# Validating Biometric Authentication Protocol

Elia El Lazkani
University of Massachusetts Dartmouth
285 Old Westport Road
North Dartmouth, MA 02747, USA
EElLazkani@umassd.edu

Hong Liu
University of Massachusetts Dartmouth
285 Old Westport Road
North Dartmouth, MA 02747, USA
HLiu@UMassD.edu

## ABSTRACT
Biometric sensors/actuators communicating via wireless body area networks has surged wide applications. From seamless healthcare to driverless transportation, biologically inspired systems benefit with productivity growth, energy efficiency, user convenience, and cost reduction. However, their invasive nature raises concerns. Security becomes the urgent task to realize biologically-enabled systems. Biological phenomena, nevertheless, can also play a vital role to secure software. Recently, several biometric authentication protocols have emerged to verify endpoints, promising one-time key for premium security. In spite of bio-enabled security advances, lack of security analysis theories and tools causes uncertainty of their safety.

This paper pioneers an experimentation on assessing security of a well-established biometric authentication protocol. Using the gold standard in software reliability, the work exploits the attack surface leveraging path analysis. The test not only identifies security vulnerabilities in a system but also pinpoints those vulnerabilities at real risk to optimize resource allocation. The automated holistic examination of the authentication process reveals a weakness in the biometric authentication protocol at study. The attack map directs its improvement validated with reexamination. The work also studies the computational complexity of the protocol, thereby, recommends the key length suitable to biometric authentication for wireless body area networks.

## Categories and Subject Descriptors
C.2.0 [**COMPUTER-COMMUNICATION NETWORKS**]: General-Security and protection.

## General Terms
Experimentation, Security, Performance, Verification.

## Keywords
Bio-inspired wireless network security (BWNS), biometric authentication, Wireless Body Area Networks (WBAN), wireless sensor networks, practical realization of physiological biometrics, validation of protocol implementation.

## 1. INTRODUCTION

Wireless Body Area Networks (WBAN) belong to special-purpose Wireless Sensor Networks. Using WBAN, sensors are attached to or embedded in a human body, and they communicate wirelessly among themselves and with other components such as network towers, processors, servers, and etc. WBAN challenges conventional cryptography-based authentication schemes. Due to WBAN heterogeneity, traditional protocols are not practical in daily operation of key management and device mobility, hence Kanjee and Liu proposed a new protocol. Kanjee-Liu Generic Authentication Protocol for Wireless Body Area Network (GAP4WBAN) offers a convenient security solution with plug-n-play feature [1].

This work validates the security strength and studies the computational complexity of Kanjee-Liu GAP4WBAN. We prototype GAP4WBAN on Android and validate our prototype with McCabe IQ. The experimental results confirm the security strength with recommended key length of 4096 bits for RSA asymmetric encryption and demonstrate the protocol applicability in WBAN.

The remaining paper is organized as follows. The problem of protocol validation for bio-inspired authentication process is articulated in Section 2. Section 3 reviews the related work. The automated validation with an industrial software quality control tool is presented in Section 4 while the computational complexity is approximated in Section 5. Section 6 makes the conclusion and points the future research direction.

**Major Contributions**

The work calls the attention of protocol validation in bio-inspired information and communication security. It also demonstrates the feasibility of automating security analysis similar to reliability analysis, the former aiming at intentional faults while the latter dealing with natural faults. The work improves an existing biometric authentication protocol.

## 2. PROTOCOL VALIDATION

How do I know that an authentication protocol is correct? How strong is its security? One way to answer the question is to implement the protocol and experiment with known attacks. However, this method cannot guarantee that the protocol works under new attacks. The other way is to analyze the protocol formally for all possible attacks, but this method is too expensive to apply in practice. The middle ground is to use a software tool to examine the prototype of the protocol to achieve a reasonable level of assurance.

## 3. RELATED WORK

Venkatasubramanian is one of the first inventors of biometric-based authentication in wireless networks of biosensors implanted in the human body [2]. The uniqueness and randomness of


BICT 2014, December 01-03, Boston, United States

DOI 10.4108/icst.bict.2014.258080

physiological signals grow a new branch in WBAN security, namely non-cryptographic authentication and identification [3]. Some particular human body activities, like photoplethysmogram (PPG) [4] and electrocardiogram (EKG) [5], possess the temporal variant feature that advocate precious one-time key. It would be too hard to launch replay or man-in-the-middle attacks as one-time key renders useless by copy. Continuous efforts are made by many to improve the secret key generation performance for on-body devices [6]. New practical solutions are brought out with enhanced effectiveness for authentication in WBAN [5]. Authenticated WBAN has been deployed in broad applications [7]. The research team, largely at the University of Arkansas, revolutionized WBAN authentication without identification by exploiting channel characteristics [4]. Their novel non-cryptographic node authentication scheme, called BANA (Body Area Network Authentication), differentiates signals of a legitimate node and an attacker to form a cluster of good nodes and reject bad nodes from communications. To enable system-wide performance while dealing with diverse devices and advancing technologies, generic programming has promised success in complex system design and development. Generic protocols have been used in traditional network monitoring and network management [8].

# 4. VALIDATE GAP4WBAN

## 4.1 Kanjee-Liu GAP4WBAN

Figure 1 shows the architecture of Wireless Body Area Network (WBAN) adopted by Kanjee-Liu's Generic Authentication Protocol (GAP4WBAN). It contains three types of nodes, each performing different well-defined functions at plug-and-play.

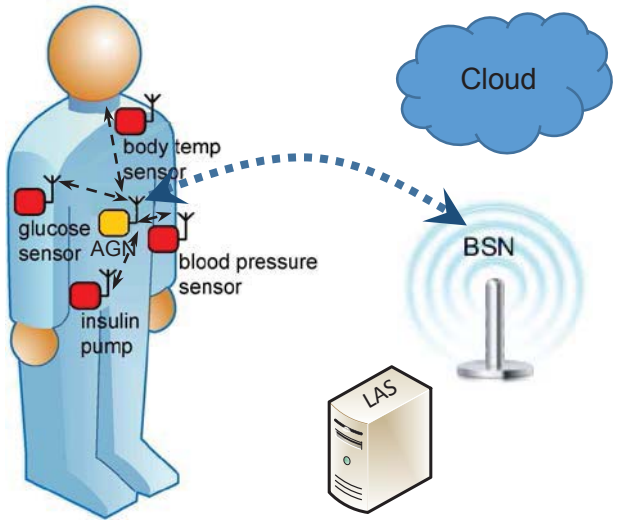


Figure 1. WBAN Architecture

- *Data Collection Nodes (DCNs)* – On-body sensors to collect physiological signs such as body temperature, blood pressure, glucose level, and photoplethysmogram (PPG). DCNs also include on-body actuators such as insulin pump.
- *Aggregation Node (AGN)* – An on-body control unit to cumulate data from DCNs and issue commands to DCNs.
- *Base Station Node (BSN)/Local Authentication Server (LAS)* – An off-body equipment to communicate wirelessly with multiple AGNs in its vicinity. BSN acts as a hot spot for an AGN to relay communications across DCNs and Cloud after successful mutual authentication. LAS serves as a gateway between each patient wearing WBAN and the Master Authentication Server (MAS) in Cloud for remote users to access the WBAN.

In addition, a device called *Oracle* generates the public/private key pairs for AGN and BSN. The following subsections describe the intricate working of the two-tiered authentication protocol proposed by Kanjee and Liu for WBAN [1]. The solution utilizes physiological signals to implement a U-key authentication scheme. The physiological features of uniqueness and randomness facilitate strong security while offer convenient usability, robust reliability, and flexible applicability.

### *4.1.1 On-Body Nodes Physiological Authentication*

The first tier gets the person's on-body nodes, i.e., several DCNs and one AGN, to authenticate among themselves. They can utilize their shared physiological measures to agree upon a symmetric cryptographic key. This secret key is used for their mutual authentication. They also can conduct non-cryptographic authentication without using any key. Due to the large discrepancy between on-body and off-body channels Received Signal Strength (RSS) variation, the on-body nodes are able to accept each other while reject off-body nodes as malicious.

Various physiological signals have been demonstrated for key agreement. Venkatasubramanian et al takes photoplethysmogram (PPG), a measure of the volumetric change in the distention of arteries due to the perfusion of blood during a cardiac cycle [9]. A multidisciplinary cross-campus research team, led by Honggang Wang at the University of Massachusetts Dartmouth, chooses electrocardiogram (EKG), a recording of the heart's electrical activity in terms of the speed and rhythm of the heartbeat as well as the strength and timing of electrical signals passing through each part of the heart [5]. Key agreement among on-body nodes involves two processes. One is to hide the secret key by extracting the feature from the physiological signal measured and constructing its vault. The other is to reveal the secret key by unlocking the vault with the feature shared by its physiological signal measure. Experiments have confirmed that these physiological stimuli are universally measureable, the duration of their capture takes low latency, and the values of individuals' physiology are distinct and vary by time. Therefore, these kinds of physiological signals provide ideal media for authentication, much effective than traditional biometrics such as fingerprints that could be copied for impersonation.

### *4.1.2 Validating/Authenticating AGN*

After registration, AGN initiates mutual authentication with a nearby BSN, in a dynamic manner dictated by a patient's mobility. As the crux between the patient's physical world and his/her cyber space, AGN, upon successful authentication with DCNs described in the previous subsection, will relay its one-time secret key shared with the DCNs to the BSN after AGN and BSN mutually authenticate each other. Thereby, a seamless secure bidirectional channel is established for a closed-loop of patient monitoring and treatment with intervention of caregivers as needed.

Figure 2 on the next page illustrates the cross-body mutual authentication between AGN and BSN. This paper focuses on the second tier of Kanjee-Liu two-tier Generic Authentication Protocol (GAP) for WBAN. The mutual authentication along with key relay involves the five stages listed below.

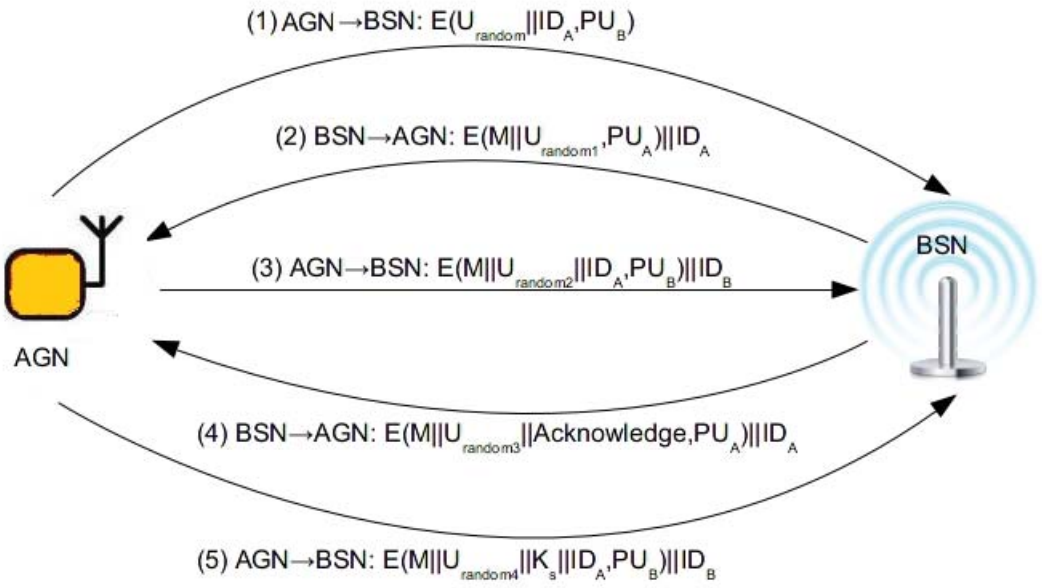


Figure 2. Cross-Body Authentication

Stage 1. AGN initiates authentication: AGN initiates the mutual authentication by sending BSN its own identifier, $ID_A$, encrypted with BSN's public key, $PU_B$. The enclosure of nonce, $U_{random}$, ensures transaction freshness. If there are multiple legitimate BSNs accessible in proximity, AGN chooses one with some optimization algorithm such as power level.

$$AGN \rightarrow BSN: E(U_{random} \,||\, ID_A, PU_B)$$

Stage 2. BSN authenticates to AGN: When BSN receives the authentication request from AGN, BSN uses its matching private key, $PR_B$, to extract AGN's ID. BSN then checks AGN's legitimacy with the hierarchy of authentication servers. If AGN is valid, BSN takes AGN's public key, $PU_A$, to encrypt a message, M, and sends back to AGN along with the synchronized nonce, $U_{random1}$. The fact, that BSN can successfully recover the encrypted $U_{random} \,||\, ID_A$, assures AGN that the responding BSN is the intended BSN.

$$BSN \rightarrow AGN: E(M \,||\, U_{random1}, PU_A) \,||\, ID_A$$

Stage 3. AGN authenticates to BSN: AGN receives the response and sends the next nonce in sequence. This serves the purpose of assuring BSN that the legitimate AGN is communicating at this moment. Using a nonce synchronization, instead of the nonce itself, prevents replay or man-in-the-middle attacks.

$$AGN \rightarrow BSN: E(M \,||\, U_{random2} \,||\, ID_A, PU_B) \,||\, ID_B$$

Stage 4. Mutual authentication completes: The intended BSN acknowledges the genuine AGN. This four-way handshake completes the mutual authentication between AGN and BSN.

$$BSN \rightarrow AGN: E(M \,||\, U_{random3} \,||\, ACK, PU_A) \,||\, ID_A$$

Stage 5. AGN relays the session key: AGN enters Authenticated State. AGN then relays its one-time secret key shared with DCNs, $K_S$, to BSN. This shared key will be used to secure communications among the entities with confidentiality and integrity for the session.

$$AGN \rightarrow BSN: E(M \,||\, U_{random4} \,||\, K_S \,||\, ID_A, PU_B) \,||\, ID_B$$

## 4.2 Prototyping GAP4WBAN

This work focuses on validating the cross-body authentication of Kanjee-Liu GAP4WBAN by prototyping. We choose to implement the protocol on Android in Java. Android <android.com> is the most popular mobile operating system based on the Linux kernel and developed by Google under open source licenses. Designed primarily for touchscreen mobile devices, Android covers a variety of specialized user interfaces: besides smartphones and tablet computers, Android Wear for smartwatches, Android TV for televisions, and Android Auto for cars. Java is Android's choice of the programming language for user interface and the native language in Android Software Development Kit (SDK).

We also choose to use RSA encryption in our prototype because GAP4WBAN adopts asymmetric encryption [1] and Java supports RSA library. Since RSA patent expired, RSA Security released the algorithm to the public domain in 2000. The crack of 768-bit RSA was announced in 2010 [10], therefore the key length must beyond 768 bits. Java RSA library needs to specify the number of bits to generate RSA public and private key pairs. We began with 2048 bit keys and stress-tested the capacity for the ARM processors the authors' smartphones.

Some changes are made to the original protocol for practical reasons. Registration at every time of usage is not realistic, so we take the assumption that the registration is done beforehand, once, in a physically secured location on a secure channel. This change dictates AGN to initiate a session with a BSN in its vicinity, an additional step to the original spec [1]. The initiation step requires a random number and AGN's ID to prevent replay attacks. Though nonce is unnecessary in RSA because RSA does not generate the same encrypted message multiple times, it might be useful in other asymmetric encryption algorithms that do not have the same dynamic feature as RSA.

Java RSA library also posts limitations on the message size no longer than the key length. To go around this problem, we programed the server to break down longer messages into packets and circulates a list of client's public keys during encryption. As a consequence, messages from clients would be limited to the sizes restricted by the server.

On the server side, we implemented a framework to accept multiple connections from clients. Each connection at the server spins its own thread to authenticate the corresponding client. The server and the clients use RSA algorithm to encrypt and decrypt data. The *RSA Key Generator* function creates a set of private/public key pairs with a specific number of bits and saves them in a stream for the server and the client. Furthermore, encrypt/decrypt functions were coded. Figure 3 below shows the process on a sample "Let's simply try to encode this key."

```
Created ServerSocket.
listening on port 3948...

Private Key:
sun.security.rsa.RSAPrivateKeyImpl@fffadddc

Public Key:
Sun RSA public key, 2048 bits
  modulus: 207496591835709802309696072572714363552155619710821736656953 5
  public exponent: 65537

Original Text:
Let's simply try to encode this key

Text Encrypted:
[B@185be8f4

Text Decrypted:
Let's simply try to encode this key
```

Figure 3. Server Encrypting/Decrypting

On the client side, the user interface (UI) leaves a large space for communications with a button to initiate the authentication. Android clients do not need to audit the authentication process by default, it gets "Auth Succeeds" on the server to pass the process. Data communicated between each client and the server will be checked by each other, following the protocol spec. If, at any point during the authentication process, something goes wrong, the authentication process fails, and the connection will be terminated. A client, once authenticated, will print a green message to inform the user successful authentication, as shown in Figure 4.

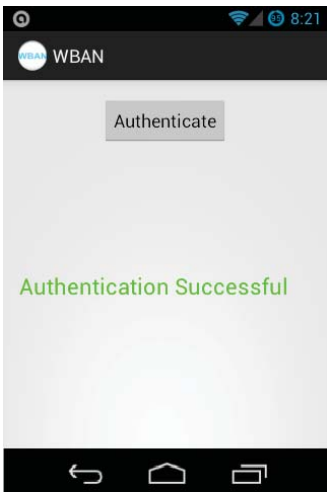


Figure 4. Client Authentication Success

The server, on the other hand, is more verbose. It outputs, for the sake of the proof of concept, each message coming from any client. As shown in Figure 5 below, the server displays the encrypted message from a client and then decrypts it as M for message. After two rounds of M to complete the four-way handshake for mutual authentication, the server receives the session key as K from the client. Thereby, the authentication is successful.

```
Created ServerSocket.
listening on port 3948...
[B@71c94ff3
{"ID":2,"URand":448}
[B@2950fcc7
{"ID":2,"URand":302,"M":"ThisIsM"}
[B@24c535d
{"ID":2,"URand":233,"M":"ThisIsM","K":"This Is K"}
This Is K
Authentication is successful.
```

Figure 5. Server Authentication Success

We overcame some challenges during the project. At the beginning of the project, the choice of the encryption key length was a challenge as we did not know how it would go with slow processors on smartphones. We took an assumption of 2048 bits for a key length as it is significantly larger than the already cracked 768 bits [10]. It turned out that the key size was an appropriate guess per the data at hand. As soon as we started Android development, problems arose due to the limitations of Android APIs. The limitations were placed there to ensure that applications work as intended but, unfortunately, they hindered the work flow for complex workarounds had to be found to implement simple procedures. For example, Android APIs do not allow the main thread to initiate any connection, only child threads can initiate a connection. That, by itself, is not a big issue. More problems were encountered when we need to update the main user interface, which can only be updated by the main thread, from the created thread. We overcame the problem by using the API handlers.

Half way through the project, after most of the coding was finished, there were problems with finding libraries to work on both Android devices and Oracle Java on the Windows server. After sorting out the library dependencies, more problems surfaced out with converting encrypted binary data to string. The conversion from one data type to the other is irreversible. To overcome this obstacle, the encrypted data was encoded in base64 before being sent and decoded on the other side, both ways. With most issues out of the way, another problem arose with the standard compatibility of RSA encryption and decryption between Android platform and the server's Oracle Java. It seems that specifying RSA alone for the encryption and decryption algorithm was not detailed enough because the two sides use different standards. We had to specify the encryption/decryption algorithm as "RSA/ECB/PKCS1Padding" on both sides, which solved the compatibility problem.

At the moment we celebrated the code running, we realized our failure in result verification. After extensive debugging and reasoning, we identified our misunderstanding of RSA modulus. With RSA asymmetric encryption, the message to be encrypted must be smaller than the key. This limitation hindered the key distribution proposed in the original GAP4WBAN protocol when a client sends its public key encrypted with the server's public key to the server at the initialization stage. The remedy is our assumption that the server has obtained the public keys of a client at registration time. The future work section suggests a better solution to the

problem by the server maintaining a key ring of all clients sorted by client IDs.

## 4.3 McCabe IQ

McCabe IQ is the gold standard in software reliability, a path-oriented software quality control tool <McCabe.com>. With dynamic code analysis, the tool identifies potential security vulnerabilities and weaknesses in applications and pinpoints those at real security risk. Among 12 application architecture categories to identify security risk, McCabe IQ lists authentication as the Number One risk control category. Automation of security analysis optimizes the allocation of scare resources to identify potential vulnerabilities, aiming at those with the greatest positive impact on security risk.

McCabe IQ can expedite the process of security analysis by decomposing the application from a security risk standpoint. It dramatically reduces the time and cost to assess potential vulnerabilities and to mitigate the risk.

McCabe IQ provides various features to analyze security vulnerabilities in software implementation: path and subtree analysis, code coverage analysis, and compliance checking tool. McCabe IQ is very powerful at finding all the possible paths that an application can take in the code. In our case, all the paths will either lead to a safe exit of the thread or into the catch if something goes wrong at any given point. In the latter case, the catch will send the client an authentication failure token and will attempt to close the connection before crashing or exiting the application. To demonstrate McCabe IQ for finding security holes in software implementation, we injected a fault in our code. Removing failsafe, the try-catch code lines, and deliberately inducing wrong authentication to mimic malicious attacks, the WBAN mistakenly passes the authentication due to catch nonexistence.

## 4.4 Vulnerability Detection

We analyze the five-stage authentication by modeling the server with a three-stage finite state machine. The finite state machine models the server's behavior as summed in Figure 6.

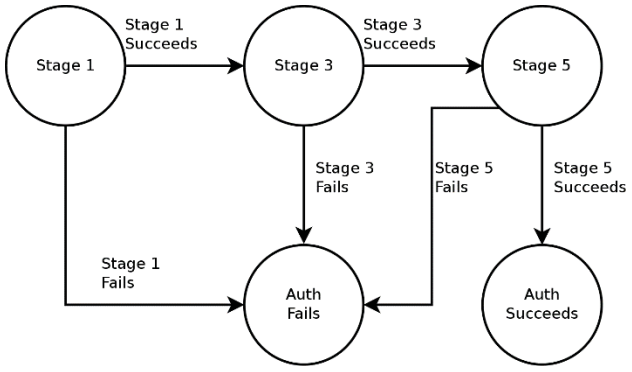


Figure 6. Finite State Machine of Server Authentication

Each of the three stages in the server (Stage 1, 3, and 5 of the Cross-Body Authentication depicted in Figure 2 earlier) is assigned a state. There are two additional states: *Auth Fails* and *Auth Succeeds*. At each of the states corresponding to the server's stages, a success event triggers a transition to the next stage state while all failure events trap the server to the Auth Fails state. Initially, the server starts at the Stage 1 state, waiting for an authentication request from a client, processes the request upon arrival, and transfers to the Stage 3 state if the server is notified the client passing its legitimacy test. Otherwise, the server moves to the Auth Fails state, which is also triggered by other exceptions including the client fails to recognize the server's authentication at Stage 2. At the Stage 3 state, the server expects the client to authenticate itself and, upon success, transitions to the Stage 5 state. Likewise, the server goes to the Auth Fails state if the client fails its authentication test or the mutual authentication cannot be completed at Stage 4. At last, the server receives the secret session key from the client at Stage 5, and the entire Auth Succeeds.

The protocol was coded with stability and security in mind. All the classes, implemented as packages in Java, are wrapped with *try and catch* and throw *exceptions* themselves if at any point something goes wrong. The thread running on the server is also wrapped with a *try and catch* but catches all the exceptions either thrown by the libraries used or by the classes written for the protocol. The catch in the thread where the process of authentication occurs includes a code to notify the client that the authentication failed, which triggers another catch on the client to end the connection and to kill the thread. In this way, we can prevent deadlocks between the server and the client during an abnormal process or by attacks. The code snippet is shown in Figure 7.

```
// If Exception, send a fail and close the connection
JSONObject failed = new JSONObject();
failed.put("b", "failed");
output.writeObject(failed.toString());
output.flush();
System.out.println("Authentication failed.");
output.close(); // closes needed to terminate connection
input.close(); // otherwise user's window goes mute
linkto.close();
```

Figure 7. Failsafe: Catching Exception

After making the necessary modifications, we feed the code to McCabe IQ for analysis where the process of authentication occurs. Figures 8 and 9, on the next page, are the screenshots of McCabe IQ, the path graph and the code annotation side by side. Figure 8 on the left, is a snapshot of the path graph displayed by McCabe IQ that depicts all the path combinations. Due to the size and complexity of the graph, we only cropped the relevant part to our example. Same with Figure 9 on the right, an annotation of the code highlights all the nodes numbered in the graph.

Let us assume a scenario of man-in-the-middle attack. The attacker, not understanding the information passed back and forth between the client and the server, injects a random packet. The original prototype will of course catch that false packet, raising an exception due to either inability to parse the message or inability to decrypt it, and will jump to the failsafe catch snippet that closes the connection safely and notifies the client to close the connection as well. The version to demonstrate McCabe IQ applicability in security, shown in Figure 9, commented off the catch line. Referring to the path graph, depicted in Figure 8, we can see a path from node 249 to node 323 and then to node 324. Node 249 comes earlier in the code and an IF statement that checks if the finite state machine is actually done from its work, in this case it hasn't due to an error. After that failure, we will jump down to node 323 which is shown in Figure 9 to be a closing bracket signifying the end of the thread and the path takes us again to node 324, another closing bracket, bypassing the failsafe procedure that would have otherwise closed the connection and sent the client an authentication fail for it to close its connection as well.

The case identifies two vulnerabilities. The first lays in the weakness that an attacker can explore to crash the server by simply sending random data. The second vulnerability is that the client, not knowing the server has crashed, keeps its connection open waiting for a reply, giving the attacker an edge to trick the client into a false success of authentication.

The improved version of GAP4WBAN is published in the sequel work by Kanjee and Liu [11].

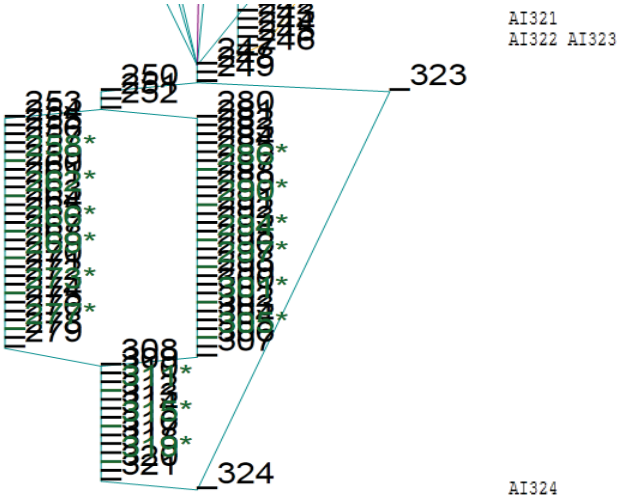


Figure 8. McCabe Path Graph

```
            return;
         }
//    }
/*    catch (Exception error) {
         try {
            // If Exception, send a fail and close the connection
            JSONObject failed = new JSONObject();
            failed.put("b", "failed");
            output.writeObject(failed.toString());
            output.flush();
            System.out.println("Authentication failed.");
            output.close(); // closes needed to terminate connection
            input.close(); // otherwise user's window goes mute
            linkto.close();
         }
         catch (Exception closing) {
            closing.printStackTrace();
            return;
         }

         error.printStackTrace();
         return;*/
   //}
}
```

Figure 9. McCabe Code Annotation

## 5. COMPLEXITY APPROXIMATION

Due to the nature of RSA, the message size, though required to be smaller than the key length, does not affect the complexity of GAP4WBAN protocol. What affects the complexity is the length of the key. Further we do not concern the time spent on RSA key generation, though most time consuming, because it is done separately from the authentication protocol. Encryption with public key uses a smaller exponent while decryption with private key uses a significantly bigger exponent, both with the same modulus. Therefore, decryption usually takes longer than encryption. On the other hand, encryption and decryption occur on both the server and the client. We implemented the server on Windows which holds much more computational power than the client that is running on a low power consumption ARM processor. We studied the system bottleneck by experimenting the decryption performance on a smartphone.

For the purpose of gathering data, we wrote an application on Android platform that generates RSA key pairs with different lengths in bits, encrypts a predefined message, decrypts the message, and measures the time it takes from encryption to decryption. For example, to measure the time of decryption, we record the system clock, in nanoseconds, once before decryption and once after and then calculate their difference.

Due to the key generation complexity, we could only generate keys up to 8192 bits on Android. Beyond that key length, the time would fold over 10 minutes. Even though our application took a while to generate keys up to 8192 bits, it worked as expected and gave us meaningful time measures with different key lengths.

There are multiple ways to calculate the theoretical complexity, big O. We used an empirical method. With our application to measure Android performance, we use the data collected by the application to approximate the complexity on the device tested. The device is a low-end Android phone, with a slow dual-core CPU and a small amount of memory. The device represented current smartphones affordable by the general public.

We then applied *Mathematica* <wolfram.com/mathematica>, a computational software. First, we plotted the data. Then, we approximated the empirical complexity with *FindFit* function. The data at hand exhibits an exponential growth so we assume $f(t) = O(K^n)$, where K is the key length in bits. *FindFit* finds a solution, to the data at hand, for n converging to the value of 2.34767. We also plotted the complexity of $O(K^{2.34767})$, As shown in Figure 10, the two curves possess the matching similarities, empirically confirms the complexity model discovered.

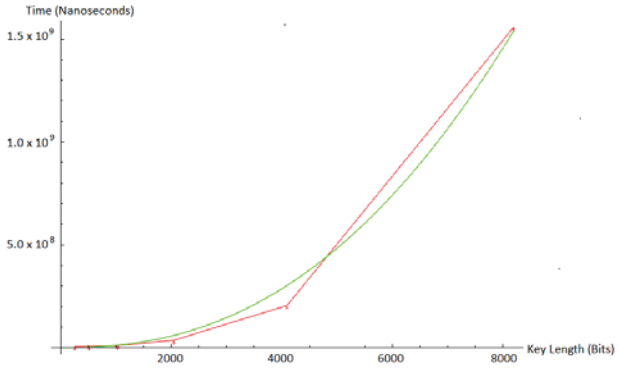


Figure 10. Computational Complexity

The complexity study convinces the usability of GAP4WBAN on Android devices up to a key length of 4096 bits. Since RSA keys no more than 768 bits have been cracked [10], the threshold is set at 1024 bits on Android. With innovative neuronal signaling, such as the stochastic optimizer proposed in [12], to speed up WBAN

communications, this study demonstrates the feasibility and applicability of biometric authentication protocols in daily usage.

## 6. CONCLUSION

This paper presents an interesting work of validating biometric authentication protocol, exemplified with GAP4WBAN [1]. The validation contains two folds: security strength analysis and computational complexity approximation. The protocol's security is analyzed by prototyping GAP4WBAN with Android for clients and Windows for servers. The prototype codebase is then examined automatically by McCabe IQ to identify its security risks. The effectiveness of McCabe IQ is confirmed with injecting false code lines to test its ability to detect exploitable paths within a codebase. A special application is developed to study the computational complexity of GAP4WBAN. With the experimental data collected by the application about RSA decryption times over varying private key size on Android, an empirical model is set by Mathematica to approximate the computational complexity.

The validating process reveals several weaknesses in the original Kanjee-Liu's generic authentication protocol for WBAN [1]. This work contributes to direct their improvement towards the current version of GAP4WBAN [11]. We have overcome several implementation problems, and lessons learned are beneficial to general bio-inspired information security solutions. The prototype demonstrates its robustness, verified by McCabe IQ. Due to RSA's computational complexity and its applicable limitations, we recommend the use of a symmetric encryption, like AES, to encrypt data, but the use of asymmetric encryption, like RSA, to distribute the shared key for symmetric encryption. Finally, our study on GAP4WBAN computational approximation confirms its feasibility in today's mobile devices, up to 4096 bits of key lengths.

Our work calls for security validation in bio-inspired information and communication technologies. We need to validate the other components involved in GAP4WBAN and to validate their integration for a holistic authentication. Invalidated security poses more danger than no security due to off-guard by false negative. Future research directions to security validation include autonomous key management in WBAN, security metrics and measurements unique and suitable to bio-inspired systems, theoretical and empirical models to quantify security strength and computational complexity for facilitating engineering tradeoffs, and software tools to automate protocol validation and prototype verification for bio-inspired wireless network security.

## 7. ACKNOWLEDGMENTS

We thank Mohammed Raza Kanjee, Lance Fiondella, and Xiaoqin (Shelley) Zhang for their insightful discussions and feedback at various stages of the research. We thank McCabe Software Company for offering their McCabe IQ package to this research free of charge. Our thanks also go to the anonymous reviewers for their valuable comments to improve the paper.

This research is sponsored in part by University of Massachusetts President's Office for Cybersecurity Curriculum Initiation Funding 2014-2015 and the Massachusetts Transportation Information Consortium.

## 8. REFERENCES

[1] M. R. Kanjee and H. Liu, "A Generic Authentication Protocol for Wireless Body Area Networks," BODYNETS, Boston, MA, USA, September 30 - October 2, 2013.

[2] S. Cherukuri, K. Venkatasubramanian and S. Gupta, "Biosec: a biometric based approach for securing communication in wireless networks of biosensors implanted in the human body," in *2003 International Conference on Parallel Precessing Workshops (ICPPW 2003)*, Kaohsiung, Taiwan, October 6-9, 2003.

[3] K. Zeng, K. Govindan and P. Mohapatra, "Non-cryptographic authentication and identification in wireles networks," *IEEE Wireless Communications - Security and Privacy in Emerging Wireless Networks,* pp. vol. 15, no. 5, pp. 56-62, October 2010.

[4] L. Shi, M. Li, S. Yu and J. Yuan, "BANA: Body Area Network Authentication Exploiting Channel Characteristics," *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Signal Processing Techniques for Wireless Physical Layer Security,* 3013.

[5] Z. Zhang, H. Wang, V. Vasilakos and H. Fang, "ECG-Cryptography and Authentication in Body Area Networks," *IEEE Transactions on Information Technology in Biomedicine,* pp. vol. 16, no. 6, pp. 1070-1078, June 26, 2012.

[6] L. Yao, S. T. Ali, V. Sivaraman and D. Ostry, "Improving Secret Key Generation Performance for On-Body Devices," in *6th International Conference on Body Area Networks (BodyNets)*, Beijing, China, 2011.

[7] C. Rong and H. Cheng, "Authenticated Health Monitoring Scheme for Wireless Body Sensor Networks," in *7th International Conference on Body Area Networks (BodyNets)*, Oslo, Norway, 2012.

[8] M. Dam and R. Stadler, "A Generic Protocol for Network State Aggregation," in *Radiovetenskap och Kommunikation (RVK)*, Linkoping, Sweden, 2005.

[9] K. K. Venkatasubramanian, S. K. Gupta and A. Banerjee, "PSKA: Usable and Secure Key Agreement Scheme for Body Area Networks," *In IEEE Transactions on Information Technology in Biomedicine,* vol. 14, no. 1, pp. 60 - 68, 2010.

[10] T. Kleinjun, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. A. Osvik, H. t. Riele, A. Timofeev and P. Zimmermann, "Factorization of a 768-bit RSA modulus," Cryptology ePrint Archive, Report 2010/006, 2010.

[11] M. R. Kanjee and H. Liu, "Authentication and Key Relay in Medical Cyber-Physical Systems," *Wiley's Security and Communications Networks journal,* 8 May 2014.

[12] J. Suzuki, D. H. Phan and H. Budiman, "A Nonparametric Stochastic Optimizer for TDMA-Based Neuronal Signaling," *IEEE Transactions on NanoBioscience,* vol. 13, no. 3, pp. 244-254, September 2014.