive acknowledgment. If the HMAC value cannot be verified using any of these keys, user sends a *RETRY* message.

If the server receives a *RETRY* message, a new similarity score is calculated with one less of the common block number, $|G_s^{'}|-1$. If this score is above the acceptance threshold, $T_{sim}$, the server generates every possible key using $|G_s^{'}|-1$ common blocks. An HMAC value is generated using each key, and all HMAC values are transmitted to the user. The user follows the same steps using the same number of genuine blocks as the server. If the user can verify any one of the HMAC values, the user sends a positive acknowledgment and the index, *i*, of the verified HMAC value. However, if none of the HMAC values can be verified, user again sends a *RETRY* message. In this case, the server applies the same process with $|G_s^{'}|-2$ number of genuine blocks. The protocol continues with the same steps in a round manner, until the calculated similarity score is less than the acceptance threshold, or any HMAC value is verified by the user. If any HMAC value is verified, it means that the server and the user agree on a symmetric cryptographic key. In contrast, if the protocol stops without generating a symmetric key, it can start from scratch upon request with new client side biometrics.

## III. Performance Evaluation of SKA-POB Protocol

Our proposed SKA-POB protocol was evaluated with 70 subjects from public CASIA-IrisV4 Interval Database [7]. In this database, there are 10 different scans of each iris. Each iris scan is segmented, normalized, unwrapped and then their corresponding iriscodes are extracted using open source MATLAB algorithm of [8]. Every final iriscode is 9600 bits long. After the iriscode extraction, each subject's iriscodes are aligned to the first sample using the circular shift method. In the circular shift method, one of the iriscodes is shifted circularly, until the minimum difference is obtained between

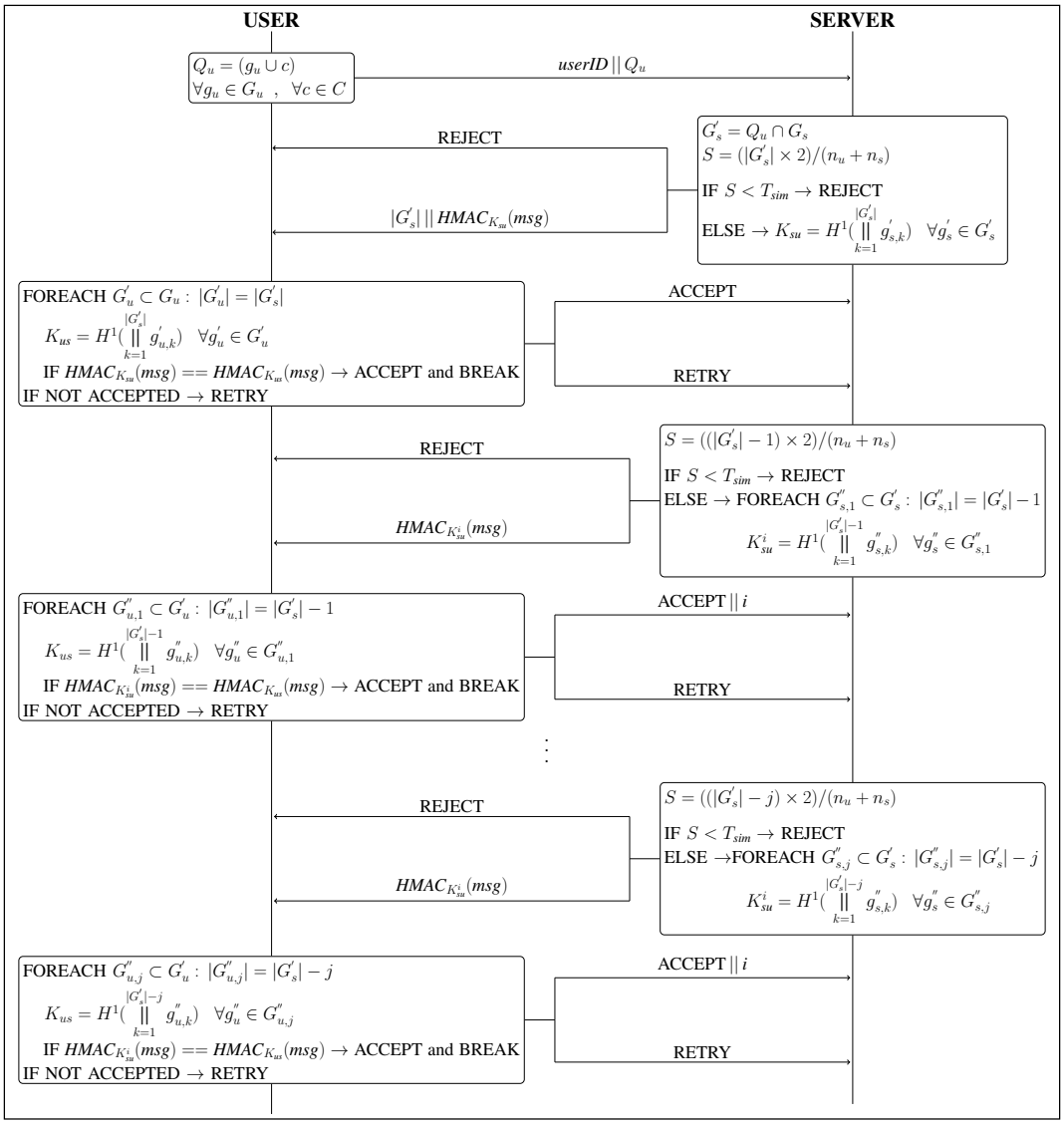


Fig. 1. Our proposed SKA-POB protocol

two iriscodes. First 3 irises are used at the server side, while generating the users' templates. The remaining 7 irises are used for testing. Since it is assumed that the users give 3 iris scans for each test, the remaining 7 iris scans are applied as combinations of 3. So, each user is tested $\binom{7}{3} = 35$ times. Not only genuine tests, but also impostor tests are performed. In an impostor test, each user's template is tested against all other users' templates. The hash function used in the protocol is SHA-256 [9]; hence all of the generated keys are 256 bits long. While the fake blocks are generated, the fake point generation constant, $X$, is taken as 100.

In following subsections, we discuss the correct and incorrect key generation rates of the system and we provide the security analyses of the protocol, as well as representing the randomness and the distinctiveness analyses of the generated keys. Finally, we analyse the complexity of the system.

### A. Correct/Incorrect Key Generation Rates

In both genuine and impostor tests, always a similarity score is calculated (Section II-B). For each subject, the average similarity score is calculated, and they are applied as acceptance threshold values of the system one by one. Acceptance threshold plays the key role while determining the Correct Key Generation Rate (CKGR) and Incorrect Key Generation Rate (IKGR) values, which are the percentage of the genuine subjects who successfully agree on a correct symmetric key with the server and the percentage of the impostor subjects who erroneously agree on a correct symmetric key with the server as if they are genuine users, respectively. Figure 2 shows the IKGR and the CKGR percentages with respect to the different acceptance threshold values. IKGR is $0\%$ and CKGR is $100\%$, when the acceptance threshold is 17. The obtained IKGR and CKGR values are strongly sufficient for a secure bio-cryptographic authentication system.

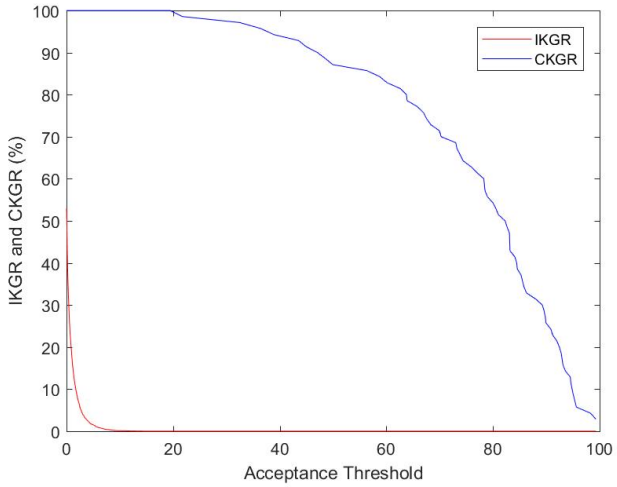


Fig. 2. IKGR and CKGR values - when average scores are picked as threshold

### B. Security Analyses

The attacker's aim is either to impersonate a genuine user or to learn the key between the server and any victim user for eavesdropping purposes. We do not assume a secure channel. Thus, the attacker can obtain all protocol messages including the genuine and fake blocks' list sent by the user and exchanged HMACs. Consequently, the attacker learns the number of blocks used for the key agreement. However, our protocol computationally resists well-known attacks as discussed below.

*1) Brute-Force Attack:* In a basic brute-force attack, the attacker may try to guess the agreed key. However, this type of attack is computationally infeasible, since the key is 256 bits long. For this reason, we describe a more intelligent brute-force attack by making use of the protocol messages. On the server side, the matching is done in a window-based strategy. The attacker can make use of this idea, as well. In order to find genuine blocks, the attacker should select the windows first. It means that the attacker should select $n_{com}^{key}$ different windows, because the attacker needs the genuine blocks which are used in the key generation. In an ideal case, there are $X+1$ values in each window, where $X$ is fake point generation constant. This yields the attack complexity of $att_c = (X+1)^{n_{com}^{key}}$.

In order to calculate the overall attack complexity of the system, the combination given above is calculated after each key agreement. Then, we take the average of the complexity results. The analysis shows that the average attack complexity of the system is $2^{148}$ bits. As discussed in [10], even with custom hardware implementation, the computation of one block of HMAC-SHA256 takes approximately $0.8977$ microseconds. Thus, the above mentioned complexity corresponds to $1.016 \times 10^{31}$ years of attack. As a result, we can conclude that our protocol efficiently resists intelligent brute-force attacks.

*2) Replay Attack:* On the other hand, in a replay attack, the attacker replays the previously exchanged messages between the victim user and the server, in order to impersonate the genuine user and get the agreed key. However, the attacker needs to know the genuine blocks to effectively calculate the generated key; otherwise, (s)he must try all blocks in each window, as explained above. As a result, the complexity of a replay attack would be the same as that of the brute-force attack. Additionally, the attacker might use his/her own iris scans instead of replaying the victim user's inputs. Our protocol shows resistance against this type of impersonation attacks with $0\%$ IKGR.

*3) Randomness:* A good quality cryptographic key should be random and distinctive. The randomness of a key is measured using the Shannon's entropy [11]. In our tests, there are $70$ subjects, and $35$ keys are generated per subject, so $2450$ keys are generated in total. These keys are results of a hash function, and it is known that the hash results have high randomness. Therefore, instead of measuring the randomness of the keys, we analysed the randomness of the concatenations of the blocks accepted as genuine, i.e. used in key generation. The entropy values of these concatenations are given in Figure 3. The more this value approaches to $1$, the more random the key is. In our case, approximately $90\%$ of the concatenations have randomness that are greater than $0.99$, and also all of the concatenations have randomness that are greater than $0.95$. Hence, the randomness of the concatenations is sufficient for being an input to a secure key generation.

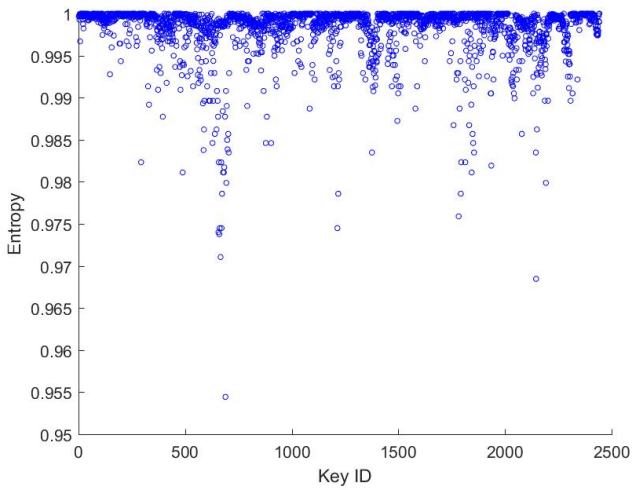


Fig. 3. Randomness results

*4) Distinctiveness:* In order to measure the distinctiveness, we calculate the percentage of Hamming Distance values between all iriscodes. The optimum value for this percentage is $50\%$ since both full distinctiveness and full indistinctiveness give the same amount of information to the attacker. Percentage of average Hamming Distance value between all users' iriscodes is calculated as $49.37\%$, which is quite close to the optimal value of $50\%$. However, due to the fact that not all blocks are used in the key generation process, we need to define another metric for measuring the distinctiveness of the keys, which is called *key-block distinctiveness index*. General idea here is that each block used in agreed keys is

compared with all the blocks of all other users' iriscodes. In each comparison, the number of different bits in correspondent positions are calculated and normalized to 100. In regard to distinctiveness, $x$ bit difference has the same effect as $384-x$ bit difference. Therefore, we need to consider the worst-case scenario, while calculating the distinctiveness. Each calculated value which is greater than $384/2$ is subtracted from 384 and normalized. After the normalization, the minimum value is selected for each block, because the minimum difference is identical to the minimum distinctiveness which is the worst-case scenario. For each key blocks, the average of minimum key-block distinctiveness indices is calculated for each user. The average key-block distinctiveness index value of the system is found as $74.61\%$. As a result, we conclude that after each protocol run of each user, a different key is generated.

*5) Discussions on Correlation Attack:* In our SKA-POB protocol, the user sends genuine and fake iris blocks to the server, as described in Section II-B. This situation can cause information leakage after repeated usage of the system. In order to avoid any kind of correlations among genuine blocks in different runs of the protocol, fake blocks also need to be selected and distributed according to a carefully designed strategy. For this reason, the percentage of the difference between the fake blocks in different protocol runs should be limited. In other words, the allowed maximum difference must be the same for both genuine blocks and fake blocks, which is indeed the acceptance threshold. In that way, it would not be possible to distinguish genuine blocks even after the repeated usage of the system, because fake blocks will also be correlated with each other. In other words, the attacker can find a correlated block for each and every genuine and fake blocks in different protocol runs. As a result, the genuine blocks would be the same as the fake blocks from an attacker's point of view. Such an intelligent design for the protocol is left as a future work.

### C. Complexity Analyses

Computational complexity analysis is realized by adding up the key generation attempts of both parties. The server generates only one key at the beginning of the protocol and in upcoming rounds, computations are made by selecting one less block out of $n_{com}$, the complexity of which is calculated as $\binom{n_{com}}{n_{com}-1}$. Iterations stop when the correct key is generated, which contains $n_{com}^{key}$ blocks. Hence, the average server complexity is average of the sum of all key generation attempts against all subjects. Results show that average server complexity is $2^0 = 1$ hash and HMAC calculation, maximum server complexity is $2^{16}$ hash and HMAC calculations. Further, the user complexity is formalized in the following way. Since the server sends the number of common found blocks, $n_{com}$, the user starts key generation attempts with all possible subsets of the genuine blocks whose size is $n_{com}$, complexity of which is calculated as $\binom{n_u}{n_{com}}$. If the key agreement cannot be established, the user tries subsets of one less block at each round, until the key is generated or the user is rejected. Thus, the average user complexity is the average of the sum of all key generation attempts from all subsets until the protocol stops. Analysis shows that average user complexity is $2^{12}$ hash and HMAC calculations, maximum user complexity is $2^{24}$ hash and HMAC calculations. These values are quite sufficient for a real time application with small response time.

The communication complexity is measured by the number of bits exchanged between the server and the user. Total communication cost of the first message sent by the server is $1320 \times 384 + 32 = 61.88$ KB, where the user ID is a 32 bit integer, each block is 384 bits long, the fake point generation constant is 100, and the average number of elements in $Q_u$ is 1320. The second message of the protocol is either a negative acknowledgment or the number of common found blocks and an HMAC value, sent by the server, and its maximum cost is $32 + 256 = 288$ bits $= 36$ bytes: an acknowledgment is just 1 bit, the number of blocks used in key generation is a 32 bit integer, and HMAC value is 256 bit. User's answer to this message is either *ACCEPT* or *RETRY*, either one of them is only 1 bit. After that, if the protocol continues with a new round, the server sends either a negative acknowledgment or a new list of HMAC values, where the number of elements is $\binom{n_{com}}{n_{com}-1}$. The average number of common found blocks by the server is 19; hence, the average communication cost of this message is $256 \times \binom{19}{18} = 4864$ bits $= 608$ bytes. In the average case, the protocol stops after this round. So the cost of this message is the maximum cost of the messages sent by the server. If the user still cannot verify any of the HMAC values, (s)he sends another *RETRY* message. Otherwise, the user sends a positive acknowledgment with the index of the HMAC value that is verified, which is a 32 bit integer. Thus, the maximum communication cost of this message is $1 + 32 = 33$ bits. Therefore, the total size of the messages transmitted by the server and the user are approximately 644 bytes and 62 KB, respectively. All of the communication costs given above are quite reasonable for today's Internet speeds.

### D. Comparison with the Baseline Protocol

As it is mentioned before, our SKA-POB protocol is built upon one of our previous works, SKA-PB [5], which uses unordered biometrics in a secure key agreement protocol. SKA-POB is indeed an adaptation of SKA-PB for ordered biometrics. In this subsection, we compare our results from both of these works and show that adaptation is performed successfully.

Our SKA-PB protocol was evaluated with 30 subjects from Verifinger Sample Database for fingerprints [13]. In SKA-PB [5], it is stated that the IKGR and CKGR values are $0.57\%$ and $99.43\%$, respectively. In our new SKA-POB protocol, these values are $0\%$ and $100\%$, respectively. These results prove that our protocol can be successfully adapted to different biometrics with different structures.

Besides, the resistance of the SKA-PB protocol against an intelligent brute-force attack is stated to be $2^{94}$ bits [5], which is $2^{148}$ bits in our new SKA-POB protocol. With the same assumption made in Section III-B, in order to break the system, in SKA-PB, it costs $5.6 \times 10^{14}$ years of attack, while in SKA-

POB, it costs $1.016 \times 10^{31}$ years of attack. As a result, we can conclude that our protocol efficiently resist intelligent brute-force attacks even with different biometrics.

Additionally, it is stated in SKA-PB [5] that $92.3\%$ of the fingerprint minutiae concatenations have randomness values greater than $0.98$. Similarly, in our new SKA-POB protocol, approximately $90\%$ of the concatenations have randomness values greater than $0.99$. On the other hand, the average Hamming Distance values among different users' keys is stated to lie in the interval of 120-135 bits, which corresponds to approximately $48.8\%$ [5]. The respective Hamming Distance value for our new SKA-POB protocol is $49.37\%$. As a result, we can say that the randomness and distinctiveness results are sufficient for cryptographic key generation in both of the proposed protocols.

## IV. Conclusions and Future Work

In this paper, we proposed a bio-cryptographic key generation solution using iriscodes. The key is generated from iriscodes directly; it is generated neither randomly, nor any random helper data is used. In our protocol, we divide the iriscodes into blocks and mask them using both one-way hash functions and considerable amount of fake blocks. Fake blocks are distributed uniformly random.

We defined a window-based strategy for comparing the blocks on the server side, by means of which the server does not need to compare all the blocks in the list. Additionally, we define a window-reset threshold, $T_{reset}$. If the distinctiveness of two blocks is less than $T_{reset}$, we reinitialize the window and decrease the number of blocks in a window, which decreases the rest of the required number of comparisons.

We analysed the security performance of our protocol, quality of the generated keys and complexities required by the protocol using a public database [7]. The verification performance of our protocol is calculated as $0\%$ IKGR and $100\%$ CKGR at the same time, which is a perfect case for a biometric system. We showed that our protocol is resistant against intelligent brute-force, replay and impersonation attacks. Considering the quality of the generated keys, we measured randomness of the block concatenations used in the key generation and their distinctiveness. Analyses showed that randomness values are high enough to be used in secure key generation process and each generated key is $74.61\%$ different than all other keys. Besides, in terms of the computational complexity, the server and the user side operations took approximately $1$ and $2^{12}$ hash and HMAC calculations, respectively. Further, we also analysed and showed that the communication complexity of our protocol is suitable for implementing the protocol with today's technology. In today's digital world, security risks are critical as never before. In this context, this protocol works flexibly, does not require huge computational power, and solves the key management problem in order to provide authentication, confidentiality and non-repudiation; so, it can be used effectively to overcome security issues in today's world [12].

As a future work, we plan to enhance our protocol to resist against correlation attacks and to reduce the information leakage after repeated usage.

## References


[1] D. Karaoğlan and A. Levi, "A survey on the development of security mechanisms for body area networks," *The Computer Journal*, vol. 57, no. 1, pp. 1484–1512, 2014.

[2] U. Uludag, S. Pankanti, S. Prabhakar and A. K. Jain, "Biometric cryptosystems: issues and challenges," *Proceedings of the IEEE*, vol. 92, no. 6, pp. 948–960, 2004.

[3] C. Rathgeb and A. Uhl, "A survey on biometric cryptosystems and cancelable biometrics," *EURASIP Journal on Information Security*, vol. 2011, no. 1, pp. 1–25, 2011.

[4] A. Cavoukian and A. Stoianov, "Biometric encryption," *Encyclopedia of Cryptography and Security*, pp. 90–98, 2011.

[5] D. Akdoğan, D. K. Altop, and A. Levi, "Secure key agreement using pure biometrics," *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 191–199.

[6] A. M. Bazen and S. H. Gerez, "Fingerprint matching by thin-plate spline modelling of elastic deformations," *Pattern Recognition*, vol. 36, no. 8, pp. 1859–1867, 2003.

[7] "CASIA Iris Image Database Version 4.0 biometrics ideal test," biometrics.idealtest.org/dbDetailForUser.do?id=4, accessed:2017-11-19.

[8] L. Masek and P. Kovesi, "Matlab source code for a biometric identification system based on iris patterns," *The School of Computer Science and Software Engineering, The University of Western Australia*, 2003.

[9] National Institute of Standards and Technology, *FIPS PUB 180-2: Secure Hash Standard*. pub-NIST, 2002.

[10] M. Juliato and C. Gebotys, "FPGA implementation of an HMAC processor based on the SHA-2 family of hash functions," *University of Waterloo, Tech. Rep*, 2011.

[11] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 4, pp. 623–656, 1948.

[12] S. Sicari, A. Rizzardi, L. A. Grieco and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: the road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.

[13] Neurotechnology Verifinger Sample DB, http://www.neurotechnology.com/download.html, 2015 , Accessed: 2015-02-28