



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ERINYES: A CONTINUOUS AUTHENTICATION PROTOCOL

by

Fabian A. Lopez and Gage A. Hathaway

September 2022

Thesis Advisor:
Co-Advisor:

Gurminder Singh
Vinnie Monaco

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 2022	3. REPORT TYPE AND DATES COVERED Master's thesis		
4. TITLE AND SUBTITLE ERINYES: A CONTINUOUS AUTHENTICATION PROTOCOL			5. FUNDING NUMBERS	
6. AUTHOR(S) Fabian A. Lopez and Gage A. Hathaway				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) The need for user authentication in the digital domain is paramount as the number of digital interactions that involve sensitive data continues to increase. Advances in the fields of machine learning (ML) and biometric encryption have enabled the development of technologies that can provide fully remote continuous user authentication services. This thesis introduces the Erinyes protocol. The protocol leverages state of the art ML models, biometric encryption of asymmetric cryptographic keys, and a trusted third-party client-server architecture to continuously authenticate users through their behavioral biometrics. The goals in developing the protocol were to identify if biometric encryption using keystroke timing and mouse cursor movement sequences were feasible and to measure the performance of a continuous authentication system that utilizes biometric encryption. Our research found that with a combined keystroke and mouse cursor movement dataset, the biometric encryption system can perform with a 0.93% False Acceptance Rate (FAR), 0.00% False Reject Rate (FRR), and 99.07% accuracy. Using a similar dataset, the overall integrated system averaged 0% FAR, 2% FRR and 98% accuracy across multiple users. These metrics demonstrate that the Erinyes protocol can achieve continuous user authentication with minimal user intrusion.				
14. SUBJECT TERMS machine learning, biometric encryption, security, FAR, false acceptance rate, FRR, false reject rate			15. NUMBER OF PAGES 97	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

ERINYES: A CONTINUOUS AUTHENTICATION PROTOCOL

Fabian A. Lopez
Lieutenant, United States Navy
BS, University of Arizona, 2014

Gage A. Hathaway
Lieutenant, United States Navy
BS, University of Connecticut, 2014

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2022**

Approved by: Gurminder Singh
Advisor

Vinnie Monaco
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The need for user authentication in the digital domain is paramount as the number of digital interactions that involve sensitive data continues to increase. Advances in the fields of machine learning (ML) and biometric encryption have enabled the development of technologies that can provide fully remote continuous user authentication services. This thesis introduces the Erinyes protocol. The protocol leverages state of the art ML models, biometric encryption of asymmetric cryptographic keys, and a trusted third-party client-server architecture to continuously authenticate users through their behavioral biometrics. The goals in developing the protocol were to identify if biometric encryption using keystroke timing and mouse cursor movement sequences were feasible and to measure the performance of a continuous authentication system that utilizes biometric encryption. Our research found that with a combined keystroke and mouse cursor movement dataset, the biometric encryption system can perform with a 0.93% False Acceptance Rate (FAR), 0.00% False Reject Rate (FRR), and 99.07% accuracy. Using a similar dataset, the overall integrated system averaged 0% FAR, 2% FRR and 98% accuracy across multiple users. These metrics demonstrate that the Erinyes protocol can achieve continuous user authentication with minimal user intrusion.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	RESEARCH QUESTIONS.....	2
B.	CONTRIBUTIONS.....	2
C.	THESIS OVERVIEW	3
II.	BACKGROUND AND RELATED WORK.....	5
A.	BIOMETRIC AUTHENTICATION	5
B.	BIOMETRIC SYSTEMS	6
C.	BIOMETRIC BASED KEY RELEASE SYSTEMS	9
1.	Cancelable Biometrics	10
2.	Biometric Encryption	12
D.	RELATED WORK	14
1.	Physical Authentication.....	14
2.	Behavioral Authentication	14
E.	KERBEROS	19
F.	SUMMARY	23
III.	ERINYES PROTOCOL INTRODUCTION.....	25
A.	NEURAL NETWORKS	27
1.	Keystroke LSTM.....	27
2.	Mouse Cursor LSTM.....	31
B.	BIOMETRIC ENCRYPTION.....	32
C.	PROTOCOL FRAMEWORK.....	34
1.	Security	34
2.	Enrollment	35
3.	BE Key Binding.....	36
4.	Verification	37
5.	Authentication	41
D.	SUMMARY	44
IV.	TESTING DESIGN, METRICS AND RESULTS.....	45
A.	KEYSTROKE MODEL AUTHENTICATION TESTING	45
B.	KEYSTROKE MODEL AUTHENTICATION TESTING RESULTS	46
C.	MOUSE CURSOR MOVEMENT MODEL AUTHENTICATION TESTING	51

D.	MOUSE CURSOR MOVEMENT MODEL AUTHENTICATION TESTING RESULTS	52
E.	COMBINED LSTM MODELS' AUTHENTICATION TESTING.....	55
F.	COMBINED LSTM MODELS' AUTHENTICATION TESTING RESULTS.....	55
G.	BIOMETRIC ENCRYPTION.....	57
H.	BIOMETRIC ENCRYPTION TESTING RESULTS.....	58
I.	FULL SYSTEM TESTING.....	60
J.	FULL SYSTEM INTEGRATION TESTING RESULTS	61
K.	SUMMARY	63
V.	CONCLUSION	65
A.	SUMMARY	65
B.	CONCLUSIONS	65
C.	DISCUSSION	66
D.	FUTURE WORK.....	67
APPENDIX A. ERINYES AUTHENTICATION CERTIFICATE		69
APPENDIX B. SAMPLE ERINYES LOGS.....		71
LIST OF REFERENCES.....		75
INITIAL DISTRIBUTION LIST		81

LIST OF FIGURES

Figure 1.	Generic Biometric System. Source: [6].	7
Figure 2.	Imposter v Genuine Match Plot. Adapted from [8].	9
Figure 3.	TypeNet’s LSTM Architecture. Source: [28].	18
Figure 4.	Kerberos Authentication Protocol. Source: [30].	20
Figure 5.	Erinyes Protocol.	25
Figure 6.	Extracted Temporal Features from Keystroke Dataset. Source: [28].	28
Figure 7.	Erinyes Keystroke LSTM Layers	29
Figure 8.	Triple Loss Architecture. Source: [28].	30
Figure 9.	Loss Function for Triplet Loss. Source: [28].	31
Figure 10.	Fuzzy Vault Process. Adapted from [39], [40], [41], and [42].	33
Figure 11.	Erinyes Encryption Process.	35
Figure 12.	Erinyes Enrollment.	36
Figure 13.	Private Key Binding.	37
Figure 14.	Public Key Binding.	37
Figure 15.	Erinyes Client-Side Verification Process.	38
Figure 16.	Erinyes Server-Side Verification Process.	40
Figure 17.	Erinyes Server-Side Verification Acknowledgement.	40
Figure 18.	Erinyes Server-Side Authentication.	42
Figure 19.	Erinyes Client-Side Authentication	43
Figure 20.	Erinyes Client-Side Authentication Reply	43
Figure 21.	Erinyes Server-Side Certificate Issuance.	44
Figure 22.	Keystroke LSTM t-SNE Projections.	48
Figure 23.	Erinyes Keystroke LSTM Metrics (5 Enrolled Users).	49

Figure 24.	Erinyes Keystroke LSTM Metrics (100 Enrolled Users).	50
Figure 25.	Erinyes Keystroke LSTM Metrics (500 Enrolled Users).	50
Figure 26.	Erinyes Keystroke LSTM Metrics (1000 Enrolled Users).	51
Figure 27.	Mouse Cursor Movement LSTM t-SNE Projections.	52
Figure 28.	Mouse Cursor Movement LSTM Metrics (Round 1).	53
Figure 29.	Mouse Cursor Movement LSTM Metrics (Round 2).	54
Figure 30.	Mouse Cursor Movement LSTM Metrics (Round 3).	54
Figure 31.	Combined Keystroke and Mouse LSTM Metrics.	56
Figure 32.	Erinyes Biometric Encryption Metrics (108 Mixed Samples).	58
Figure 33.	Erinyes Biometric Encryption Results (59 Genuine Samples).	59
Figure 34.	Erinyes Biometric Test Results for Enrollment Embeddings.	60
Figure 35.	Erinyes Full System Metrics (114 Mixed Sample Test).	62
Figure 36.	Erinyes Issued Limited Duration Certificate.	63
Figure 37.	Sample Erinyes Server Log.	63
Figure 38.	Sample Erinyes Client Log.	63
Figure 39.	Zero Trust Architecture. Source: [44].	67
Figure 40.	Erinyes Issued Certificate.	69
Figure 41.	Erinyes Server Log Connections.	71
Figure 42.	Erinyes Server Log Part 1 ('127.0.1', 52349).	72
Figure 43.	Erinyes Server Log Part 2 ('127.0.1', 52349).	73
Figure 44.	Erinyes Client Log Part 1 ('127.0.0.1', 52349).	73
Figure 45.	Erinyes Client Log Part 2 ('127.0.0.1', 52349).	74

LIST OF ACRONYMS AND ABBREVIATIONS

AES	Advanced Encryption Standard
ANN	Artificial Neural Network
ASCII	American Standard Code for Information Interchange
BE	Biometric Encryption
BEAT	Biometrics and Evaluation Testing
CNN	Convolutional Neural Network
DES	Data Encryption Standard
ECC	Error Correcting Codes
EER	Equal Error Rate
FAR	False Acceptance Rate
FRR	False Reject Rate
HL	Hold Latency
IL	Inter Key Latency
IP	Internet Protocol
KDC	Key Distribution Center
LSTM	Long Short-Term Memory
ML	Machine Learning
NIST	National Institute of Standards and Technology
OS	Operating System
PA	Policy Administrator
PDP	Policy Decision Point
PE	Policy Engine
PEP	Policy Enforcement Point
PKI	Public Key Infrastructure
PL	Press Latency
POHMM	Partially Observable Hidden Markov Model
RFC	Request for Comments
RL	Release Latency
RNN	Recurrent Neural Network
RSA	Rivest - Shamir – Adleman

SSO	Single Sign-On
SVM	Support Vector Machine
TAR	True Acceptance Rate
TGS	Ticket Granting Server
TGT	Ticket Granting Ticket
T-SNE	T-distributed Stochastic Neighbor Embedding
VM	Virtual Machine
VPN	Virtual Private Network
ZTA	Zero Trust Architecture

ACKNOWLEDGMENTS

We have many people to thank for getting us to the end of this journey. We want to start by thanking Dr. Vinnie Monaco for choosing to remain as our co-advisor throughout his new adventure away from NPS. His advice and guidance throughout challenged us to pursue bigger questions and harder tasks which made the ultimate project much more successful. In a similar manner, we also want to thank Dr. Gurminder Singh for taking the time to be our advisor and provide that initial problem statement to build upon. His push for “what else can be done” made our team incrementally improve on our ideas. Lastly, we want to thank our wives: Lorena and Ashley. Lorena, you held together a family of two energetic boys while we got to lock up in rooms and code. Thank you for putting up with our long nights of work. Ashley, thank you for being our audience to all our frustrations and all our practice presentations. You sat there patiently taking notes, giving us feedback, and trying to cheer us up when we felt the sky was falling. We are proud of where we have gotten, but this truly was a journey we could not have done alone.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The number of digital interactions involving sensitive data has been rapidly growing. As [1] describes, recent events, such as the worldwide COVID-19 pandemic, have further accelerated the digitization of transactions that normally occurred on paper. DocuSign’s website presents digital transactions as anything that may include verifying, signing, or sending documents such as contracts and agreements during real estate purchases, or involving bank accounts and loans [2]. In [3], the authors specifically describe that the number of financial transactions that occur via digital means have increased by nearly 10% in the past 4 years and [4] projects an increase in “a transactional value of 12.31% resulting in a projected total amount of \$15 trillion by 2027” [4]. As the number of transactions increases, the security risks multiply. As discussed in [5], many digital transaction systems tend to rely on traditional cryptographic systems to provide secure exchanges. The authors in [5] further describe that since these systems require mutual secrecy, difficult-to-guess keys are generated and either encrypted or hashed to be stored on a user’s local device, only to be released upon proper authentication. Typically, passwords are chosen as the authentication method. For the user, this interaction is transparent. When the user needs to digitally sign or create a digital transaction, they present an identifier, typically a username, and their authenticator, typically their password. If the identifier and authenticator are the correct combination, the system authenticates the user. However, as [5] describes, in the background, the system is using the credentials to verify the user, release the user’s cryptographic key, and the key is the ultimate authentication. The authors in [5] describe these type of authentication systems as “password release” systems.

Password release systems present two general issues. First, what happens if the password is stolen? As [2] describes, digital transactions are legally binding documents. Although there are prevention methods which can be taken to avoid stolen passwords, such as varying the difficulty requirements set on the password depending on the severity of the system, the system developers must balance out the risk of falsely authenticating a user with burdening the user’s ability to remember the password. This risk balance can be

augmented by using multi factor authentication (MFA) or by limiting how many transactions the user will be allowed to make. This latter point presents the second issue of password release systems, how often should the system attempt to authenticate the user: once only or once per transaction? Once only authentication lessens the intrusion on a user's experience but increases the security risk of a stolen authentication. On the contrary, once per transaction authentication ensures each transaction is authenticated, but burdens true user's experience and fails to solve the issue of a stolen password.

A. RESEARCH QUESTIONS

Password released cryptography systems by themselves provide an inherent risk. If the password is stolen, then the authentication factor is compromised, and the secrecy associated with the corresponding cryptographic key is no longer valid [5]. For this reason, these systems are usually augmented with backup methods such as multi factor authentication. The goal of our study is to:

1. Identify how a biometric encryption system based on behavioral biometrics, specifically keystroke timing sequences and mouse cursor movements, could provide a feasible alternative to a password release system.
2. Additionally, if the biometric system performs accurately, how can it be used to provide continuous authentication?

B. CONTRIBUTIONS

We propose an alternative to a password release system that could also provide continuous authentication. Our specific contributions are:

- We showed that rich feature embeddings produced by Artificial Neural Networks (ANNs) optimized with contrastive learning techniques can be used in a scalable biometric authentication system.
- We highlighted that a multimodal system had minimal impact on performance and significantly improved security.

- We demonstrated that behavioral biometrics, specifically keystroke dynamics and mouse cursor movements, can be utilized to produce a high-performance biometric encryption system. Although there is previous research of other modalities in a biometric encryption system, to our knowledge, there is limited work done on these modalities.
- Lastly, we implemented a biometric encryption system in a multi-threaded, client-service architecture to provide continuous authentication without causing any user distraction or additional work for the user.

C. THESIS OVERVIEW

For this study, we began by researching background and related work into alternatives to password release systems. We focused our research on biometric authentication and single-sign-on (SSO) systems. In Chapter II, we detail the findings of our research, as well as how those systems have been enhanced by improvements in machine learning (ML) and biometric encryption. In Chapter III, we propose a protocol based on our findings that leverages state of the art ML models, biometric encryption of asymmetric cryptographic keys, and a trusted third-party client-server architecture to continuously authenticate users based on their behavioral biometrics. We propose that this protocol could provide an alternative to password release systems as well as provide continuous authentication with minimal user intrusion. In Chapter IV, we describe our testing procedures for the protocol as well as the results of our trial runs. Lastly, in Chapter V, we conclude with an evaluation of our protocol, potential implications, and propose future work that can enhance the framework we have developed.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND RELATED WORK

Many traditional authentication systems rely on possession of cryptographic secret keys. These keys are long and implausible to remember, so they are usually stored on an individual's device through encryption or hashing and released, when needed, using some form of authentication, typically a password. However, the passwords used to release the keys come with their own issues. If the password difficulty settings are too low, the passwords can be compromised using brute force or simple dictionary attacks. On the contrary, if the settings are too high, users may fall victim to social engineering trying to use aids, such as sticky notes, in remembering their passwords. As Uludag et al. argue in [5], once passwords are compromised, the secret keys are compromised, and a system based on keys being kept secret falls apart. In traditional systems, one solution to this problem is having the ability to change passwords and renew those secret keys. If a user believes their password, and by association their secret key, has been compromised, they can request a new secret key, set a new password to authenticate key release, and re-establish their trusted connections with other users under this new key. However, there are administrative and operational overheads to accomplish those tasks. For these reasons, there are alternative solutions to password release systems that have been proposed, including biometric release systems and single sign-on systems, which we explore in our thesis. This chapter explores key concepts and background work into these solutions.

A. BIOMETRIC AUTHENTICATION

Biometric authentication is the verification of an ownership claim to a given template. As Uludag et al. argue in [5], biometrics are inherently more reliable than passwords because there is no need to remember anything. As such, an individual cannot forget or lose their method of authentication, they cannot share it, it is extremely difficult to copy, and the authentication can only occur with the user present. As described on a security online news article in [6], a biometric sample, data captured by a specialized device, is used to extract unique features, and create a digital representation of the user, called a biometric template, that is stored in a database for biometric authentication and/or

identification. One-to-one comparisons to the database are used for authentication or verification, and identification is a one-to-many comparison [6].

B. BIOMETRIC SYSTEMS

Biometric systems are the avenue for providing authentication and identification capabilities for digital services. D. Thakkar states in [6] that most biometric systems have been designed to follow a similar framework architecture that can be applied to various biometric modalities. First, at the interface between human and machine, a biometric sensor module is employed to collect the targeted biometric data, such as, images of human irises, human fingerprints, or timestamps of a user's keyboard pressings. After the targeted biometric data have been collected, they will be passed to a pre-processor module, which serves to clean up the biometric data by removing the random noise that was captured with the biometric trait. The cleaned-up biometric sample is then passed to a feature extractor module. This modules' intended purpose is to amplify the most useful features present in the cleaned-up biometric data sample. Many contemporary biometric systems employ machine learning (ML) techniques, such as artificial neural networks (ANNs), to perform the feature extraction function. The output of the feature extractor is a tensor of numeric values that represents the collected biometric sample, called the biometric template. As Figure 1 depicts, the template generator takes the output matrix and transforms it into a form amenable to the systems database/data structure architecture.

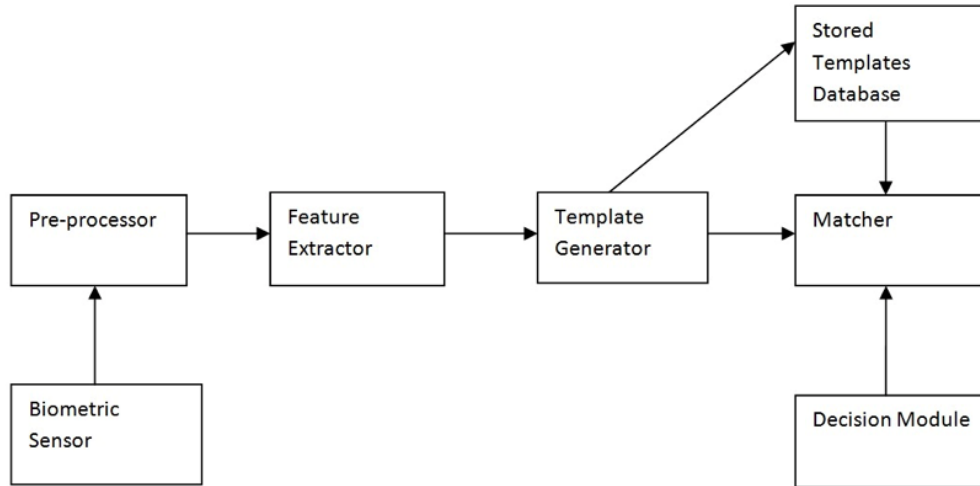


Figure 1. Generic Biometric System. Source: [6].

If the biometric sample is being collected from an individual for the first time, or if the sample is being used to update an association with an individual, the system is in enrollment operation mode. In this mode, the template, which was produced by the template generator, is stored in the systems' database, along with the identity of the individual who provided the sample.

D. Thakkar further highlights how a biometric system will be capable of performing authentication and identification tasks when it has enrolled templates in its database. Authentication is a one-to-one comparison between the template generated from the collection of a biometric sample of a particular individual and an enrolled template for that individual who claims to be. Identification is a one-to-many comparison between a generated template and all the enrolled templates in the systems' database. For either task, a matcher module is needed to produce a measure of similarity between a generated template and enrolled template(s). Lastly, a decision module is built either as a part of the matcher module, or as a standalone module. The decision module allows the biometric system administrators to implement the thresholds that will be used to determine if the obtained similarity score constitutes successful authentication, or a positive identification.

To measure the effectiveness of a biometric authentication system, several metrics are commonly employed. Biometric authentication is a binary classification problem. The

European Union's Biometrics and Evaluation Testing (BEAT) project defines False Acceptance Rate, or FAR, as "an empirical estimate of the probability (the percentage of times) at which the system incorrectly declares that a biometric sample belongs to the claimed identity when the sample actually belongs to a different subject (imposter)" [7]. Furthermore, security researchers Vandana and Kaur define FAR mathematically as the count of False Positives divided by the count of False Positives and True Negatives [8]. Another commonly reported metric in biometric systems is the True Acceptance Rate (TAR), which is defined as $1 - \text{FAR}$. Conversely, BEAT specifies False Reject Rate (FRR) as "an empirical estimate of the probability at which the system incorrectly rejects a claimed identity when the sample actually belongs to the subject (genuine user)" [7]. FRR is defined mathematically as the count of False Negatives divided by the count of False Negatives and True Positives [8]. Lastly, BEAT states that the rate where FAR is equal to FRR is known as the Equal Error Rate (EER) [7]. The EER is often used to determine the threshold of the scoring system; any matching score that falls below the threshold is considered to originate from an imposter distribution (and thus not authentic), and any matching score that falls at or above the threshold is considered to originate from a genuine distribution (and thus authentic). Threshold is a parameter of the biometric system that can either increase the amount of True Positives at the cost of increasing False Positives (by shifting the threshold to a lesser value) or decrease the amount of False Positives at the cost of also decreasing True Positives (by shifting the threshold to a greater value). Figure 2 is a graphical depiction from Vandana, and Kaur of the effect threshold has on the FRR and FAR.

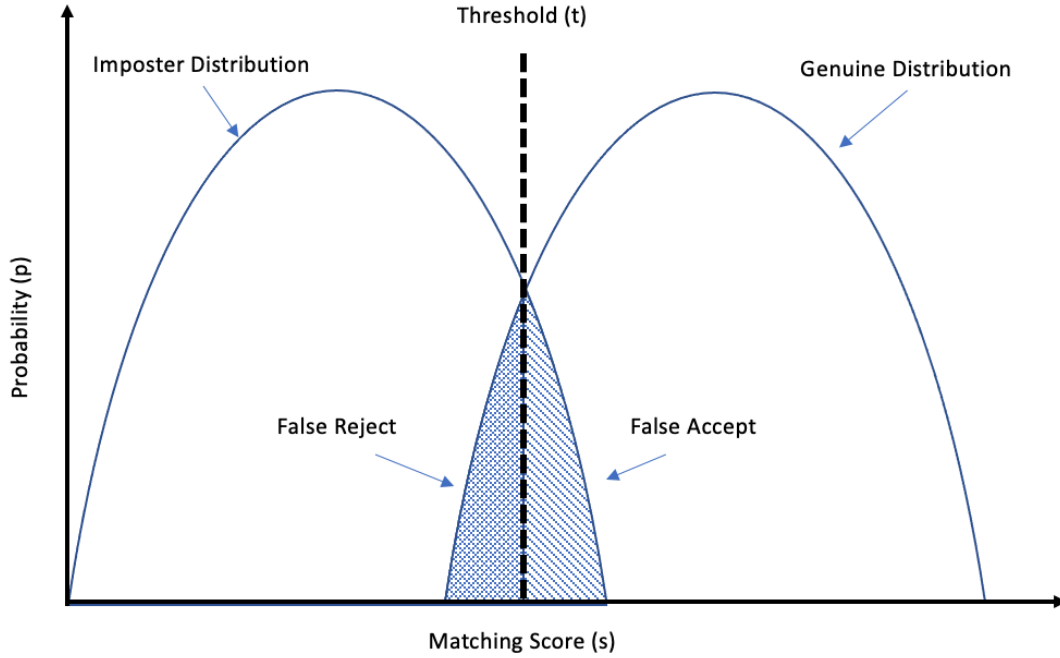


Figure 2. Imposter v Genuine Match Plot. Adapted from [8].

C. BIOMETRIC BASED KEY RELEASE SYSTEMS

As introduced in this chapter, [5] proposed biometric-based authentication for key release as an alternative solution to password-based key release. Uludag et al. describe in [5] that the basic idea behind a biometric-based key release system is to use biometric authentication to authenticate the user, and a generic cryptographic system to handle all other portions of the transaction, such as secure communications. These systems utilize a database to store a user's biometric template, username, and corresponding cryptographic key, which is then only released upon successful biometric authentication. The key is then only released upon successful biometric authentication [5]. For example, Bob would enroll his username and template along with a key to the database. When he wanted to get his key, he would use the biometric system to request his key, whereupon the system would generate a biometric template based on his biometrics, compare it to the one stored in the database under his username, and then release his key upon successful authentication (matching). Bob could then use his key to conduct transactions such as online shopping or signing documents on the internet. On the contrary, if Alice tried to obtain Bob's key by

impersonating his username, the template generated by Alice would not match the template stored under Bob's username, and the system would not release the key.

Uludag et al. describe in [5] that a biometric-based key release system has two characteristics; 1) The system needs access to a biometric template for matching and 2) Authentication and key release occur in two decoupled processes. They point out that there are a few significant issues with this method of key release. First, since the system needs a template to compare to during authentication, a template needs to be stored in the database during enrollment. The authors write that although this eliminates having to remember passwords, unlike passwords, templates cannot simply be changed once they are stolen. They state this is a dangerous practice because if the database were attacked and the template stolen, that template could now be used to impersonate the user on this application and potentially other applications that use the same authentication system [5]. Second, they describe that due to the decoupling of authentication and key release, the system is susceptible to Trojan horse attacks (e.g., attacks that flip bits on Authentication results to change rejects to accepts and releases the key to unwanted users and prevents the actual owner from retrieving their key) [5]. Just as it is important for traditional systems to maintain password integrity to protect the overall integrity of the system, in biometric-based key release systems, biometric template security is important. Biometric template protection is done in one of two general ways: completely altering the template or locking the template. Both solutions integrate biometric matching and cryptographic techniques and are known as Cancelable Biometrics and Biometric Encryption respectively.

1. Cancelable Biometrics

Cancelable Biometrics was first explored by Soutar et al. [9]. and falls under a larger category of Feature Transformations. As [10] describes, in this method the biometric system transforms the extracted features via some function during the enrollment process, and only this transformed template is stored in the database. Once it is time to identify or authenticate a user, the same function is used on a new extracted sample, and comparisons can be made to the stored sample. Since the stored template is a transformed version, if it is stolen, a different transform function can be applied to a new sample and the database

can be updated [10]. The original biometric template remains uncompromised, and the system remains intact. Feature transformations can be done in one of two general ways, bio hashing or salting and through noninvertible transforms.

a. Bio Hashing/Salting

As described by Arias in [11], password salts in cryptography are non-secret data which help prevent duplicate passwords from hashing to the same value and help increase the search space of offline attacks. He describes the salt as being, typically, randomly generated and assigned to a password on enrollment and appended to the end of the password. That appended password and salt combination is then hashed using a Hash Function. A password file is then used to store a combination of the hash and salt pair. To authenticate, the user must provide their password and the system will automatically retrieve the user's corresponding salt, hash the two components, and compare to the stored hash in the password file [11]. Jin et al. describes that, in a similar approach, Biometric Salting aims at transforming the biometric template through use of a salt for storing in a database [10]. They state however, that rather than randomly generating a salt and appending, the salt is generated by the user, either as a password or a random number. The salt is then blended with the biometric data through matrix transformations to generate the transformed template that is stored on the database. Since the transformation depends on an externally generated key, this version of template protection suffers from similar vulnerabilities as password-based systems [10]. An instance of biometric salting, BioHashing was introduced by Jin et al. BioHashing uses an inner product of the extracted template and a tokenized pseudo-random number matrix generated by the user to produce a coded template called the BioHash [12]. This version of template protection was examined in [10] with various modalities and nearly zero error rates but suffered during the stolen-token scenario in which a real user token is stolen and used by an imposter.

b. Noninvertible Transform:

In computer science, non-invertible functions, also called one-way functions, are functions which can be easily computable on any input but hard to invert on just the output. Jin et al. describes in [10] that non-invertible transforms utilize one-way functions to

completely transform the biometric template into a new form within the context of feature space. Ratha et al. discovered in [13] that a benefit of this type of template protection is that since the function does not directly interact with the biometrics, the function, known as the key, does not need to be kept secret. Ratha et al. also introduced three non-invertible transform functions known as the Cartesian, Polar, and Surface folding transformations and showed success with fingerprint data [13]. However, as mentioned in [13], all three transformations make it is possible for two minutiae from the original template to map to the same location on the new transformed template. Quan et al. argued in [14] that this would degrade the one-to-one invertible mapping and Shin et al. then proved in [15] that it could be inverted. Tulyakov et al. and Maiorana et al. continued the work using new schemes in [16] and [17] respectively.

c. Issues

Although [9], [10], [13], [16], and [17] all demonstrated progress in the field of Cancelable Biometrics, this method of key release suffers from the non-invertibility problem. As [5] describes, if the template transformation is non-invertible, there is high confidence that the original template is secured. However, as [5] also points out, due to the variability in biometric characteristics, when a new sample is taken by the biometric system for authentication matching, your error rate will increase. Conversely, if you allow your transformation to be invertible, your biometric system matcher will be more accurate, but your template will not be secure [5]. For these issues, we chose to avoid Cancelable Biometrics as our method for key release and instead used Biometric Encryption.

2. Biometric Encryption

Biometric Encryption (BE) was first introduced by G. Tomko et al. [18]. As described in [19], BE uses “encryption” of a digital key by either binding it to a user’s biometric template or by generating a key from the template. Upon proper authentication via a user’s biometric, the key is “decrypted” and released to the user [19]. One benefit of BE over Feature Transformations is that the user’s template does not have to be stored for the system to work [5]. As previously discussed, this lowers the threat of the template being stolen, regardless of whether it has been transformed or not. There are two general

approaches for BE: Key Binding, randomly generating a key and non-deterministically hiding it in the template, and Key Generation, deriving a key from the template.

a. Key Generation

Described by Uludag et al. in [5] and Cavoukian et al. [19], in Key Generation a random key is generated on enrollment so that neither the user nor anyone else knows the key. Once a biometric sample is taken, a biometric template is generated, and the key is hidden in the user's template through a secret non-deterministic bit replacement algorithm. Cavoukian et al. further points out in [19] that this algorithm is designed to account for variations in biometric samples and outputs a Biometric Encrypted Key (BE Key). Thakkar et al. [6]. and Cavoukian et al. [19]. detail how both the original key and the template are discarded and only the BE Key is stored in the database. Cavoukian et al. describe the verification process. For verification, a new template is generated from a new sample. The BE Key is then retrieved using helper data, such as username or other identifying information stored in the database. The new template and BE Key are then applied to a BE retrieval algorithm, which will "release" the original key upon proper authentication to the biometric system [19]. Cavoukian et al. then touch on how rather than release the key to the user, the key is then hashed, and the hash is released to the user, ensuring the original key never leaves the system. The released hash then goes on to serve as the cryptographic key, hence the term, key generation. The sample and template are once again thrown away and the BE Key is re-stored for future use [19].

b. Key Binding

Key Binding is described in [5], [6] and [19]. Uludag et al. detail how, in Key Binding, the same process as Key Generation occurs except the random key generated at the beginning of the process is a traditional cryptosystem Public/Private Key pair [5]. Uludag et al. [5]. and Cavoukian [19] demonstrate how the Private Key is then non-deterministically hidden in the template and released only upon proper biometric authentication. In this application, biometrics are an augmentation of passwords in traditional cryptosystems [5]. This form of Biometric Encryption is referred to as a Biometric Cryptosystem and is a focus of our research.

D. RELATED WORK

The field of biometric authentication is a well-established and researched field. There are two general types of biometric authentication, physical and behavioral.

1. Physical Authentication

Physical Biometric Authentication includes research in the fields of fingerprints, iris recognition, hand geometry, and facial recognition. Work in this area started as early as mid-1800s, when William Herschel and Francis Galton studied and implemented systems with fingerprints [20], and has been ongoing as recently as [21] with facial recognition. These systems are often expensive to implement due to the special equipment required to obtain the data samples and are highly user intrusive. For these reasons, they often fare lower in preference to traditional authentication systems.

2. Behavioral Authentication

Behavioral Authentication systems include those such as voice pattern recognition, body movements (gait, and posture), and device-based gestures (keystroke dynamics and cursor movements). Various efforts have been incorporated into improving the accuracy of these identifications including the integration of deep learning techniques. More recently, as described in [22], financial institutions have used keystroke biometrics to their advantage as a component in multi-factor authentication through third party providers. These providers are drawing the attention of tech industry giants, as evident by Google's commitment of \$7 million to fund the keystroke biometric startup company, TypingDNA, in 2019 [23]. Keystroke technologies continue to find implementation in authentication schemes across multiple domains, including web services such as banking and mail, databases, and the newest smartphones and handheld tablets [24]. Allied Market Research, a global market research firm, claimed that the global keystroke dynamics market will grow from \$129.8 million (in 2017) to \$754.9 million by 2025 (a rate of up to 25% per year) [25].

As [22] describes, literature is quick to point out, however, that keystroke dynamics cannot reliably be used as the sole factor in authentication schemes. Biometric attacks, such

as those that obtain an individual's keystroke timing information from logs inherent in online collaboration tools, have been shown to be effective in defeating keystroke authentication schemes employed by TypingDNA [22]. One solution to the biometric attack dilemma is to stack multiple modalities into the authentication scheme. As demonstrated in the paper *Actions Speak Louder Than (Pass)words: Passive Authentication of Smartphone Users via Deep Temporal Features*, a passive, multi-modal authentication model can achieve authentication accuracies above 97% in smartphones. By having each modality compromise a fraction of the overall authentication decision and combining user identification with device fingerprinting, the scheme will be more robust against targeted biometric attacks [26].

a. Enabling Technology

Recent developments in key technologies have enabled behavioral biometrics to reliably perform user authentication. One of those key technologies are Recurrent Neural Networks (RNNs). RNNs are a specialized form of ANNs that allow state information to persist through iterations of neural net calculations. For example, suppose a model needed to predict what word would likely occur next in the following sentence, "the fish swim in the..." The sequence of previous words in the sentence informs what the correct output of the model should be (in this case, the word could be sea, ocean, or any body of water). An online blog from AI expert Christopher Olah observes that a traditional ANN does not have the capability to preserve state information that would enable future predictions of sequential data [27]. RNNs were developed to provide this capability.

Olah explains that the implementation of a feedback loop in RNNs allows for the modeling of sequential data [27]. For example, suppose iteration t represents the RNN model's processing of an element of sequential input data during the current iteration of the loop, $t-1$ would have been the model's processing of the previous sequential element, and $t+1$ will be the model's processing of the next sequential element in the input data. By having the output of the neural network at iteration $t-1$ feed into the model during iteration t , the previous output will influence the way the model processes the current sequential element input.

Going back to the previous sentence example, a RNN will be able to use the previous processing of the words fish and swim to produce the correct output of sea or ocean. Due to the looped nature of the model architecture, RNNs have become pervasive in industry and academia for sequential data-heavy tasks, like natural language processing. Olah highlights, however, that RNNs have trouble dealing with sequence data that have high degrees of long-term dependencies between the elements. The feedback nature of a native RNN allows for the retention of recent information from the sequence data, but when the information needed for the model to make an accurate prediction is found a non-trivial distance away in the sequence from the current element being processed, it is more than likely that the necessary information is no longer accessible for the model. For example, a sentence taken from [27] highlights when the long-term dependencies can become problematic for a RNN: “I grew up in France... I speak fluent French,” where the ellipsis in the middle of the sentence represents an arbitrary number of additional words within the sentence. If a native RNN was tasked with predicting the last word of the sentence, there is a significant chance that the information related to the input “France” from the beginning of the sentence could not promulgate throughout all the iterations of the model. Without that piece of information, the model will have difficulty in producing the correct output of “French.”

To tackle the long-term dependency problem in RNNs, Olah’s blog specifies that a specialized form of RNN was developed, Long-Short-Term Memory (LSTM) networks. As a derivative of RNNs, LSTMs retain the feedback nature of the network. Additionally, LSTMs employ a cell-state mechanism that is separate from the output of the network, but also gets fed into the next iteration of the network. The cell-state mechanism enables careful regulation for what gets added or removed from the cell state. In the prior French sentence, common words, such as up or in, do not provide much value in the network’s prediction capability, and so that information will likely not be heavily added to the persistent cell state. The words of value, such as France, would be allowed to be added to the cell-state, and will be promulgated to future iterations of the network.

b. State of the Art in Keystroke Dynamics

LSTMs have been the backbone of several state-of-the-art algorithms that enable user authentication through keystroke dynamics. Acien et al. explain that keystroke authentication algorithms are typically categorized into two bins: free-text and fixed-text keystroke authentication algorithms [28]. Fixed-text algorithms, the authors highlight, are characterized by their utilization of pre-fixed sequences (sequences and sentences are used interchangeably in this context), such as a username and password. In authentication schemes, the keystroke sequences for the enrolled sample (also known as gallery sample) and test sample (also known as a challenge sample) will be the same. Furthermore, accuracies north of 95% are common in fixed-text authentication algorithms. In contrast, Acien et al. discuss how free-text keystroke authentication algorithms utilize variable length sequences. The gallery and corresponding test samples do not have to be from the same sequence. For example, the sequences extracted from a user claiming to be an enrolled user typing an email can be compared to gallery samples from that claimed enrolled user to determine if the test sample was a genuine match. Free-text algorithms have typically not performed as well as fixed-text algorithms [28].

Acien et al. [28]. introduced TypeNet, a keystroke dynamic authentication and identification algorithm trained with free-text keystroke sequences from over 100,000 participants. The algorithm sought to demonstrate that high free-text authentication and identification accuracies could be achieved and scaled to hundreds of thousands of users. Acien et al. first describe how the underlying architecture includes two LSTM layers, intertwined with various regularization layers. Next, the input to the LSTM models is fixed to sequences of character length, M . Due to the free-text nature of the algorithm, the raw input sequences may be less than or greater than M . If sequences are greater than M , then they are truncated down to M . Conversely, if they are less than M , the sequences are padded with zeros up to M . Additionally, to ensure the zeros are not used in the models' loss calculation, the authors introduce a masking layer immediately following the input layer. Lastly, the output from the model is a 1×128 array (also known as an embedding vector) containing the extracted features from the keystroke sequences and is used in the

authentication portion of the algorithm. Figure 3 shows a graphical depiction of the architecture.

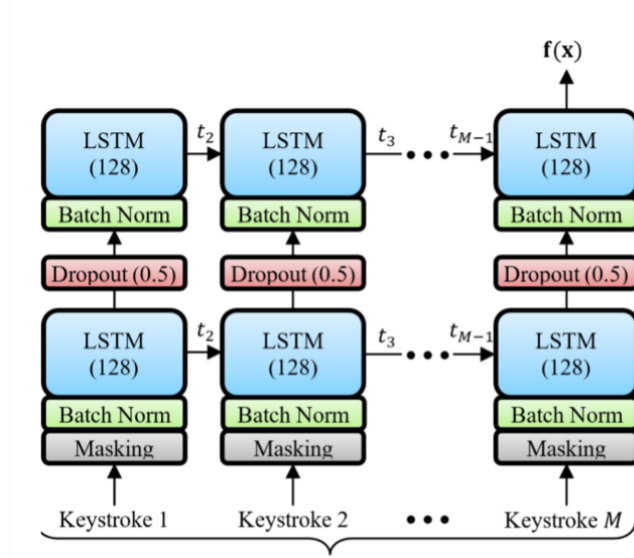


Figure 3. TypeNet’s LSTM Architecture. Source: [28].

The output embedding vector is matched via Euclidean distance to gallery embedding vectors, resulting in a test score [28]. The authors explain that the test score dictates whether the challenge results in a genuine match (test embedding vector matches the gallery embedding vector) or imposter match (test embedding vector does not match the gallery embedding vector). Genuine matches are considered authenticated, while imposter matches are not authenticated. The authors were able to achieve a 1.2% EER for the authentication protocol on a keystroke dataset obtained from Desktop computers for 1,000 enrolled users. Scaling up the users to a maximum of 100k, experienced a small but acceptable increase in EER. Specifically, after 10K users, the EER stabilized. The performance achieved by TypeNet was better than three other state-of-the-art free-text keystroke authentication algorithms that employed Partially Observable Hidden Markov Models (POHMM), support vector machines (SVM), or Convolutional Neural Networks (CNN) paired with RNNs [28]. Keystroke dynamics can also be deployed in multi-modal biometric authentication schemes. Ross et al. were able to show that smart phones can

accurately authenticate their users via the fusion of multiple modalities, to include keystroke dynamics; TAR and FAR north of 99% are possible in the mobile phone domain when 8 modalities are fused [26].

E. KERBEROS

Another alternative to password release systems is Kerberos, shown in Figure 4. SSO solutions for secure key release architectures are rooted in the Kerberos protocol. Kerberos was introduced by Steiner et al. at MIT in 1988 to provide a trusted third-party authentication service on an open network [29]. The authors explain that the protocol was designed to be secure, reliable, transparent, and scalable. To achieve these goals, from a high-level perspective, Kerberos keeps a database that associates the Kerberos clients (also called principals) with a cryptographic symmetric key. Each principal has its own symmetric key (secret key) that is only known to itself and the Kerberos Key Distribution Center (KDC). Principals can be user clients attempting to access a network service and/or the network service itself. While there are asymmetric implementations of Kerberos, the bulk of implementations used in practice are based on cryptographic symmetric keys [29]. The secret keys are negotiated at registration between the KDC and the principals; secret keys for user principals are encrypted passwords. Kerberos will craft messages which authenticate principals with each other and will generate and distribute temporary cryptographic session keys that allows for a secure channel between participants during authentication [29].

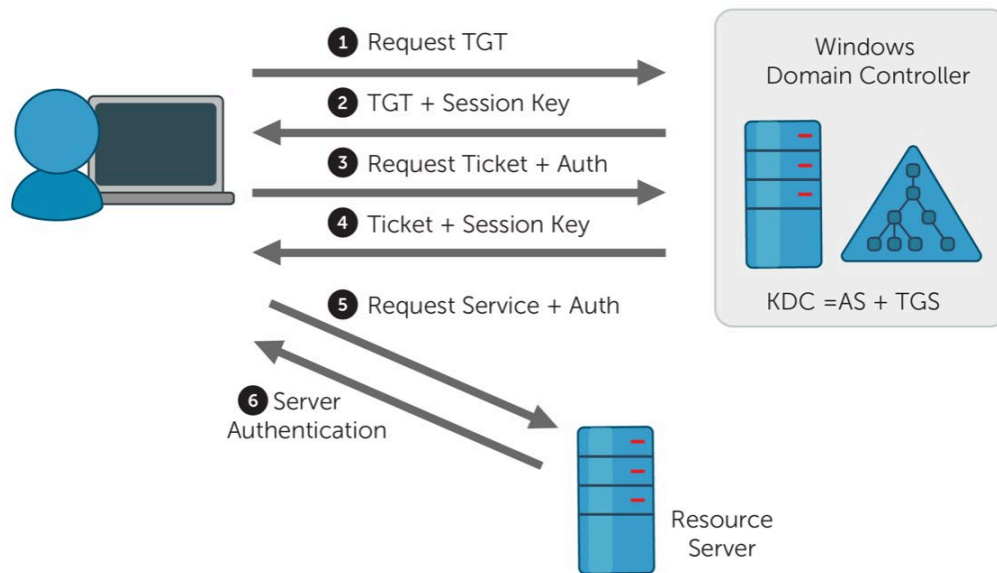


Figure 4. Kerberos Authentication Protocol. Source: [30].

To allow access to a network service, a user must establish their identity. Broadly, Steiner et al. illustrates, Kerberos provides the authentication service by providing a user with a ticket that can be presented to a network service (and vice versa when the user requires authentication of the server) [29]. First, the actual authentication process can be broken into three phases. In the first phase, a user obtains credentials to be used to request access to other services. Credentials can be tickets or authenticators. Tickets are authentication messages drafted by the authentication server portion of the KDC and are used to pass the identity of a user to the service being requested. The ticket itself will contain the server's ID, the user's ID, the IP address of the user, a timestamp, a lifetime, and a session key for the two principals' to securely authenticate with. The ticket will be encrypted with the server's secret key and can be used multiple times between the two principals, until the ticket expires (as indicated in the lifetime field of the ticket). The user also drafts their own authenticator credentials. Authenticators contain the user's ID, IP address, and a timestamp, and are encrypted with the same session key that is contained in ticket. In the first phase, the user receives the session key along with the ticket from the authentication server. That message is encrypted in the requesting principal's secret key by the authentication server. Secret keys are only known to the KDC and the principal, thus

the user will be proving its identity to the authentication server by successfully decrypting the credentials message. The service the ticket was issued for is known as the Ticket Granting Server (TGS), which is another component of the KDC. The user uses TGS to request authentication for other specific servers on the network [29].

In the second phase of Steiner et al.'s authentication process, the user requests authentication for a specific service [29]. From phase one, the user has a ticket, known as the Ticket Granting Ticket (TGT), that allows the user to request services for the lifespan of the ticket. In SSO implementations, the TGT is often used as the credential that allows a user access to a resource for the lifetime of the ticket. From this point, the authors assert that the TGT will be sent from the user to the TGS, along with the server's ID, and the user's Authenticator (encrypted with the session key between the user and the TGS). Next, the TGS will decrypt the ticket, obtain the session key, decrypt the authenticator, and ensure the information on the ticket and the authenticator match and the lifetime is still within bounds. If everything checks out, the TGS will send the user back a new session key that will be used between the end server and the user, a ticket for the end server (encrypted with the end server's secret key) that has the same structure as the ticket generated in phase one. Lastly, this entire message is encrypted with the session key between the TGS and the user.

For the third and final phase of the authentication process, the user presents the credentials received from the TGS to the end server [29]. This process mirrors the process from phase 2, just with the credentials for authentication between the user and the end server. A new authenticator will be generated by the client. To defend against replay attacks, the end server will compare the timestamps between the Authenticator and the Ticket; if the request is too far in the past/future, the request will be dropped [29]. If everything is valid, the user has been successfully authenticated to the server and may proceed with its requests to the server. Should the user require mutual authentication from the server, the server can increment the timestamp from the Authenticator by 1, encrypt it in the session key, and send back to the user [29].

Kerberos, however, is not without its share of security concerns. Windows uses Kerberos as its default backend authentication technology; thus, the ubiquitous nature of

the protocol makes for a lucrative target for malicious actors (implementations of Kerberos can also be found in Apple OS, FreeBSD, Unix and Linux distributions). Varonis Systems, a cybersecurity software company, described 6 different known attacks against Kerberos: Pass-the-ticket, Golden Ticket, Silver Ticket, Credential stuffing/Brute force attacks against the user password, Encryption Downgrade with Skeleton Key Malware, and a DCShadow attack [31]. Of the attacks listed, the Golden Ticket attack is perhaps the most egregious. On a Windows domain that is controlled by Domain Controller, a Kerberos authentication ticket for the hidden account that encrypts all the authentication tickets for the domain is known as a Golden Ticket [32]. A Golden Ticket allows whoever holds the golden ticket to “arbitrarily generate Kerberos TGT tickets for any user of the target domain. Therefore, it can be used to impersonate anybody” [33]. The likelihood of a Golden Ticket attack, however, is low. Varonis states, “... a somewhat improbable chain of events. A hacker would have to login to the domain controller machine and gain elevated permissions and find the NTLM hash of krbtgt, a special user set up by Kerberos [the previously mentioned hidden account]. That last part, by the way, provides the special key used to encrypt all the TGTs” [34]. A more likely attack against Kerberos are attacks against weak user passwords.

Apart from malicious attacks, RFC4120 (the specification of Kerberos version 5) highlights several other security concerns of the latest iteration of the Kerberos service. First, the reference states, “by itself, Kerberos does not provide authorization; applications should not accept the issuance of a service ticket by the Kerberos server as granting authority to use the service” [35]. Furthermore, Kerberos does not solve Denial of Service (DOS) attacks [35]. An unfortunate side effect of 3rd party authentication is the centralization of all secrets required for authentication; the compromise of the Kerberos KDC will compromise any principal that is administrated by the compromised KDC [35].

Previous versions of Kerberos utilized DES for their symmetric cryptography functions. DES is no longer considered secure, and RFC4120 now requires all implementations of Kerberos to support at least the AES256 algorithm [35]. Backwards compatibility should be implemented for AES128 and DES legacy implementations. The RFC also reiterates that “principles must keep their secret keys secret. If an intruder

somehow steals a principal's key, it will be able to masquerade as that principal or impersonate any server to the legitimate principal" [35].

F. SUMMARY

As demonstrated in [23] the field of behavioral biometrics has shown tremendous improvement as the enabling technologies required for them become refined and available. Consequently, biometric-based key release systems, which rely on biometric authentication [1], have also seen improvement and have the potential to address the flaws presented by password-based release systems describe in [1]. Traditional SSO implementations, such as Kerberos, provide a means to authenticate network participants on an untrusted network, but have been shown to have security flaws [29], and do not provide a means to provide continuous authentication for users and services. Our research differs from those mentioned in this chapter in that we: 1) build on [23] to implement a multimodal biometric authentication system which 2) serves as the foundation for a BE key release system described in [1], [2] and [14] that can 3) be used to continually authenticate a user while minimizing user intrusion.

THIS PAGE INTENTIONALLY LEFT BLANK

III. ERINYES PROTOCOL INTRODUCTION

Our previous chapter introduced two alternatives to password based released systems, Biometric Based Key Release Systems and Kerberos. As previously summarized, we propose an approach that combines aspects of both alternatives to continually authenticate a user. Figure 5 demonstrates our proposal, which we have named the Erinyes Protocol, which utilizes a client-server model to continuously authenticate a user via biometrics and incorporates biometric encryption and Public Key Infrastructure (PKI) for secure communications. Both the client and server portions of the protocol have their own respective software implementations.

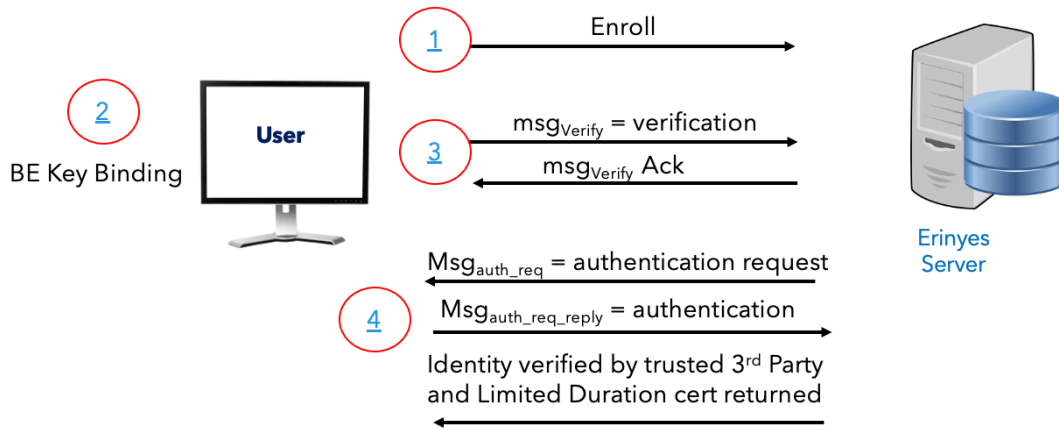


Figure 5. Erinyes Protocol

The Erinyes protocol, which will be discussed in further detail in section C, begins with enrollment (see Figure 5). During enrollment, biometric features are extracted from user's keyboard timing and mouse cursor events by using LSTMs to produce 10 combined feature embeddings that are added to a user profile in the Erinyes server. Next, the Erinyes client creates an asymmetric cryptographic key pair and binds it to the user's biometrics using a fuzzy vault to create a BE key. The original public key is also hashed for server-side verification. The Erinyes client then sends an encrypted message to the Erinyes server to verify its public key. The message contains the original public key's digest, the

Biometrically Encrypted Key, the client's user ID and the client's biometrics. The server-side of the protocol decrypts the message and attempts to verify the user in the following manner: 1) the user ID and IP from the message is matched to the associated user ID and IP in the user profile, 2) the client's biometrics are run through the server's LSTMs, and the resulting embeddings are compared to the client's enrolled gallery samples. If the Euclidean distance is less than a configurable threshold, then 3) the client's public key is retrieved by running the client's biometrics through the biometric decoding process on the BE key and 4) the retrieved public key is hashed, and the resulting digest is compared to the digest sent in the message. If verification succeeds, the client's retrieved public key replaces the client's stored embeddings in their user profile. The server then acknowledges successful verification to the client who can discard their BE public key and only store the original cryptographic public key for communications.

To provide continuous authentication, the server issues limited duration x.509 certificates. Upon expiration of the authentication timer, the server sends an encrypted authentication request to the client. The client is then able to decrypt and validate the digital signature of the authentication request by releasing its cryptographic private key through a BE retrieval process that requires matching biometrics. If decryption is successful, the client replies to the server with an encrypted message. Lastly, if the digital signature from the client is valid, the server re-issues a new x.509 certificate. The new certificate is valid from the time of issuance through time of issuance + the authentication timer value. Upon authentication timer expiration, the issued certificate is no longer valid, and the Erinyes server will send a new authentication request to the client.

Through the "silent" incorporation of biometrics as an authentication method, we minimize user intrusion while maximizing user authentication. Our protocol is aided by two key enabling technologies, Neural Networks and Biometric Encryption, which were introduced in the previous chapter and will be expanded on in this chapter. Both technologies help create a framework for our four-phase protocol which covers the remainder of this chapter.

A. NEURAL NETWORKS

The Erinyes Protocol is reliant on ANNs to produce rich temporal biometric feature embeddings that allow a user to be authenticated continuously. Acien et al. [28]. demonstrated the ability of an LSTM model to accurately authenticate users based on the timing between the users' keystrokes. Furthermore, [26] determined that adding multiple modalities to the authentication scheme increases the accuracy of the authentication process. For this reason, the Erinyes Protocol is comprised of two modalities to make its authentication decisions: keystroke timings and mouse cursor movements. Although our research focused on only these two modalities, any biometric feature that can be represented by a 1-dimensional feature embedding can continuously and non-invasively be used in the protocol. Each modality has an LSTM that is trained off a corresponding keystroke or mouse cursor movement dataset. Both LSTMs from our research mirror the architecture introduced in [28].

1. Keystroke LSTM

The Erinyes keystroke LSTM is trained off a dataset produced using the same input pipeline as in [28]. Acien et al. [28]. created their dataset using data produced by Dhakal et al. from Aalto University, which can be found in [36]. The Dhakal et al. dataset contains over 5GB of keystroke data collected from 168,000 participants utilizing a desktop computer keyboard [36]. To acquire data, the researchers in [36] created an online application that recorded press and release key event timings with millisecond precision and employed controlled free-text data collection methods where random English sentences were selected from a set of 1,525 examples. Participants were asked to memorize the randomly selected sentences and then quickly type them [36]. The sentences contained at least 3 words, each with a maximum of 70 characters. Participants in the Dhakal et al. study completed 15 sessions, where each session was a memorized and typed sentence (also called a sequence) [36]. The resulting dataset consists of the key code of the pressed key (in ASCII), the key press time, and the key release time (both in milliseconds) [36].

The dataset in [28] was created by extracting temporal features from the Dhakal et al. dataset and adding a fifth feature corresponding to the key code. Figure 6 demonstrates

the four temporal features that the authors in [28] were able to extract from the raw keystroke data.

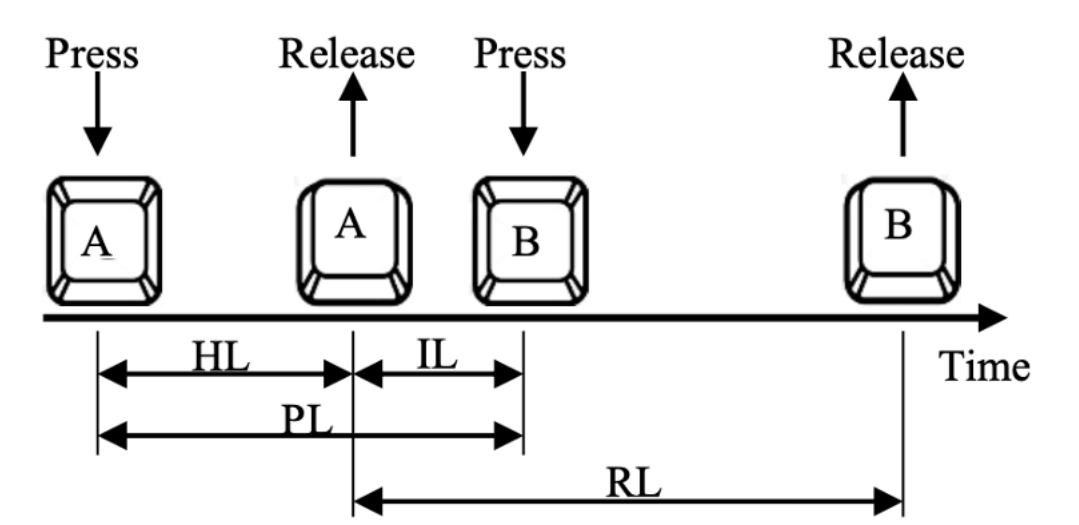


Figure 6. Extracted Temporal Features from Keystroke Dataset.
Source: [28].

The first, hold latency (HL), is a measure of how long a key was held down. The second, inter key latency (IL), is the time difference between releasing a key and pressing the next key in the sequence. Next, press latency (PL), is the time delta between two consecutive key presses. Lastly, release latency (RL) is the time delta between consecutive key releases. After producing temporal features, the authors in [28] normalized their data. The authors in [28] note that, “normalization is important so that the activation values of neurons in the input layer of the network do not saturate (i.e., all close to 1).” To normalize the four extracted features and the key code, the extracted features are converted to seconds (as it is unlikely that greater than 1 second occurs for any of the features) and the key codes are divided by 255 (the number of ASCII code integers) [28]. The result for both [28] and Erinyes is that for each raw keystroke data sequence that is processed through the input pipeline, the corresponding output is a feature matrix of size $N \times 5$, where N is the number of keystrokes in the sequence and 5 corresponds to the normalized features. Acien et al. [28]. go on to explain that training RNNs using standard backpropagation in the time

dimension requires the constraint that we standardize the time dimension, in this case the number of keystrokes. We chose to standardize the character lengths to sequences of size 70 by zero padding when the sequences were <70 and truncating when they were >70 . This results in the input layer to our RNN (LSTM) model as 70×5 feature matrices.

Figure 7 is a model summary provided by Google’s TensorFlow library that depicts the constituent layers of the Erinyes keystroke dynamic LSTM model. Although Figure 7 largely mirrors the graphical depiction of the TypeNet model introduced in Figure 3, it differs in the addition of the dense and L2 normalization layers (lambda layer). These differences reflected the model presented in [37] and allow for proper input format into a triplet loss layer shown Figure 7. TensorFlow identifies triplet loss as, “a loss function that trains a neural network to closely embed features of the same class while maximizing the distance between embeddings of different classes. To do this an anchor is chosen along with one negative and one positive sample” [37]. Figure 8 illustrates how the authors in [28] display in what way an LSTM architecture can be purposed to utilize triplet loss.

```
Model: "RNN_Model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 70, 5)]	0	
masking (Masking)	(None, 70, 5)	0	input_1[0][0]
lstm (LSTM)	(None, 70, 128)	68608	masking[0][0]
dropout (Dropout)	multiple	0	lstm[0][0] lstm_1[0][0]
batch_normalization (BatchNorma	(None, 70, 128)	512	dropout[0][0]
lstm_1 (LSTM)	(None, 128)	131584	batch_normalization[0][0]
dense (Dense)	(None, 128)	16512	dropout[1][0]
lambda (Lambda)	(None, 128)	0	dense[0][0]

```

Total params: 217,216
Trainable params: 216,960
Non-trainable params: 256

```

Figure 7. Erinyes Keystroke LSTM Layers

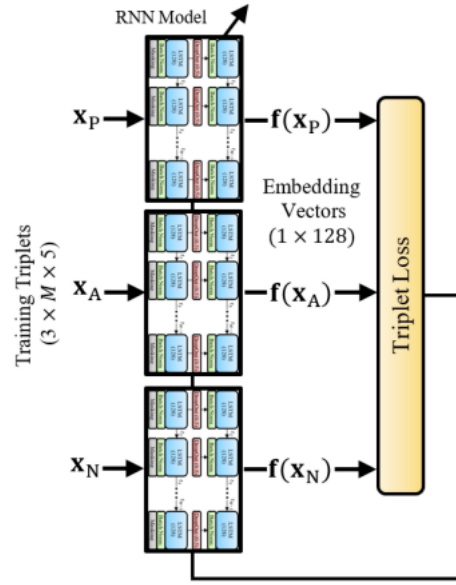


Figure 8. Triple Loss Architecture. Source: [28].

First, three processed input sequences are provided to three instances of the same LSTM model. One of the input sequences is the anchor sequence, which is a sequence from a user. Another input sequence, called the positive sample, is selected from the same user. The last input sequence, called the negative sample, is a sequence from a different user than that of the anchor. The Euclidean distance is then measured from the square of the absolute value of the distance between the output embedding vector of the anchor and the output embedding vector of the positive sample, minus the square of the absolute value of the distance between the output embedding vector of the anchor and the output embedding vector of the negative sample [37].

The mathematical depiction of the Euclidean distance function is provided in Figure 9. Throughout the training process, the goal of the model is to produce embedding vectors that result in the lowest possible loss value. Once trained, the model is capable of producing feature embeddings that are a numerical representation of the keystroke biometric trait of an individual.

The loss function is described as a Euclidean distance function:

$$\mathcal{L}(A, P, N) = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0)$$

Figure 9. Loss Function for Triplet Loss. Source: [28].

2. Mouse Cursor LSTM

The LSTM architecture is not limited to processing only keystroke biometrics. The same model architecture can be applied to multiple modality datasets and is applied to a mouse cursor movement dataset introduced by Chuda et al. in [38]. The dataset described in [38] contains mouse events from 100 users on a web application, where the mouse events collected were cursor movements (horizontal and vertical coordinates), button clicks, and button releases. From the mouse events, Chuda et al. calculated discrete point and click actions by dividing the sequence of cursor movement events by clicks or pauses. The processed point and click actions were then associated with the coordinates and timestamps of the actions. Each user had at least 100 valid actions [38].

Chuda et al. extracted 10 features from the processed point and click actions: click duration, pause before click, pause after click, curve length, standard deviation of velocity, standard deviation of curvature, jitter (ratio of smoothed and raw movement lengths), direction, angle of curvature, and the curvature distance [38]. The LSTM model for the mouse cursor movement modality in the Erinyes protocol is trained with these extracted sequences of point and click actions for each user.

The feature embeddings produced by the keystroke and mouse cursor movement LSTMs models are then used to authenticate enrolled users in the Erinyes protocol. Since the authentication process only requires a feature vector for each modality, the protocol can be extended to include any biometric modality that can be represented by a numeric feature vector.

B. BIOMETRIC ENCRYPTION

The second enabling technology fundamental for Erinyes' framework is Biometric Encryption (BE). The origins and functionality of BE were covered in the previous chapter; however, for the purposes of our research, there are a few items we will expand upon in this section. First, Erinyes uses *BE Key Binding*, which requires three main items to properly work: 1) A biometric template, 2) a random key, and 3) an algorithm(s) that non-deterministically hides the key in the template and retrieves it when presented the correct biometric template. To produce the biometric template needed for BE, Erinyes uses both the keystroke and mouse cursor LSTM's presented in the previous section. From a user's keystroke data, a resulting 1×128 embedding is formed. A second 1×128 embedding is then generated for the mouse cursor data. The two embeddings are concatenated resulting in a 1×256 embedding.

Figure 10 is an overview of the process Erinyes uses for BE Key Binding, known as Fuzzy Vault. Fuzzy Vault was first introduced in [39] and has since been expanded in other research, including [40] and [41]. As described in [40], Fuzzy Vault is meant to work with biometric features which are represented as unordered sets. The process of creating a Fuzzy Vault consists of creating a secret key and then using the key as the coefficients of a polynomial onto which a biometric template is projected. The biometric template in [40] is produced using fingerprints and in [41] using iris scans. The projected points create a *vault*, which is then augmented with randomly generated chaff points. Since there are variations in a user's biometrics, [40] describes how the generated templates may create variations, and, thus, the process is known as *fuzzy*. To try and remediate this issue, [40] explains how error correcting codes (ECC) can be utilized. As an example, during the retrieval process, a user produces a new biometric template by presenting their biometrics. If the new template matches the original, then enough points will overlap with points in the vault. A polynomial close to the original polynomial can then be extrapolated and a correction code can be applied to produce the exact polynomial and, thus, the coefficients representing the secret key.

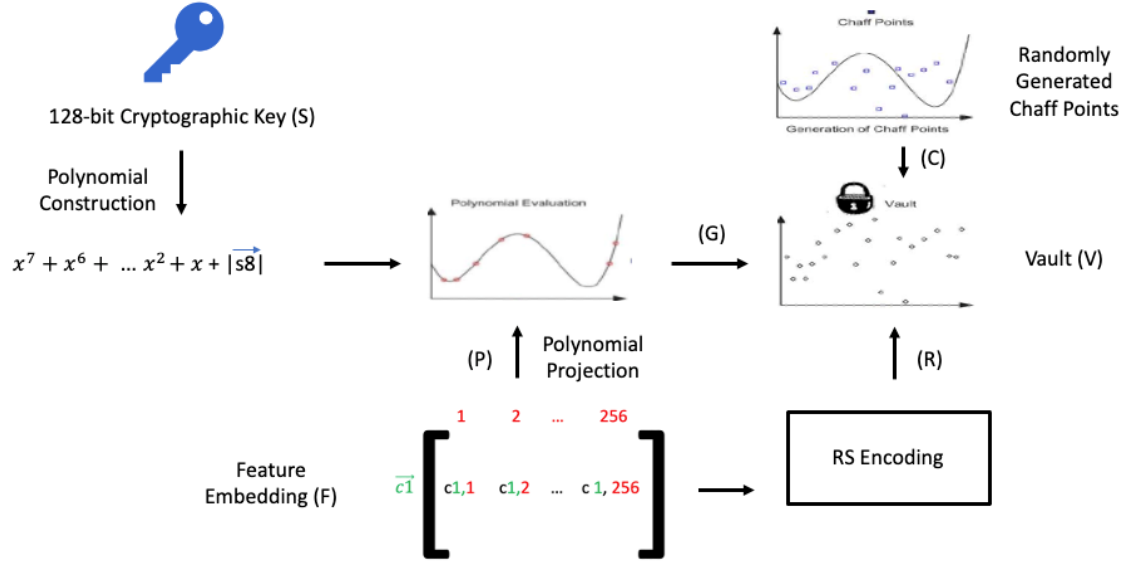


Figure 10. Fuzzy Vault Process.
Adapted from [39], [40], [41], and [42].

Since Erinyes uses time dependent data such as keystroke typing patterns, the biometric features can be considered ordered features, Wu et al. proposed an adaptation to Fuzzy Vault in [42] for working with ordered features. Erinyes follows this adaptation in developing and extracting points from fuzzy vault. First, the Python cryptography package [43], which depends on the openssl library, is used to produce corresponding pairs of 128-bit RSA public and private keys that are ultimately used for secure communications. The 128-bit keys are temporarily placed in a 8 x 16 matrix, and each row of the matrix is treated as an individual vector to produce the coefficients (magnitude of each vector) of a polynomial of degree 7 (number of rows - 1). The 1 x 256 embedding produced from the user's data is then projected onto the polynomial by using each value as the x value in the polynomial. The resulting 256 points are then stored in a vault alongside randomly generated chaff points. The chaff points are generated by creating random (x,y) pairings in the range of our genuine points. The random (x,y) values are chosen following two rules: 1) the x value of the chaffing point cannot be one of our embeddings, and 2) the y value of the chaff point cannot equal the projection of the x point onto the polynomial. This complete vault is then stored as a series of (x,y) coordinates in an array, forming the helper

data that is referred to as a BE Key in Erinyes. Similar to [40],[41], and [42], to retrieve a key, a user presents Erinyes the BE key (vault) and their biometrics. Erinyes uses the new embedding to filter out chaff points. We then apply Lagrange interpolation, similar to [40], [41], and [42] to produce a polynomial. Provided sufficient points overlap between the original polynomial and the new projection, we then apply an error correcting code to retrieve the original key.

C. PROTOCOL FRAMEWORK

The two technologies described in the previous sections allow for the implementation of Erinyes as seen in Figure 5. Erinyes Protocol is a cumulation of 4 phases: *enrollment*, *BE Key Binding*, *verification* and *authentication* which are detailed in the following subsections. To properly function, the Erinyes protocol makes the following assumptions during its execution: 1) the Erinyes server is a self-signing root Certificate Authority; 2) behavioral biometrics are unique to individuals; and 3) there is a continuous connection between the Erinyes client and server, and the Erinyes client software can collect user keystrokes and mouse cursor movements as needed.

1. Security

Due to the potential for communications between the client and server to be compromised, Erinyes employs an encryption security feature to conduct transactions between the client and server. Figure 11 demonstrates the encryption process utilized by Erinyes.

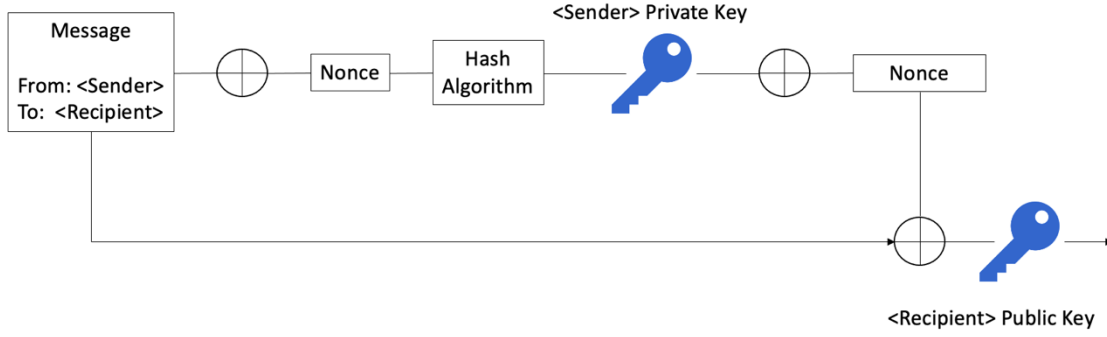


Figure 11. Erinyes Encryption Process.

Throughout all phases, cryptographic signatures are specified as follows: the message being signed, in its unencrypted form, is appended with a *To: <Recipient>* where *<Recipient>* will either be the Erinyes server or the Erinyes client. The appended message is then hashed. The resulting digest is appended with a nonce, then encrypted with the sender’s private key. The protected digest will ensure that the recipient of the message can detect any modifications, accidental or malicious, to the message. Message authenticity is achieved through the usage of the sender’s private key and the use of a protected nonce. Together, these measures provide integrity and a data-centric protection against replay attacks. Additionally, encryption of messages in the Erinyes protocol are specified as follows: the digital signature is appended to the corresponding original, unencrypted message, and then that entire unit is encrypted with the recipient’s public key.

2. Enrollment

The Erinyes Protocol starts with an Enrollment Phase as shown in Figure 12. During enrollment, rich feature embeddings produced by the LSTM models are associated with a User Profile that resides on the Erinyes server. The Erinyes server establishes User Profiles that each contain a UserID, IP addresses the user might use, and biometric feature embeddings extracted from a user’s raw biometric samples. As reported in [28], the best authentication results for the keystroke modality are achieved by enrolling 10 feature embeddings per user. The Erinyes server takes a similar approach. For each user, feature embeddings from keystrokes and mouse cursor movements are generated with LSTM

models. The keystroke feature embeddings are concatenated with the mouse cursor movement feature embeddings, resulting in a single combined feature embedding that represents one enrolled sample for a user. Ultimately, 10 combined feature embeddings are produced and added to the User Profile.

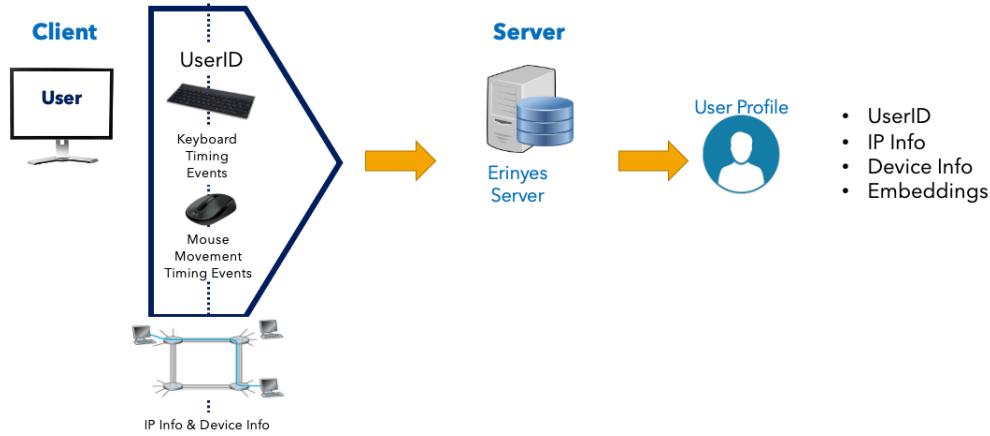


Figure 12. Erinyes Enrollment.

Ideally, enrollment of a user would be a strictly supervised affair, where the administrators of the Erinyes protocol deployment would have oversight of the complete enrollment process to ensure that the feature embeddings enrolled for a particular user legitimately came from that user. Thus, enrollment done local to the deployed Erinyes environment constitutes the lowest risk method of enrollment. Remote enrollment is possible, since the required LSTM models are deployed both on the client and server Erinyes software, but likely requires other technologic security mechanisms, like a VPN, to ensure the trustworthiness of the enrollments.

3. BE Key Binding

Next, as shown in Figures 13–14 and discussed in the previous Biometric Encryption section of this chapter, the Erinyes client generates an asymmetric cryptographic key pair and binds newly obtained biometric feature embeddings to the public and private keys to produce BE Private and Public Keys. The client’s generated public key is also hashed and verified by the Erinyes server and the verification phase.

Once BE keys are produced, the user's embedding and the original asymmetric keys are discarded. Only the BE Keys are stored as helper data to retrieve the original keys hidden amongst the vault.

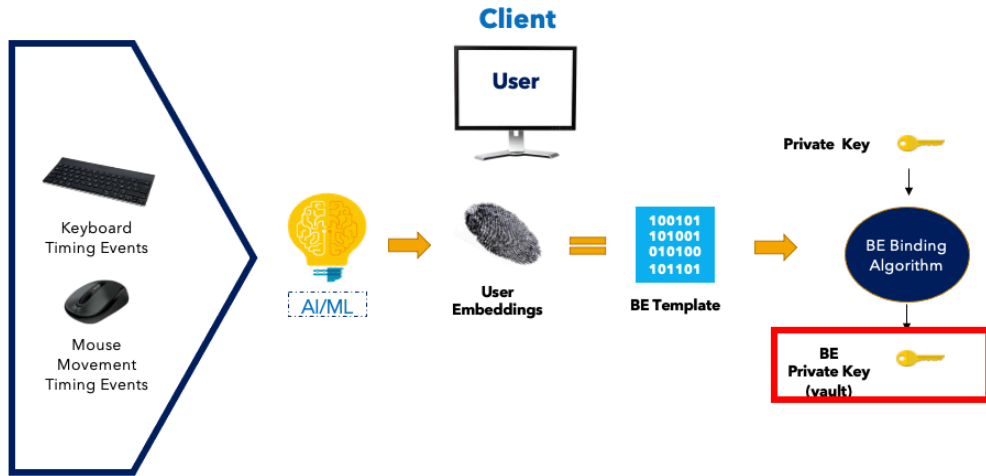


Figure 13. Private Key Binding.

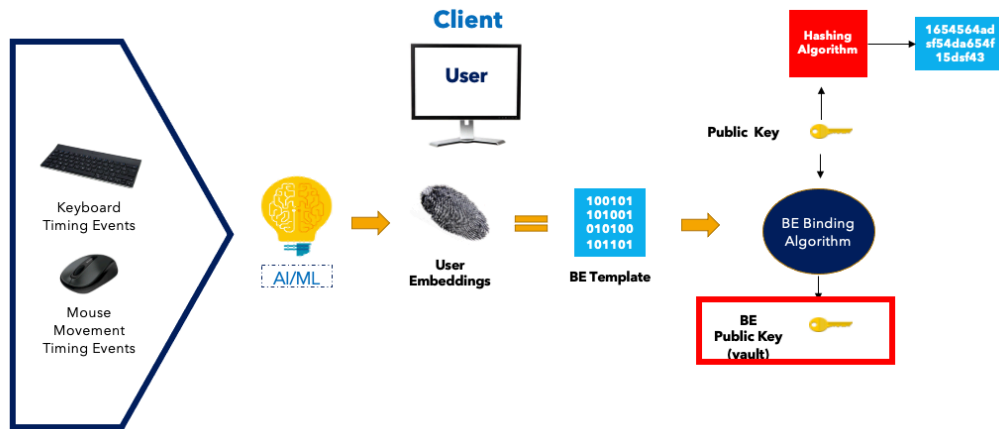


Figure 14. Public Key Binding.

4. Verification

In the next phase of the Erinyes protocol, the public key of the user/client is verified as genuine by the Erinyes server. Recall from the previous phase that the user's generated

public key was hashed and then encrypted with keystroke and mouse feature embeddings extracted by LSTM models on the client machine.

As shown in Figure 15, to begin the verification phase, the user crafts a message, hereafter referred to as *msgVerify*. *msgVerify* contains a *msgVerify* tag, the UserID (established by the Erinyes server during the enrollment phase), recent raw biometric samples (keystrokes and mouse cursor movements) collected from the user's system, the BE Public Key, and the hash of the public key. Once crafted on the user's machine, *msgVerify* is encrypted with the Erinyes server's public key, as per the Erinyes message encryption specifications. To achieve integrity, *msgVerify* is digitally signed in a manner consistent with the Erinyes Protocol specifications presented in the introduction to this chapter. *msgVerify* is then sent over to the Erinyes server, and the recent biometric samples are deleted from the client's machine. Figure 15 details the first part of the verification phase from the client's point of view.

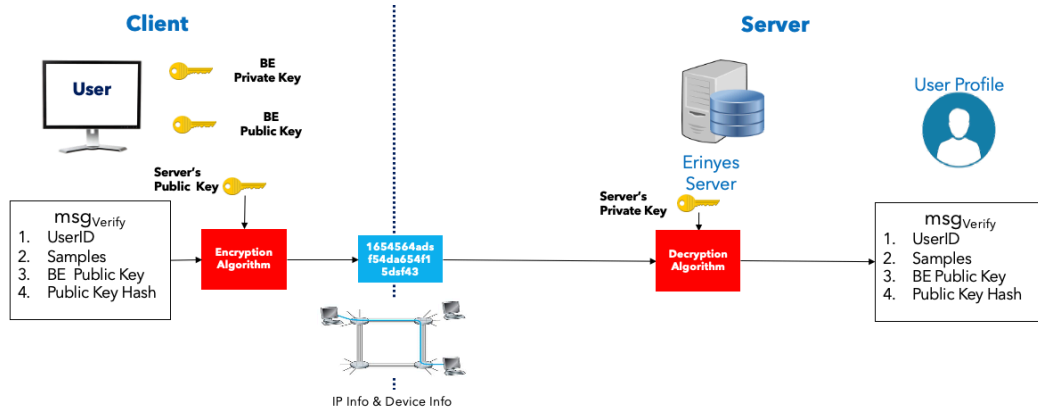


Figure 15. Erinyes Client-Side Verification Process.

As shown in Figure 16, when the Erinyes server receives *msgVerify* and parses the *msgVerify* tag, it retrieves the source IP of the sender and searches for the corresponding client's user profile. Then, using its private key, the Erinyes server decrypts *msgVerify* and begins verifying the contents of the message. First, the UserID retrieved from *msgVerify* is compared to the UserID associated with the user profile. Next, the raw biometric samples are extracted from the message, and processed through the Erinyes server's LSTM models.

Of note, the LSTM models deployed on the server are identical to the LSTM models deployed on the client machines. The resulting feature embeddings (one for the keystroke data, and one for the mouse cursor data) are concatenated and then compared to the concatenated enrolled embeddings associated with the user profile. Euclidean distances between the enrolled resulting feature embeddings are averaged across all the enrolled embeddings and compared to a threshold, *distance_thresh*. If the Euclidean distance is less than or equal to *distance_thresh*, then the samples are determined to have originated from the individual that generated the enrolled embeddings and verification continues. The Erinyes server then uses the same feature embeddings extracted from the raw samples to run the Biometric Encryption Retrieval Algorithm on the BE Public Key sent in *msgVerify*. The derived public key is hashed, and the result is compared to the public key hash sent in *msgVerify*. If the hashes match, then the Erinyes server will have enough information to associate the public key with the user profile; the public key was retrieved with embeddings that 1) matched the enrolled feature embeddings for that user (meaning the provided samples were generated from that user) and 2) successfully decrypted the user's BE public key. Next, the Erinyes server will verify the digital signature of the message. If the signature check passes, then the Erinyes server successfully concludes the verification process. Since the Erinyes protocol is reliant on embeddings to provide a means of authentication, the storage of feature embeddings should be kept to an absolute minimum to decrease the likelihood of feature embedding leakage. Thus, the enrolled feature embeddings are deleted from the user profile and replaced with the user's public key on the Erinyes server and are never stored on the client.

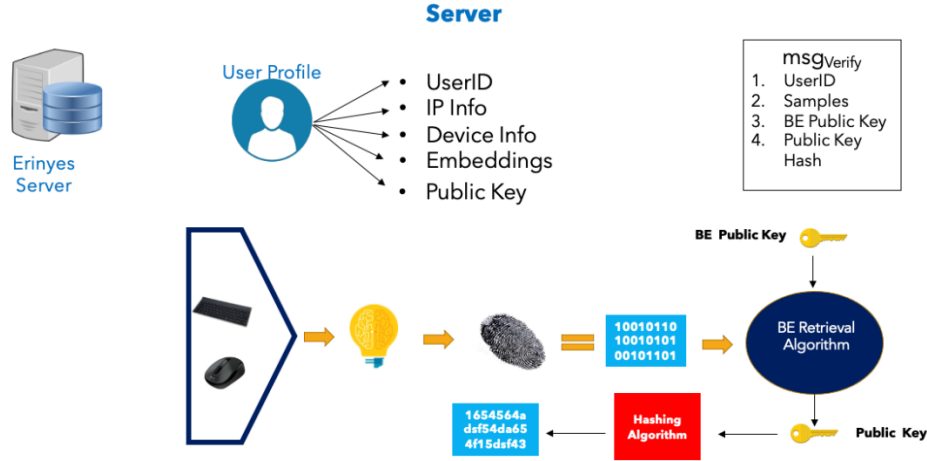


Figure 16. Erinyes Server-Side Verification Process.

Figure 17 highlights the major steps taken by the Erinyes server during the verification acknowledgement. As shown in Figure 17, the Erinyes server then drafts, signs, and encrypts a *msgVerify* ACK, which is an acknowledgment message back to the client, that contains a *msgVerify* acknowledgment tag and the UserID. Upon receipt of the *msgVerify* acknowledgment, the client only needs to keep its public key that is registered with the server; the BE public key is deleted from memory. *msgVerify*, is then sent in accordance with the security process previously discussed in the security section.

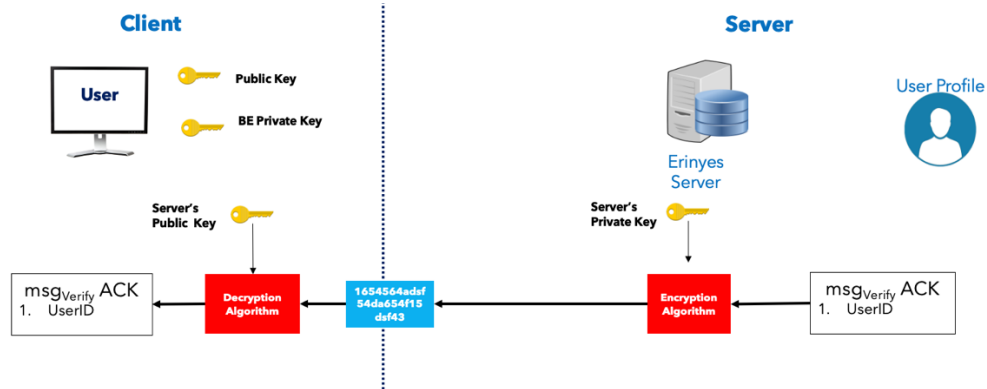


Figure 17. Erinyes Server-Side Verification Acknowledgement.

5. Authentication

Lastly, the relationship between the client's biometric feature embeddings and their asymmetric keys are leveraged by the Erinyes server to provide a continuous authentication capability that results in the issuance of limited-duration authentication certificates from the Erinyes server to the Erinyes client.

The limited-duration certificates issued by the Erinyes server contain the same information as a traditional x.509 certificate (subject name, issuer name, validity timeframe, and the signer's public key), but differ in that the validity timeframe is shortened to a configurable timeframe in the minute range, corresponding to creating a continuous authentication process.

After the client's public key has been verified, the Erinyes protocol is ready to transition to the authentication phase of its employment. Since both the client's public key, registered in the user profile on the Erinyes server, and the client's private key are only accessible by providing correct biometric samples, the server can leverage the relationship between public and private keys in asymmetric cryptography to authenticate an active client at any point in time.

As Figure 18 shows, the Erinyes server then initiates authentication requests based on a configurable timer, *auth_time*. Whenever *auth_time* expires, an authentication request message, *Msg_Auth_Req*, is drafted and sent by the server to the target client that needs to authenticate itself to the Erinyes server. *Msg_Auth_Req* contains an *Msg_Auth_Req* tag and the UserID of the target client. The message is encrypted as per the security process described earlier in this chapter.

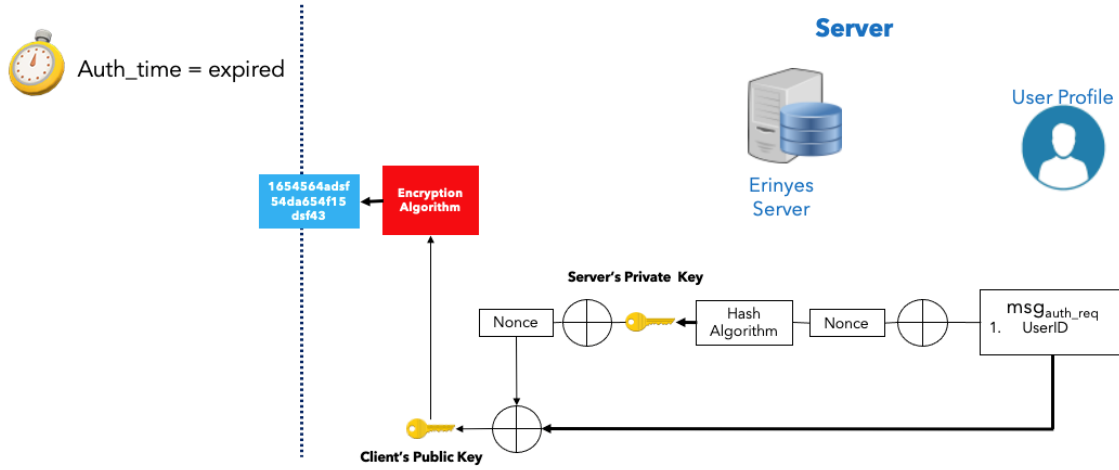


Figure 18. Erinyes Server-Side Authentication.

As Figure 19 demonstrates, when the client receives a *Msg_Auth_Req*, it engages the on-machine collection module to produce raw samples for the two biometric modalities. The raw samples are processed through the client's LSTM models, and the produced feature embeddings are concatenated and used in the BE Retrieval Algorithm on the client's BE private key. The ultimate result is the client's private key. The raw samples and feature embeddings are removed from memory immediately after use. The client's private key is used to decrypt the message, and the digital signature is verified. Assuming the checks are valid, and the UserID in the message belongs to the client, the client will then draft a response message, *Msg_Auth_Reply* shown in Figure 20, which contains a *Msg_Auth_Reply* tag, and the UserID. *Msg_Auth_Reply* is then encrypted and signed in accordance with the security process described in the security section of this chapter.

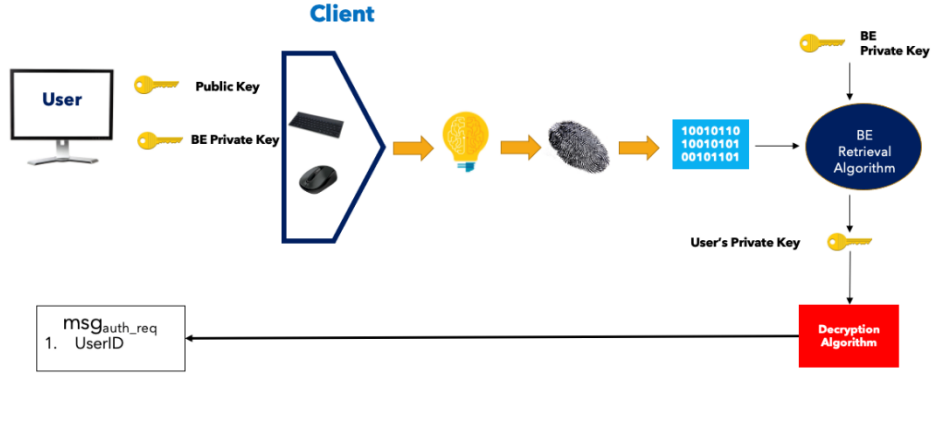


Figure 19. Erinyes Client-Side Authentication

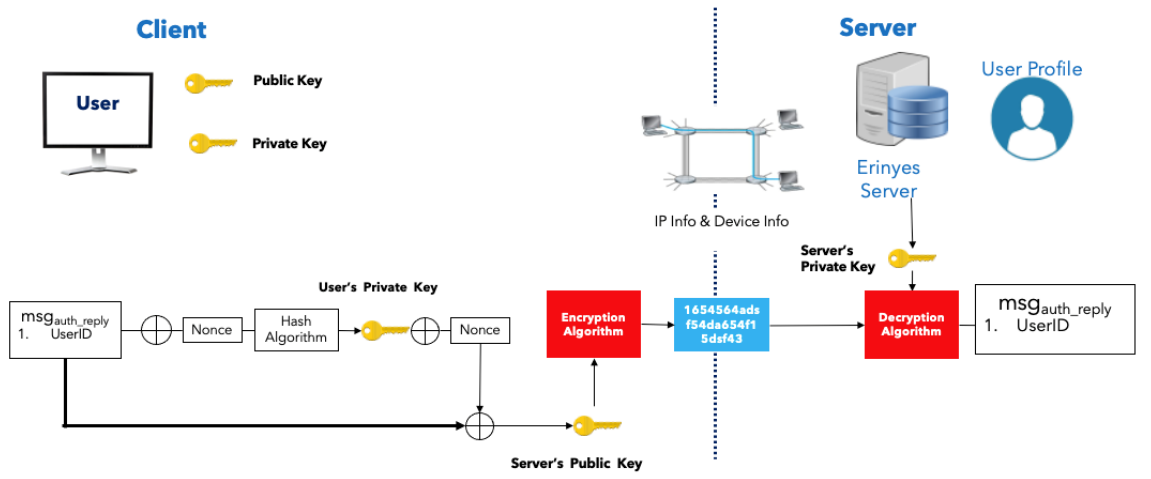


Figure 20. Erinyes Client-Side Authentication Reply

Upon receipt of the *Msg_Auth_Reply*, the server will obtain the IP address of the sender and decrypt the *Msg_Auth_Reply* to retrieve and UserID of the client. The UserID and the IP address are used to lookup the user profile on the Erinyes server. From the user profile, the server will retrieve the user's associated public key and attempt to verify the *Msg_Auth_Reply* signature. If the digital signature is successfully verified, the server considers the client authenticated and will send the client an authentication certificate as shown in Figure 21. The certificate is valid from the time of issuance to the time of issuance plus *auth_time*. If the digital certificate cannot be verified, then server will not issue a certificate for the current time plus *auth_time*.

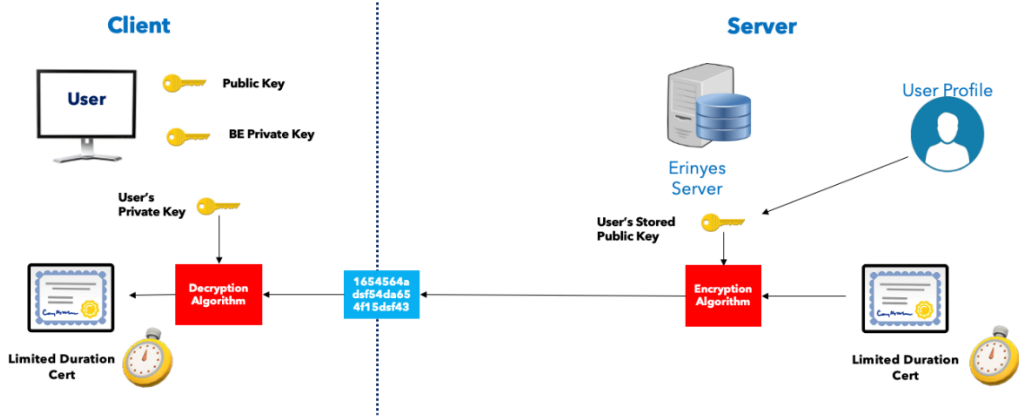


Figure 21. Erinyes Server-Side Certificate Issuance

D. SUMMARY

Traditional cryptographic systems suffer from vulnerabilities introduced by password-based release systems. We propose a protocol to aide in alleviating the vulnerabilities introduced by passwords. Our proposal, the Erinyes protocol, is enabled by ANNs and Biometric Encryption. The protocol consists of four phases: *enrollment*, *BE Key Binding*, *verification*, and *authentication*. We have discussed security aspects of our protocol, as well as a detailed discussion of its framework. In the following chapters, we present the metrics we have utilized to judge our protocol's effectiveness, as well as the results of our research and paths going forward.

IV. TESTING DESIGN, METRICS AND RESULTS

In Chapter II we introduced various metrics used by leading experts in the field (FAR, FRR, ERR, and Accuracy) to measure the effectiveness of biometric authentication systems. To fully function as a continuous authentication system, the Erinyes protocol uses multiple modules that must individually perform accurately and communicate with each other. To ensure accurate results throughout the protocol, we utilized a gradual testing design, which first measures each key modules' error rate accuracy and then performs integration testing for error rate accuracies and system performance of the fully incorporated service. This chapter covers the implementation of the graduated testing design and our results for each phase. We begin by first describing the testing and results of each of the LSTM modules. We then introduce our testing procedure and results for Biometric Encryption. Lastly, we discuss how we tested the fully integrated system and how the system performed.

A. KEYSTROKE MODEL AUTHENTICATION TESTING

There are two foundational LSTM models in the Erinyes protocol. The first model is the keystroke dynamic LSTM, described in this section. The second model, the mouse cursor movement model, is described in Section G. To measure performance, the keystroke model was subjected to an authentication test where the error rates of the models' predictions were compared as the number of enrolled users increased. In the multiple rounds of model testing, the number of enrolled users, k , was increased from 5 to 100, then 500 and lastly to 1,000. For each round, in which k is fixed, the testing harness was invoked in two passes. In the first pass, the EER of the model was determined and used as the authentication threshold. The second pass selected a threshold in range of the determined thresholds across the k values from the first pass. The selected threshold was then used on another subset of the testing data to measure how well the model performed with the selected threshold for any given k .

As discussed in Chapter III, each user in the keystroke dataset contains 15 keystroke sequences [36]. Acien et al. reported that 10 enrolled sequences were optimal for their

LSTM authentication testing [28]. For this reason, our testing harness used 10 of the 15 user sequences as enrollment sequences, and the remaining 5 sequences as a test set. Like in [28], to generate imposter sequences, for each enrolled user, we choose one test sequence from each remaining subject. The result was that, for each enrolled user, their test set contained 5 genuine sequences and $k-1$ imposter sequences. Sequences from a user’s test set were processed through the LSTMs, and the resulting feature embeddings were compared, via Euclidean distance, to feature embeddings that had been enrolled for the same user. If the calculated Euclidean distance, averaged out across the 10 gallery samples, was less than the determined threshold, the test harness predicts a match. If the distance is greater than the determined threshold, the test harness predicts a non-match. In each round of fixed k testing, error rates, expressed in terms of accuracy, FAR, and FRR, were calculated. FAR and FRR calculations compared the match and non-match predictions with the ground truth. Overall accuracy was calculated by dividing the number of correct predictions by the number of samples processed during the testing round.

B. KEYSTROKE MODEL AUTHENTICATION TESTING RESULTS

Prior to obtaining numerical metrics, we wanted to visualize our keystroke LSTM model’s ability to produce feature embeddings that could accurately classify users distinctly apart from each other. To produce the visual representations, the dimensionality of the feature embeddings was reduced to two dimensions with a t-distributed stochastic neighbor embedding (t-SNE) plot. The t-SNE plot for the keystroke LSTM model was produced using batches of samples that were obtained from the test keystroke dataset; there were 32 sets of sequences per batch, and each set of sequences contained all 15 sequences from a particular user. Due to the data pipeline’s randomness in selecting users from the test dataset during each batch, it was expected that several iterations of t-SNE plotting would result in different users and different plots.

Figure 22 shows t-SNE plots across three iterations where, in each iteration, three batches of test data were sampled. The coloring of the users only signifies the numeric ordering of the users’ number. Although each cluster is unique to an individual, we also see groupings formed in each of the three iterations. These types of patterns could be

indicative of users with similar typing behaviors, such as speed. Despite similar behaviors and groupings, there is little overlap amongst the group. Heuristically, the combination of decent clustering of embeddings from the same user in 2-D space and lack of overlap indicated that the model would be able to perform authentication tasks with sufficient accuracy.

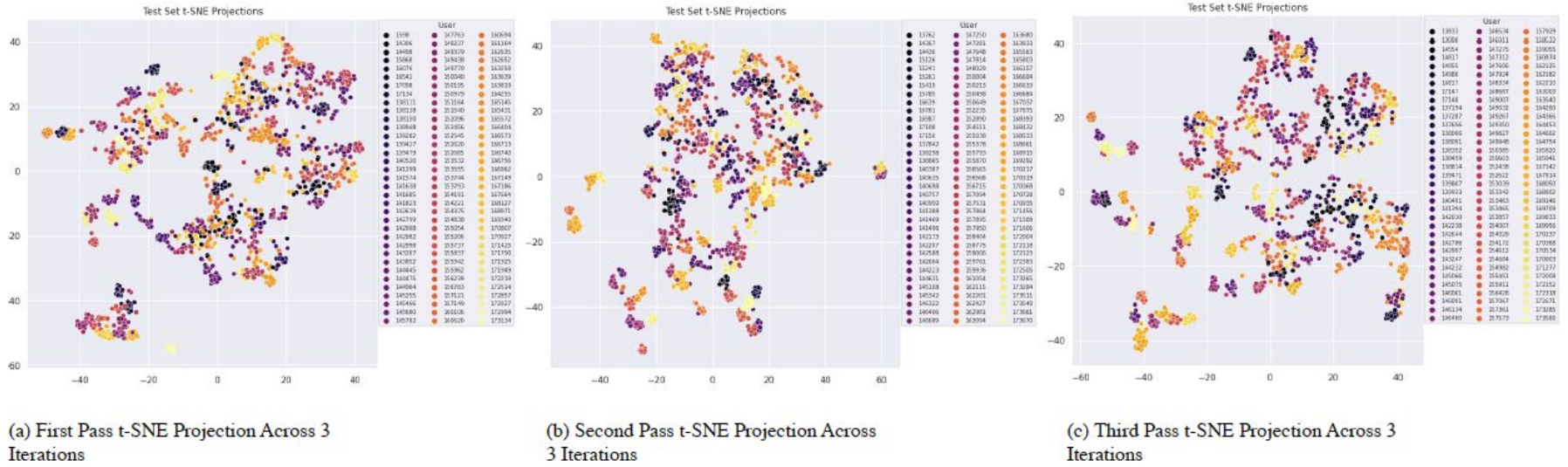


Figure 22. Keystroke LSTM t-SNE Projections.

The keystroke LSTM model was then subjected to the keystroke testing harness described in Section E to produce numerical data that would support the visual t-SNE projections. Figures 23–26 demonstrate the threshold and metrics results obtained for the first and second pass of the testing harness for $k=5$, 100, 500, and 1000 users respectively. As shown, as the value of k increased, all error rates decreased. The optimal thresholds (which also equals the EER) ranged from 0.6 to 0.9, first pass accuracies from 93.33% to 99.56%, FARs from 2.22% down to 0.16%, and FRRs from 4.44% to 0.28%. From those results a threshold of 0.6 was chosen for the second pass test. The 0.6 threshold was too low for $k=5$ and we saw accuracy plummet to 64.44% and the FRR increased to 35.56%. However, for the other values of k , the threshold of 0.6 didn't significantly deviate optimal performance. Apart from $k=5$, second pass tests resulted in accuracies ranging from 97.13% to 99.56%, FARs from 0.39% to 0.16%, and FRRs from 2.47% to 0.28%. Of note, while error rates continued to decrease as the number of k increased, for a threshold of 0.6, the rate of increase drastically reduced between $k=500$ and $k=1000$.

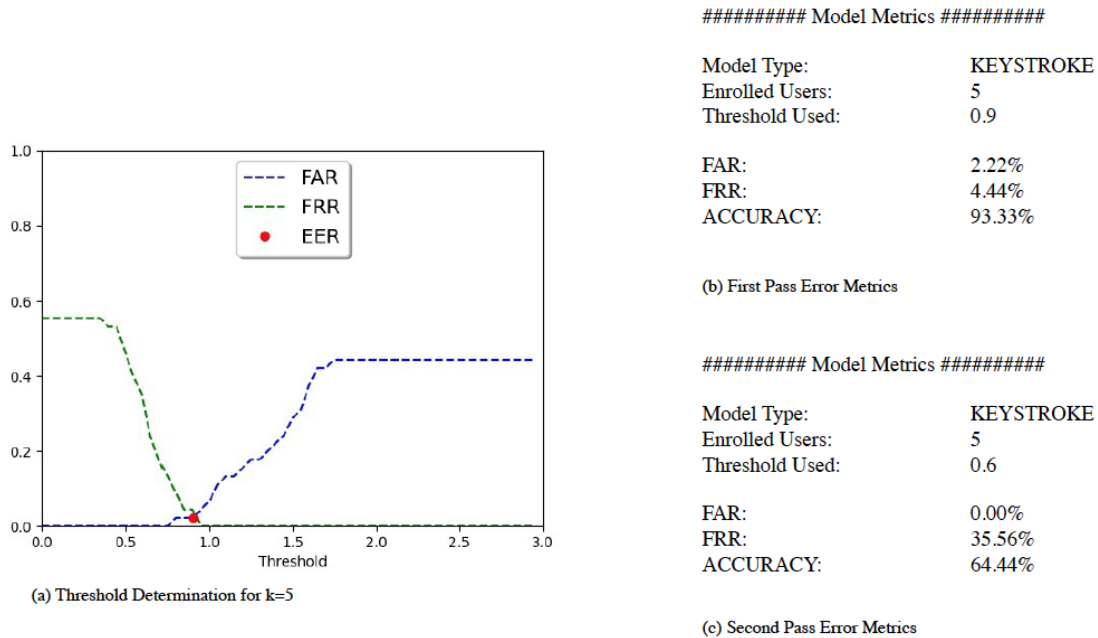
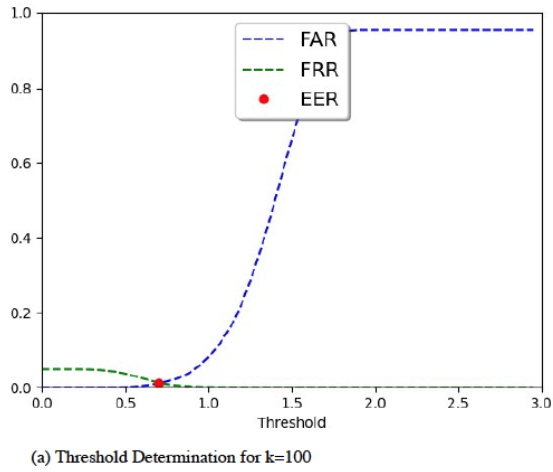


Figure 23. Erinyes Keystroke LSTM Metrics (5 Enrolled Users).



```
##### Model Metrics #####

Model Type:          KEYSTROKE
Enrolled Users:      100
Threshold Used:       0.7

FAR:                  1.02%
FRR:                  1.25%
ACCURACY:              97.73%
```

(b) First Pass Error Metrics

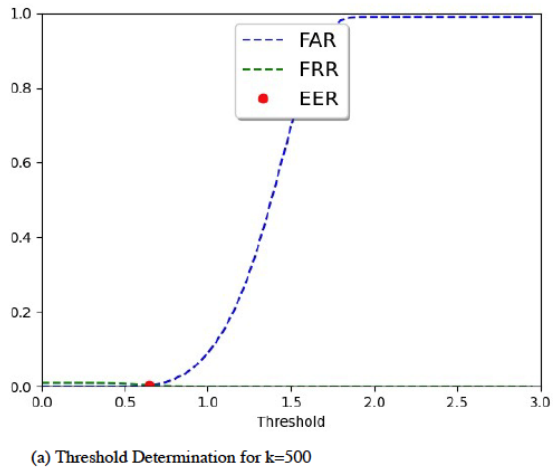
```
##### Model Metrics #####

Model Type:          KEYSTROKE
Enrolled Users:      100
Threshold Used:       0.6

FAR:                  0.39%
FRR:                  2.47%
ACCURACY:              97.13%
```

(c) Second Pass Error Metrics

Figure 24. Erinyes Keystroke LSTM Metrics (100 Enrolled Users).



```
##### Model Metrics #####

Model Type:          KEYSTROKE
Enrolled Users:      500
Threshold Used:       0.65

FAR:                  0.35%
FRR:                  0.41%
ACCURACY:              99.24%
```

(b) First Pass Error Metrics

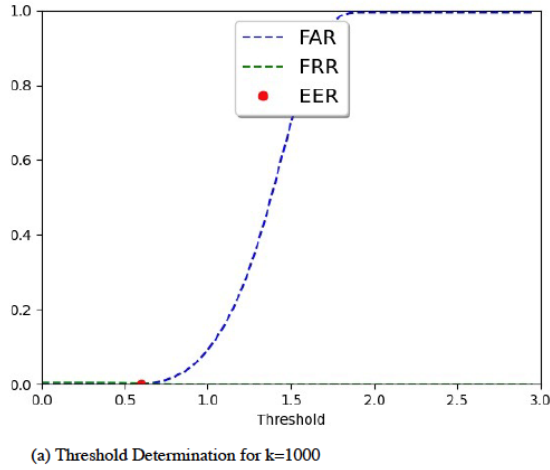
```
##### Model Metrics #####

Model Type:          KEYSTROKE
Enrolled Users:      500
Threshold Used:       0.6

FAR:                  0.16%
FRR:                  0.54%
ACCURACY:              99.30%
```

(c) Second Pass Error Metrics

Figure 25. Erinyes Keystroke LSTM Metrics (500 Enrolled Users).



```
##### Model Metrics #####
Model Type:          KEYSTROKE
Enrolled Users:      1000
Threshold Used:      0.6

FAR:                 0.16%
FRR:                 0.28%
ACCURACY:            99.56%
```

(b) First Pass Error Metrics

```
##### Model Metrics #####
Model Type:          KEYSTROKE
Enrolled Users:      1000
Threshold Used:      0.6

FAR:                 0.16%
FRR:                 0.28%
ACCURACY:            99.56%
```

(c) Second Pass Error Metrics

Figure 26. Erinyes Keystroke LSTM Metrics (1000 Enrolled Users).

C. MOUSE CURSOR MOVEMENT MODEL AUTHENTICATION TESTING

Once the keystroke model was reflecting effective performance, the mouse cursor movement model was developed and tested. The mouse cursor movement LSTM module was subjected to a similar test harness as the keystroke dynamics module with a few key modifications. First, the number of enrolled users, k , was fixed at 100 users due to the limitations of the available mouse cursor movement dataset. Additionally, each user had 50 point and click actions, of which, ten point and click actions constituted one sequence for a user. As a result, each user had a total of 5 sequence samples. The number of enrolled samples per user was made variable, such that there were three rounds of testing in which 1, 2, and 3 point and click sequence feature embeddings were enrolled for each user.

The mouse cursor test harness also executed two passes to determine an appropriate threshold per round in the same manner as described for the keystroke test harness. For every round in which the number of enrolled samples was fixed, each user's test set contained the corresponding genuine test samples (5 - the number of enrolled samples) and 99 imposter samples (one sequence taken from every other user in the mouse cursor

movement dataset). The samples in each user’s test set were processed and performance was measured and reported in a similar fashion to the keystroke dynamics test harness.

D. MOUSE CURSOR MOVEMENT MODEL AUTHENTICATION TESTING RESULTS

Like the keystroke dynamic LSTM model, t-SNE plots for the mouse cursor movement LSTM model were generated to heuristically determine the model’s capacity to perform authentication tasks. As shown in Figure 27, three passes of the t-SNE generator were conducted. Each pass consisted of various iterations, in which an iteration consisted of the grabbing of one-batch of test data. In contrast to the keystroke dynamic dataset, which features over 168,000 users, the mouse dataset only contains 100 users. Thus, every user was captured in a single pass of the t-SNE generator. Like the keystroke t-SNE plots, the clustering present across the three passes of t-SNE plotting indicated that the model would be able to perform the desired authentication tasks.

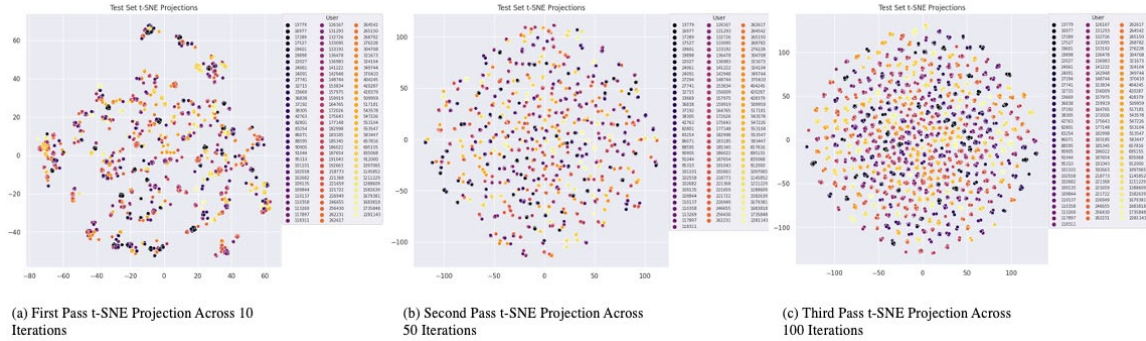


Figure 27. Mouse Cursor Movement LSTM t-SNE Projections.

As detailed in the mouse cursor movement training harness in Section G, the number of enrolled users, k , was fixed at 100. The mouse cursor testing harness measured the performance of the model as the number of enrolled samples per user, G , was increased. For each value of G , two passes of the harness were conducted; the first to determine a suitable threshold value, the second to evaluate the selected threshold across all G values.

Figures 28–30 detail the performance of the model across the two passes as G is increased from 1 to 2 to 3. As shown, as the number of enrolled samples, G , increased, accuracies increased, and error rates decreased. For the first pass, thresholds ranged from 0.7 to 0.8, and resulted in accuracies ranging from 93.51% to 96.96%, FARs ranged from 2.89% to 1.14%, and FRRs ranged from 3.59% to 1.90%. From those results, a threshold of 0.6 was selected to be used in the second pass. For $G=1$, the 0.6 threshold resulted in slightly better accuracy of 94.89% (up from 93.41% on the first pass) and FAR of 1.43 (down from 2.89% on the first pass). The decrease in FAR, however, came at the slight cost of increasing the FRR from 3.59% on the first pass to 3.68%. A similar outcome occurred for $G=2$ and $G=3$. The top performing second pass test at a 0.6 threshold occurred when $G=3$. During this test, the model achieved an accuracy of 97.89%, FAR of 0.14%, and FRR of 1.97%.

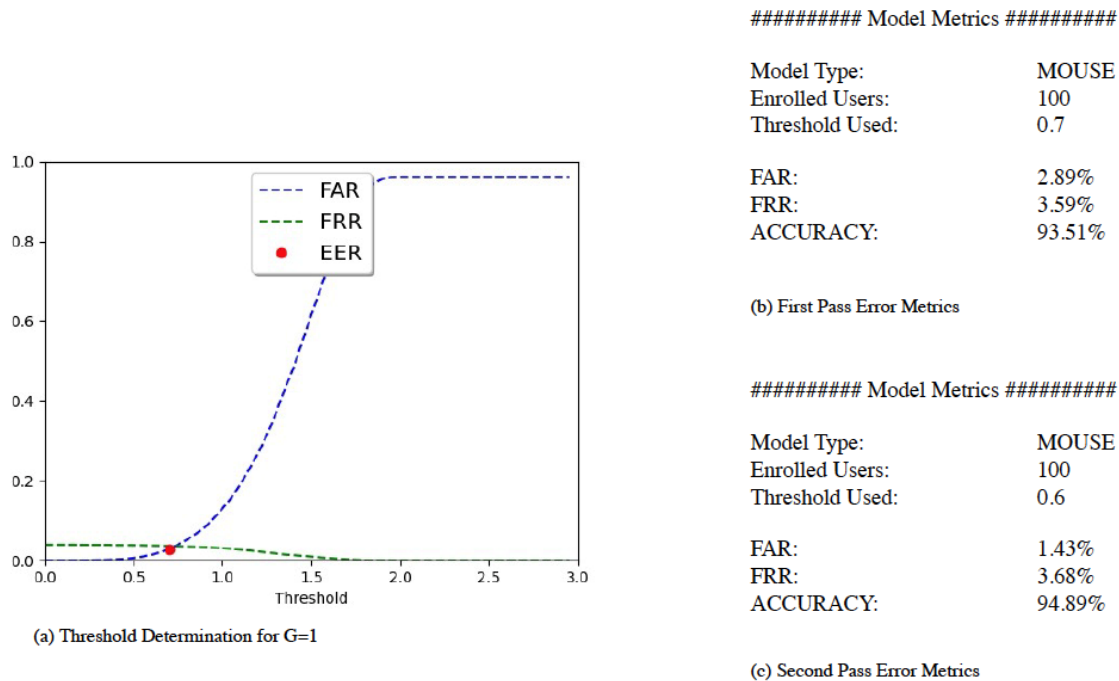
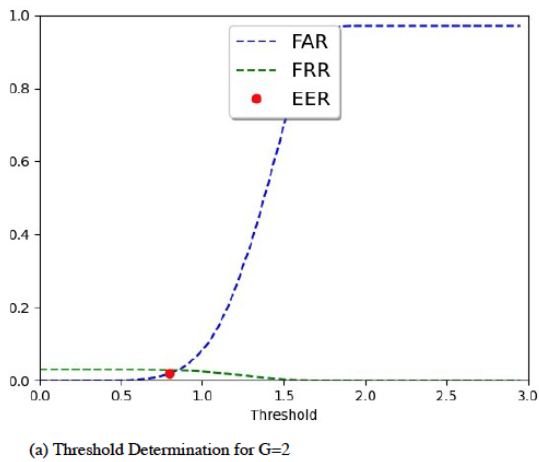


Figure 28. Mouse Cursor Movement LSTM Metrics (Round 1).



Model Metrics

Model Type: MOUSE
Enrolled Users: 100
Threshold Used: 0.8

FAR: 1.81%
FRR: 2.76%
ACCURACY: 95.42%

(b) First Pass Error Metrics

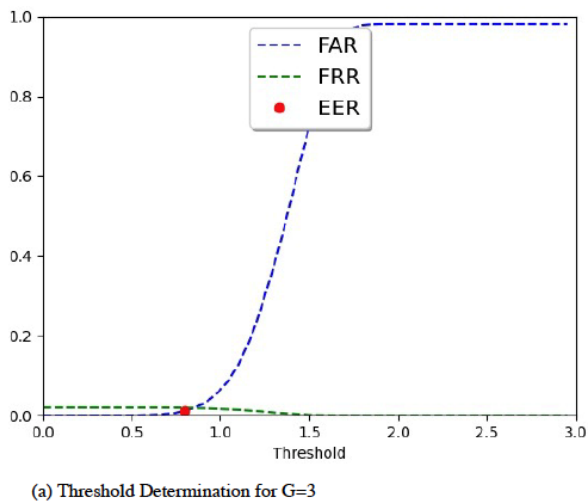
Model Metrics

Model Type: MOUSE
Enrolled Users: 100
Threshold Used: 0.6

FAR: 0.27%
FRR: 2.92%
ACCURACY: 96.80%

(c) Second Pass Error Metrics

Figure 29. Mouse Cursor Movement LSTM Metrics (Round 2).



Model Metrics

Model Type: MOUSE
Enrolled Users: 100
Threshold Used: 0.8

FAR: 1.14%
FRR: 1.90%
ACCURACY: 96.96%

(b) First Pass Error Metrics

Model Metrics

Model Type: MOUSE
Enrolled Users: 100
Threshold Used: 0.6

FAR: 0.14%
FRR: 1.97%
ACCURACY: 97.89%

(c) Second Pass Error Metrics

Figure 30. Mouse Cursor Movement LSTM Metrics (Round 3).

E. COMBINED LSTM MODELS' AUTHENTICATION TESTING

As described in [26] and discussed in Chapter III, the Erinyes protocol was designed as a multiple modality authentication system to combat biometric attacks, such as those described in [22]. The next stage of integration testing sought to measure the authentication performance of the combined output from the keystroke and mouse cursor movement LSTMs. To do so, the keystroke model's output feature embeddings were concatenated with the mouse cursor movement model's output feature embeddings. Since the number of users across the two models was the same, concatenated feature embeddings were simulated as coming from the same user. The concatenated feature embeddings adopt the user label from the constituent keystroke embedding.

For the testing harness, the number of enrolled users, k , was lower bounded by the number of users available in the mouse cursor movement dataset (100 users). Additionally, the number of enrolled combined feature embeddings for each user, G , was bounded by the optimal results reported in [28] for the keystroke dataset. Thus, G was set equal to 10 combined feature embeddings. For each user, all $G_k=10$ enrolled keystroke embeddings were concatenated with the same associated enrolled $G_m=1$ mouse embedding. This design was chosen to simulate a situation in which the enrolled embeddings for the keystroke and mouse models are unable to be equal, while also preserving enough mouse samples to be used for each user's test set. Each user's test set included 99 imposter embeddings, and 5 genuine test embeddings. Performance of the model, measured in the same manner as the keystroke and mouse test harness, was calculated as the threshold was varied between the optimal thresholds (threshold = EER) for: the keystroke model, mouse model, and combined models.

F. COMBINED LSTM MODELS' AUTHENTICATION TESTING RESULTS

Figure 31 depicts the first pass and second pass threshold and metrics produced by the combined *keystroke+mouse* testing harness. The metrics in (d) of Figure 31 represent the first pass run of the test harness. During this run the optimal threshold was found to be 1.35 and the model performed with an accuracy of 95.17%, FAR of 2.02%, and FRR of 2.81%. Since k and G are fixed, the second pass of the testing harness adjusted the

threshold. The metrics in (b), (c), and (e) all depict the results of that threshold adjustment. In (b) and (c), when the threshold was lowered from the optimal 1.35 value, accuracy slightly increased and FAR was reduced to 0%. However, the performance gains across these two metrics came at the tradeoff of an increase in FRR of nearly 2% for both runs of the test. In (e), the threshold on the second pass was increased from 1.35 to 1.75. This change resulted in a reduced FRR from 2.81% to 0.68%, but also significantly increased FAR from 2.02% to 21.41% and significantly decreased the accuracy from 95.17% to 77.90%. Our concluding analysis for the combined keystroke and mouse cursor model was that the threshold of 1.35 was best for the Erinyes protocol, as it preserved high accuracy (upwards of 95%) while maintaining a low and balanced error rate across the FAR and FRR metrics. The 1.35 EER for this combined model does not represent significant degradation from the 1.2 EER achieved in [28] on the singular keystroke model, and by providing multiples modalities, [26] describes, allows for protection against biometric attacks such as those introduced in [22].

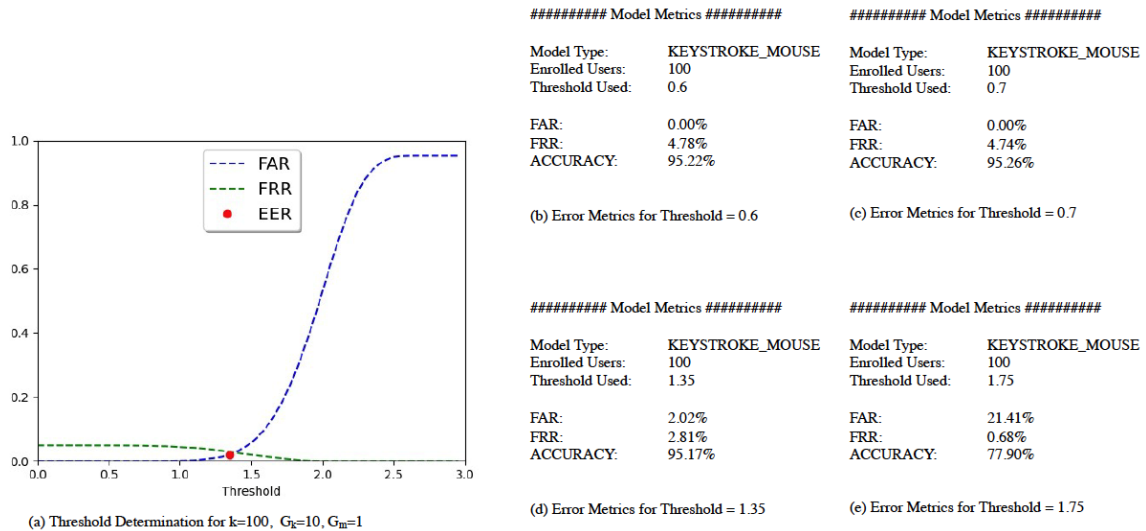


Figure 31. Combined Keystroke and Mouse LSTM Metrics.

G. BIOMETRIC ENCRYPTION

The next phase of development and testing of the Erinyes protocol was biometric encryption, BE. Like the combined keystroke and mouse testing harness, the BE testing harness was lower bounded by $k=100$ number of users of the mouse dataset. A temporary dataset was created to feed the testing dataset. For each of the 100 users, the temporary dataset was passed the user's 10 combined feature embeddings and their UserID. The testing dataset was then generated using the stored temporary dataset. First, one random user from the set of 100 was chosen as the genuine user. From the genuine user's 10 combined embeddings, a random one was chosen as the binding embedding. To complete our testing dataset, the remaining 9 embeddings and associated UserID of the genuine user were added to the testing set. The imposter samples were then added to the test dataset by randomly choosing 1 of the 10 embeddings for each of the remaining 99 users in the temporary dataset. The resulting dataset was 108 total samples comprised of the user's 9 enrolled embeddings, and 99 imposter samples.

The process of creating a test dataset, choosing a genuine user, and testing against the test dataset was repeated for 5 different users. During each run, a 128-bit RSA key was created and hashed for ground truth comparison. The 128-bit RSA key was then used to bind to the binding embedding by creating a polynomial function, using the process described in Chapter III, and projecting the embedding onto the polynomial to generate the fuzzy vault, or BE Key. For each of the remaining 108 samples in the testing dataset, each sample was used to attempt and retrieve the coefficients of the polynomial, and thus the cryptographic key, from the BE Key. During each comparison, the cryptographic key produced by the comparing sample was hashed and compared to the hash of the original 128-bit RSA key. If the user was able to reproduce the original key and the embedding was from one of the genuine test embeddings, then the produced hash and the original hash matched, and the test harness reported a match. If the user was not able to reproduce the original key or the user was one of the imposter embeddings, then the hashes did not match, and the test harness reported a non-match. In each round of testing, error rates, expressed in terms of accuracy, FAR, and FRR, were calculated. FAR and FRR calculations compared the match and non-match predictions with the ground truth. Overall accuracy

was calculated by dividing the number of correct predictions by the number of samples processed during the testing round.

H. BIOMETRIC ENCRYPTION TESTING RESULTS

The testing harness for the biometric encryption module of Erinyes was concurrently performed with 5 different test datasets. On each run, a different user was selected as a genuine user. As Figure 32 shows, FRR across each of the 5 users averaged 0.74% with only one user falsely rejecting more than one genuine sample. The FAR and accuracy averages across the 5 users were 1.11% and 98.15% respectively.

Erinyes Metrics for 5 Clients Running Concurrently: - Selection Pool Size = 108 (per run)					
##### Erinyes Metrics #####		##### Erinyes Metrics #####		##### Erinyes Metrics #####	
Genuine User:	11884	Genuine User:	11749	Genuine User:	11775
Genuine Test Samples Used:	9	Genuine Test Samples Used:	9	Genuine Test Samples Used:	9
Imposter Test Samples Used:	99	Imposter Test Samples Used:	99	Imposter Test Samples Used:	99
FAR:	1.85%	FAR:	0.00%	FAR:	1.85%
FRR:	0.00%	FRR:	0.00%	FRR:	2.78%
ACCURACY:	98.15%	ACCURACY:	100.00%	ACCURACY:	95.37%
(a) Error Metrics for User 11884		(b) Error Metrics for User 11749		(c) Error Metrics for User 11775	
##### Erinyes Metrics #####		##### Erinyes Metrics #####			
Genuine User:	11842	Genuine User:	1209		
Genuine Test Samples Used:	9	Genuine Test Samples Used:	9		
Imposter Test Samples Used:	99	Imposter Test Samples Used:	99		
FAR:	0.00%	FAR:	1.85%		
FRR:	0.93%	FRR:	0.00%		
ACCURACY:	99.07%	ACCURACY:	98.15%		
(d) Error Metrics for User 11842		(e) Error Metrics for User 1209			

Figure 32. Erinyes Biometric Encryption Metrics (108 Mixed Samples).

While the module performed well in these tests, there was an inherent lack of balance between genuine samples and imposter samples. For each test set, there were only 9 genuine samples compared to 99 imposter samples. Thus, the test harness was more likely to measure the ability of the module to detect imposter samples versus the ability of the module to determine if a sample is genuine. To measure the module's performance for discerning genuine samples, the module was subjected to another round of testing with test

sets that contained only genuine samples. To create this new test dataset, first, we chose a random user and stored their 10 enrollment embeddings into a temporary dataset. Next, from the 10 embeddings, 1 random embedding was chosen as the binding embedding and then the other 9 embeddings were added to the test dataset. To simulate the randomness that would occur in an authentication system, we supplemented the dataset with a random mixture of 50 additional samples chosen from the 10 enrollment embeddings. The final dataset was 59 test samples comprised of a random mixture of the user's 10 enrolled embeddings, with all 10 guaranteed to appear at least once. Figure 33 show the results of the 5 runs of this test. As shown, FRR was perfect, and accuracy jointly performed at 100% accuracy. Since there were no imposter samples, of which the system could falsely accept, FAR was an irrelevant contributor for this test.

Erinyes Metrics for 5 Clients Running Concurrently: - Selection Pool Size = 59 (per run)					
##### Erinyes Metrics #####		##### Erinyes Metrics #####		##### Erinyes Metrics #####	
Genuine User:	1195	Genuine User:	11885	Genuine User:	1201
Genuine Test Samples Used:	59	Genuine Test Samples Used:	59	Genuine Test Samples Used:	59
Imposter Test Samples Used:	0	Imposter Test Samples Used:	0	Imposter Test Samples Used:	0
FAR:	0.00%	FAR:	0.93%	FAR:	0.00%
FRR:	0.00 %	FRR:	0.00 %	FRR:	0.00 %
ACCURACY:	100.00% 59/59	ACCURACY:	100.00% 59/59	ACCURACY:	100.0% 59/59
(a) Error Metrics for User 1195		(b) Error Metrics for User 11885		(c) Error Metrics for User 1201	
##### Erinyes Metrics #####		##### Erinyes Metrics #####			
Genuine User:	11822	Genuine User:	11895		
Genuine Test Samples Used:	59	Genuine Test Samples Used:	59		
Imposter Test Samples Used:	0	Imposter Test Samples Used:	0		
FAR:	0.00%	FAR:	0.00%		
FRR:	0.00%	FRR:	0.00%		
ACCURACY:	100.00% 59/59	ACCURACY:	100.00% 59/59		
(d) Error Metrics for User 11822		(e) Error Metrics for User 11895			

Figure 33. Erinyes Biometric Encryption Results (59 Genuine Samples).

This test indicated that the system was able to identify users based off their enrollment samples, but during the creation of our imposter sets, the system was creating new users under the same user number but different embeddings. To prove this was more of an issue in our generation of random sets and not an issue with the biometric system

being able to distinguish between correct embeddings, we ran the original test but modified the creation of the test set to ensure imposter samples did not match the genuine sample user number. Since this extra check increased the complexity of test set creation, the overall execution time of the test increased but as Figure 34 shows, the results matched our predictions. In all 5 tests FRR dropped to 0.00% and the average across the 5 jointly increased to an average of 99.07%.

Erinyes Metrics for 5 Clients Running Concurrently: - Selection Pool Size = 108 (per run)											
##### Erinyes Metrics #####				##### Erinyes Metrics #####				##### Erinyes Metrics #####			
Genuine User:		11954		Genuine User:		1216		Genuine User:		11761	
Genuine Test Samples Used: 9				Genuine Test Samples Used: 9				Genuine Test Samples Used: 9			
Imposter Test Samples Used: 99				Imposter Test Samples Used: 99				Imposter Test Samples Used: 99			
FAR:		0.00%		FAR:		0.93%		FAR:		1.85%	
FRR:		0.00%		FRR:		0.00%		FRR:		0.00%	
ACCURACY:		100.00%		ACCURACY:		99.07%		ACCURACY:		98.15%	
		108/108				107/108				106/108	
(a) Error Metrics for User 11954				(b) Error Metrics for User 1216				(c) Error Metrics for User 11761			
##### Erinyes Metrics #####				##### Erinyes Metrics #####							
Genuine User:		11857		Genuine User:		1195					
Genuine Test Samples Used: 9				Genuine Test Samples Used: 9							
Imposter Test Samples Used: 99				Imposter Test Samples Used: 99							
FAR:		0.00%		FAR:		1.85%					
FRR:		0.00%		FRR:		0.00%					
ACCURACY:		100.00%		ACCURACY:		98.15%					
		108/108				106/108					
(d) Error Metrics for User 11857				(e) Error Metrics for User 1195							

Figure 34. Erinyes Biometric Test Results for Enrollment Embeddings.

I. FULL SYSTEM TESTING

Following satisfactory performance of the biometric system, we continued our test to the fully integrated system with the understanding that our results would be driven by the makeup of the test dataset. To test full system integration, the testing harness was designed to measure performance during the authentication phase of the Erinyes protocol. The constraints regarding enrolled users, enrolled feature embeddings, and test set composition from the previous test harnesses carried over to the full system test. As a result, the testing harness contained $k=100$ number of users. For each of the 100 users, the testing harness was passed $G=10$ combined feature embeddings for enrollment, 5 genuine test

embeddings and 99 imposter embeddings. 5 separate threads were launched to simulate 5 clients. The user enrolled in each of the 5 clients was chosen at random from the available 100 users.

A time, t , was used to denote one *auth_time* cycle from the authentication phase of the Erinyes protocol. The testing harness performed a configurable amount of t_cycles for each of the 5 randomly chosen enrolled users. During each t , a random sequence from the enrolled user's test set was run through the authentication phase of the Erinyes protocol. Successful authentication was indicated by the issuance of the Erinyes certificate to the user, while negative authentication was indicated by the lack of a certificate (if $t = 1$) or an expired certificate. Performance was measured in terms of accuracy, FAR, and FRR for each t . Upon completion of all t_cycles for all 5 users, the error rates were averaged across all cycles for all 5 users, resulting in the final performance metrics for the full integrated system.

J. FULL SYSTEM INTEGRATION TESTING RESULTS

As described in the previous section, the integration test was performed with 5 clients running concurrently on the multi-threaded Erinyes Server. Additionally, the Erinyes software was configured to run for 20 t_cycles with an *auth_time* of 90 seconds each t_cycle .

Figure 35 represents the results of Erinyes' full system integration test. Each of the 5 users in the figure are identified by a tuple containing their IP address and port number. As the results show, the protocol performed well on the overall test harness. Across all 5 clients, the average FAR was 0%, meaning that no certificates were issued to clients that presented imposter biometric samples. The FRR averaged to 2% across all 5 clients, meaning that in 2 instances out of 100, a client wasn't granted a certificate despite presenting genuine biometric samples. Overall system accuracies were all >95%. However, as described in the BE results section, it is important to note that these metrics were largely driven by the framework of the test dataset.

Erinyes Metrics for 5 Clients Running Concurrently:			
- T_cycles =		20 cycles	
- Auth_time =		90 seconds	
- Selection Pool Size =		114 samples	
##### Erinyes Metrics #####			
Client IP:	127.0.0.1	Client IP:	127.0.0.1
Client Port:	52348	Client Port:	52351
FAR:	0.00% 0/20	FAR:	0.00% 0/20
FRR:	5.00% 1/20	FRR:	0.00% 0/20
ACCURACY:	95.00%	ACCURACY:	100.00%
(a) Error Metrics for Client (127.0.0.1, 52348)		(d) Error Metrics for Client (127.0.0.1, 52351)	
##### Erinyes Metrics #####			
Client IP:	127.0.0.1	Client IP:	127.0.0.1
Client Port:	52349	Client Port:	52352
FAR:	0.00% 0/20	FAR:	0.00% 0/20
FRR:	0.00% 0/20	FRR:	5.00% 1/20
ACCURACY:	100.00%	ACCURACY:	95.00%
(b) Error Metrics for Client (127.0.0.1, 52349)		(e) Error Metrics for Client (127.0.0.1, 52352)	
##### Erinyes Metrics #####			
Client IP:	127.0.0.1		
Client Port:	52350		
FAR:	0.00% 0/20		
FRR:	0.00% 0/20		
ACCURACY:	100.00%		
(c) Error Metrics for Client (127.0.0.1, 52350)			

Figure 35. Erinyes Full System Metrics (114 Mixed Sample Test).

Figure 36 depicts an issued certificate for client (127.0.0.1, 52349) with its corresponding validity period in GMT. The entirety of the decoded certificate can be viewed in Appendix A. Figures 37 and 38 show samples of the logs from the Erinyes server and client (127.0.0.1, 52349), respectively, that highlight the sequence of events leading to successful authentication. The timestamps from the logs are reported in local time (GMT-7) and match the validity time reported on the certificate. Full text logs for the interactions between the Erinyes server and client (127.0.0.1, 52349) are made available in Appendix B.

```

Version: 3 (0x2)
Serial Number:
    5e:64:ad:a2:5d:59:9a:c1:fe:6a:c6:1d:80:ad:7d:0a:ba:21:4a:fb
Signature Algorithm: sha256WithRSAEncryption
Issuer: C = US, ST = California, L = Monterey, O = Erinyes LLC, CN = Erinyes Protocol
Validity
    Not Before: Aug 24 15:49:30 2022 GMT
    Not After : Aug 24 15:51:00 2022 GMT

```

Figure 36. Erinyes Issued Limited Duration Certificate.

```

5055-08-54 08:20:35.143 - 26L6GL(,J51'0'0'T,' 25340) - INFO - 1~cLcfe 5 c0mbf6f6eq'
5055-08-54 08:40:33.045 - 26L6GL(,J51'0'0'T,' 25340) - INFO - CjTeuf 270u9f06 m2e AEB7IED' AV7ID CEHLIEICVIE I22NED 10 C7IEM1: (,J51'0'0'T,' 25340)'
5055-08-54 08:40:30.110 - 26L6GL(,J51'0'0'T,' 25340) - INFO - vnefu~Leb7l w622906 zncce224nf7l q6c1Abf6q'
5055-08-54 08:40:30.140 - 26L6GL(,J51'0'0'T,' 25340) - INFO - vnefu~Leb7l w622906 uouce b922eq Leb79l cpeck' meqyn05400:40:5050554200 m2e 9q9eq 20 tpe nouce c0cme'
5055-08-54 08:40:04.213 - 26L6GL(,J51'0'0'T,' 25340) - INFO - Euc1Abf6q 9nfu~Leb7l w622906 L6c67l6q f4ow c7Teuf'
5055-08-54 08:40:04.404 - 26L6GL(,J51'0'0'T,' 25340) - INFO - 1~cLcfe 5: 26L6GL 26uf 9nfu~Ledn622 20 c7Teuf: (,J51'0'0'T,' 25340)'

```

Figure 37. Sample Erinyes Server Log.

```

2022-08-24 08:49:04.494 - client_('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 08:49:04.494 - client_('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:49:26.964 - client_('127.0.0.1', 52349) - INFO - BE Key Retrieval was SUCCESSFUL: auth_req was decrypted.
2022-08-24 08:49:26.964 - client_('127.0.0.1', 52349) - INFO - Auth_req nonce passed replay check. WedAug2408:49:04202240107 was added to the Nonce Cache.
2022-08-24 08:49:28.970 - client_('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:49:33.022 - client_('127.0.0.1', 52349) - INFO - Cert issue message nonce passed replay check. WedAug2408:49:3120224511 was added to the Nonce Cache.
2022-08-24 08:49:33.026 - client_('127.0.0.1', 52349) - INFO - Erinyes Certificate granted! Cert is valid From: Wed Aug 24 15:49:30 2022 To: Wed Aug 24 15:51:00 2022.

```

Figure 38. Sample Erinyes Client Log.

K. SUMMARY

The Erinyes protocol is comprised of multiple modules that must each perform accurately to provide effective continuous authentication. A gradual testing design was implemented to ensure accurate results throughout the development phase. First, the keystroke dynamic LSTM model was subjected to an authentication test where the error rates of the model's predictions were compared as the number of enrolled users increased. Next, the mouse cursor movement LSTM underwent a similar testing harness, but the number of enrolled users was made static, and the number of enrolled gallery samples was made variable. The last LSTM test merged the feature embeddings from the keystroke and mouse cursor movement LSTMs and measured the authentication performance of the combined vectors. BE was tested by creating a vault using, randomly chosen, enrolled users. A test set, containing genuine test sets and imposter sets, was then used to measure authentication performance by attempting to retrieve the original key used to create the vault. Lastly, a full system integration test on separate threads was used to measure the

effectiveness of the overall Erinyes protocol. The integration testing harness mirrored the authentication phase of the Erinyes protocol.

Although our LSTM models were able to perform with a greater EER on their individual test harnesses than during their concatenated run, the dual modality allowed us to protect against biometric attacks as discussed in [26]. The resulting embeddings from the LSTM models were tested on the biometric encryption test harness, which performed with 0% FRR, 0.93% FAR and 98.15% accuracy, consistent with industry standards, as discussed in [28]. That behavior in performance carried over to the overall system integration test, which was measured during the authentication portion of the protocol. Results for this portion were similarly driven by the type of testing set it received but performed with 0% FAR, 2%FRR and 98% accuracy. These results indicate that the protocol can be used to achieve continuous authentication accurately.

V. CONCLUSION

The need for user authentication in the digital domain is paramount as the number of digital interactions that involve sensitive data continues to increase. We have proposed and developed a protocol to provide an alternative to password release systems and allow for remote continuous authentication services. This chapter summarizes the findings of our research, potential implications and proposes future work to improve on the framework initiated.

A. SUMMARY

We developed a protocol with a multi-threaded client-server architecture that enhances current security practices employed by IT systems. The Erinyes protocol leverages state of the art ML methods to transform a user's keystroke timings and mouse cursor movements into a vector of embeddings that is used to authenticate users. The output behavioral biometric embeddings are utilized to securely store and retrieve asymmetric cryptographic keys through biometric encryption. Across 5 users, the biometric encryption module yielded an average of 0.93% FAR, 0.00% FRR, 99.07% accuracy and the overall protocol yielded an average of 0% FAR, 2% FRR and 98% accuracy. Our results have demonstrated that a system that provides remote continuous authentication service using biometric encryption can be achieved.

B. CONCLUSIONS

The goals of our study were to: 1) determine the feasibility of behavioral biometric modalities, specifically keystroke timings and mouse cursor sequences, in a biometric encryption system, and 2) determine if such a system could be used for continuous authentication. The embedding vectors produced from the behavioral biometrics were of sufficient quality to enable user authentication. As presented in Chapter IV, when input through the LSTM models, the keystroke data, mouse data, and keystroke and mouse combined data produced accuracies of 95% and greater during testing. After a client enrolls biometric feature embeddings, and the client's public key is verified remotely by the server, the Erinyes server was able to deny authentication for imposters with near perfect accuracy

and authenticate users with 2% FRR and 98% accuracy. Thus, the current implementation of the Erinyes protocol performs well enough to provide a reliable continuous authentication service.

C. DISCUSSION

In addition to being a standalone remote continuous authentication service, the Erinyes protocol could be utilized within Zero Trust Architectures (ZTA). NIST defines ZTA as, “an end-to-end approach to enterprise resource and data security that encompasses identity (person and non-person entities), credentials, access management, operations, endpoints, hosting environments, and the interconnecting infrastructure” [44]. NIST continues the definition of ZTA by introducing 7 basic tenets of Zero Trust. The tenet most applicable to the employment of the Erinyes protocol states that, “Access to resources is determined by dynamic policy - including the observable state of client identity, application/service, and the requesting asset-and may include other behavioral and environmental attributes” [44].

Figure 39, provided by NIST, highlights two major components of ZTAs, the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP) [44]. The PDP can be further subdivided into a Policy Engine (PE) and a Policy Administrator (PA). The PE is, “responsible for the ultimate decision to grant access to a resource for a given subject,” while the PA is, “responsible for establishing and/or shutting down the communication path between a subject and a resource” [44]. The decision made by the PDP is then passed to the PEP, which will physically enable the connection between requesting client and resource [44]. The Erinyes protocol could provide a continuous authentication capability for a requesting client to the PDP, ultimately aiding in the PDPs access-granting calculus.

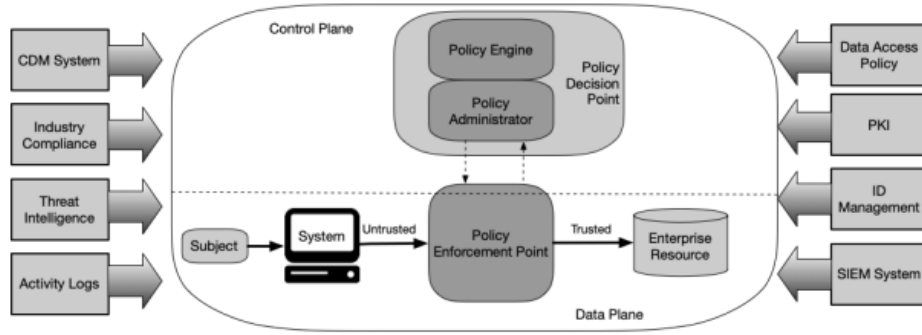


Figure 39. Zero Trust Architecture. Source: [44].

D. FUTURE WORK

We have created a framework upon which to build from. While our initial results look promising, there are many future work possibilities that could and should be explored. To start, our analysis was done using test data sets, which presents a few complications. One specific restriction that our analysis encountered was the limited size of the mouse cursor data set. For future work considerations, the Erinyes protocol should be expanded to include other biometric modalities, since other biometric datasets are more readily available. The protocol is designed such that any biometric feature that can be represented in a vectorized form can be used as input. Thus, Erinyes lends itself to inclusion of other biometrics such as iris or facial scans. The amplification of biometric inputs meshes well with the utilization of the protocol in ZTAs.

Another complication of using test data sets, is that it is difficult to find available datasets that have combined modalities attributed to the same user. For our analysis, a user was created by combining data from a keystroke embedding and a mouse cursor embedding. Obtaining raw samples across multiple modalities from the same user will ensure that any user-specific confounds, such as a correlation between slow typing and slow mouse cursor movements, will be able to be extracted by the ML techniques and used in authentication decisions.

Additionally, different ML architectures, such as transformers, could be explored to handle the sequential data of the raw biometrics. Although we designed our protocol

with security best practices in mind, an in-depth security analysis of the protocol should be conducted to find potential vulnerabilities prior to real-world deployment.

The last task to explore prior to real-world implementation should be an analysis of efficiency and performance. There is potential in the protocol to improve speed that we have not explored. Once these tasks have been completed and the protocol has improved speed performance, the implementation of the Erinyes protocol should be expanded to include a sample collection module that collects raw keystrokes and mouse cursor movements on the client software (and/or any other modality that is introduced into the implementation). With the sample collection module in place, a real-world user study can be undertaken, and the Erinyes protocol formally evaluated for real-world deployment.

APPENDIX A. ERINYES AUTHENTICATION CERTIFICATE

The certificates issued by the Erinyes server contain the verified public key of an authenticated client. Certificate validity encompasses the time of issuance to the time of issuance + the configured *auth_time* of the server. Figure 40 is an example of an issued certificate.

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
    5e:64:ad:a2:5d:59:9a:c1:fe:6a:c6:1d:80:ad:7d:0a:ba:21:4a:fb
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: C = US, ST = California, L = Monterey, O = Erinyes LLC, CN = Erinyes Protocol
  Validity
    Not Before: Aug 24 15:49:30 2022 GMT
    Not After : Aug 24 15:51:00 2022 GMT
  Subject: C = US, ST = California, L = Monterey, O = Erinyes LLC, CN = Erinyes Protocol
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public-Key: (1024 bit)
    Modulus:
      00:b1:2c:e3:85:0d:8a:72:cd:a9:66:21:ea:ed:5f:
      36:d0:2f:4f:26:50:bd:af:32:4d:3f:78:87:d3:05:
      60:77:87:3c:37:d9:85:0e:8b:95:01:b4:5c:45:b0:
      61:ea:85:10:4c:eb:ad:7e:e9:39:9d:cf:c4:c1:39:
      b8:20:6b:a8:d8:c0:f5:85:f1:55:b4:47:a8:18:a3:
      4b:82:26:9e:8e:4d:27:22:51:17:ee:4d:d5:fe:9d:
      17:c6:64:2c:0a:22:ba:4a:28:90:48:56:c6:2a:7e:
      ca:e2:6f:9c:ee:6d:fd:9c:5d:42:7d:2d:ac:8c:3f:
      e8:db:e5:f0:c5:91:63:16:9b
    Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Subject Alternative Name:
      DNS:localhost
  Signature Algorithm: sha256WithRSAEncryption
    6c:31:d1:63:f4:54:69:5b:26:16:8b:8b:bb:b1:69:60:a7:a6:
    4c:0b:52:1e:8c:9b:b7:38:8c:70:64:52:0f:0c:29:da:e8:2c:
    42:fa:71:7b:87:d3:2f:92:45:65:e7:4d:8f:1c:e0:9b:6e:dc:
    5b:73:7e:80:64:3f:b1:e3:86:15:d5:1f:66:ed:dd:a8:70:05:
    90:fa:2b:ef:2e:f7:bb:12:ea:66:6b:9c:93:42:42:73:84:13:
    32:e4:e6:9b:66:dd:6a:9b:d9:31:db:dd:cb:7d:34:e6:3d:00:
    1e:d5:c5:a9:3e:bf:5b:50:65:aa:7f:1e:ef:8e:ec:65:60:26:
    53:79
-----BEGIN CERTIFICATE-----
MIICbTCCAdagAwIBAgIUXmSto11ZmsH+asYdgK19CrohSvswDQYJKoZIhvcNAQEL
BQAwZjELMAkGA1UEBhMCVVMxEzARBgNVBAgMCKNhbG1mb3JuaWExETAPBgNVBAcM
CE1vbmlRlcmV5MRQwEgYDVQQKDAZFcmVudWVzIEExMQZEMBCGA1UEAwQXJpbnll
cyBQcm90b2NvbDAeFw0yMjA4MjQxNTQ5MzBaFw0yMjA4MjQxNTUxMDBaMGYxCzAJ
BgNVBAYTALVTRMRWEQYDVQQIDApDYWxpZm9ybmlhMREwDwYDVQQHDAhNb250ZXJl
eTEUMBIGA1UECgwLRXJpbnllcyBMTExGTAxBGNVBAMMEEVyaW55ZXMGUHVjdG9j
b2w2wGZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBALEs44UNinLNqWYh6u1fNtAv
TyZQva8yTT94h9MFYHeHPDfZhQ6LLQG0XEWwYeqFEEzrrX7p0Z3PxME5uCBRqNjA
9YXxVbRHqBijS4Imno5NjyJRF+5N1f6dF8ZkLAoiukookEhWxip+yuJvn05t/Zxd
Qn0trIw/6NvL8MWRyxabaGMBAAAgGDAAWMBQGA1UdEQNMMAuCCWxvY2FsaG9zdDAN
BgkqhkiG9w0BAQsFAA0BgQBSMdfj9FRpWYyWi4u7sWlgp6ZMC1IejJu30IXwZFIP
DCna6Cx+nF7h9MvkkVL502PH0Cbbtxbc36AZD+x44YV1R9m7d2ocAWQ+ivvLve7
Eupma5yTQkZzhBMys0abZt1qm9kx293LFTTmPQAe1cWpPr9bUGWqfx7vjuxLYCZT
eQ==
-----END CERTIFICATE-----
```

Figure 40. Erinyes Issued Certificate.

This certificate was issued to client (127.0.0.1, 52349) by the Erinyes Server on August 24th, 2022, at 15:49:30 GMT. The Erinyes server was configured with an *auth_time* of 90 seconds, thus the validity of the certificate ended on August 24th, 2022, at 15:5100 GMT. The subject's public key is the same key that has been verified and associated in the Erinyes server's user profile database. The certificate is signed with the Erinyes server's private key, and the hash algorithm utilized is listed.

APPENDIX B. SAMPLE ERINYES LOGS

The Erinyes server connection log displays information regarding the configuration of the Erinyes server, network status updates, and the launching and joining of connected client handler threads. Figure 41 is an example of an Erinyes server connection log. As shown in the provided connection log, the server was configured with an *auth_time* of 90 seconds and was bound to port 9999. Client (127.0.0.1, 52349) connected to the server at 2022-08-24 08:40:42,445 PST, and the thread was joined at 2022-08-24 09:16:02,997 PST.

```
2022-08-24 08:40:35,214 - Server Connections - INFO - Server configured with an auth_time of 90 seconds.
2022-08-24 08:40:35,215 - Server Connections - INFO - Server Socket is bound to PORT: 9999.
2022-08-24 08:40:35,215 - Server Connections - INFO - Server Socket is in LISTENING MODE.
2022-08-24 08:40:35,218 - Server Connections - INFO - Server User Profile Database was established.
2022-08-24 08:40:42,439 - Server Connections - INFO - Connected to: 127.0.0.1:52348.
2022-08-24 08:40:42,440 - Server Connections - INFO - Thread launched for: 127.0.0.1:52348.
2022-08-24 08:40:42,444 - Server Connections - INFO - Connected to: 127.0.0.1:52349.
2022-08-24 08:40:42,445 - Server Connections - INFO - Thread launched for: 127.0.0.1:52349.
2022-08-24 08:40:42,498 - Server Connections - INFO - Connected to: 127.0.0.1:52350.
2022-08-24 08:40:42,498 - Server Connections - INFO - Thread launched for: 127.0.0.1:52350.
2022-08-24 08:40:42,502 - Server Connections - INFO - Connected to: 127.0.0.1:52351.
2022-08-24 08:40:42,503 - Server Connections - INFO - Thread launched for: 127.0.0.1:52351.
2022-08-24 08:40:42,504 - Server Connections - INFO - Connected to: 127.0.0.1:52352.
2022-08-24 08:40:42,505 - Server Connections - INFO - Thread launched for: 127.0.0.1:52352.
2022-08-24 09:15:13,166 - Server Connections - INFO - Thread 0 has been joined.
2022-08-24 09:16:02,997 - Server Connections - INFO - Thread 1 has been joined.
2022-08-24 09:16:02,998 - Server Connections - INFO - Thread 2 has been joined.
2022-08-24 09:16:02,998 - Server Connections - INFO - Thread 3 has been joined.
2022-08-24 09:16:02,998 - Server Connections - INFO - Thread 4 has been joined.
2022-08-24 09:16:02,998 - Server Connections - WARNING - SERVER SHUTTING DOWN! All threads have successfully exited.
```

Figure 41. Erinyes Server Log Connections.

Figures 42 and 43 show the Erinyes server log for client (127.0.0.1, 52349), which displays information from all phases of the Erinyes protocol during the server/client (127.0.0.1, 5234952349) session. The Erinyes client log for client (127.0.0.1, 5234952349), shown in Figures 44 and 45, captures information from the same session, but from the client's perspective. The validity period for any certificates granted by the server to the authentication client will match the timestamps for certificate issuance in the server and client logs.

```

2022-08-24 08:40:42,445 - server_({'127.0.0.1', 52349}) - INFO - Client number retrieved from connecting client: WedAug2408:40:4202255105.
2022-08-24 08:40:42,527 - server_({'127.0.0.1', 52349}) - INFO - Encrypted enrollment message received from client.
2022-08-24 08:40:43,219 - server_({'127.0.0.1', 52349}) - INFO - Enrollment message successfully decrypted.
2022-08-24 08:40:43,219 - server_({'127.0.0.1', 52349}) - INFO - Enrollment message nonce passed replay check. WedAug2408:40:42202217483 was added to the Nonce Cache.
2022-08-24 08:40:43,224 - server_({'127.0.0.1', 52349}) - INFO - Server SUCCESSFULLY completed the enrollment phase. 11947 was sent back to client.
2022-08-24 08:41:04,958 - server_({'127.0.0.1', 52349}) - INFO - Encrypted verification message received from client.
2022-08-24 08:44:49,487 - server_({'127.0.0.1', 52349}) - INFO - Verification message successfully decrypted.
2022-08-24 08:44:49,515 - server_({'127.0.0.1', 52349}) - INFO - Verification message nonce passed replay check. WedAug2408:40:48202231726 was added to the Nonce Cache.
2022-08-24 08:46:01,668 - server_({'127.0.0.1', 52349}) - INFO - Client Public Key Verified
2022-08-24 08:46:02,703 - server_({'127.0.0.1', 52349}) - INFO - Server SUCCESSFULLY completed the verification phase.
2022-08-24 08:46:05,280 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 0: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 08:46:05,298 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 08:46:34,964 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2408:46:3320222918 was added to the Nonce Cache.
2022-08-24 08:47:32,711 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 08:47:34,391 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 0 completed.
2022-08-24 08:47:34,407 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 1: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 08:48:00,334 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 08:48:01,859 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2408:47:59202249355 was added to the Nonce Cache.
2022-08-24 08:49:02,728 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 08:49:04,494 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 1 completed.
2022-08-24 08:49:04,513 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 2: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 08:49:30,148 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 08:49:30,170 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2408:49:2820224599 was added to the Nonce Cache.
2022-08-24 08:49:33,642 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message successfully decrypted.
2022-08-24 08:50:32,743 - server_({'127.0.0.1', 52349}) - INFO - Client signature was VERIFIED. VALID CERTIFICATE ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 08:50:34,447 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 2 completed.
2022-08-24 08:50:34,447 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 3: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 08:51:00,830 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 08:51:01,114 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2408:50:59202213830 was added to the Nonce Cache.
2022-08-24 08:52:02,771 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 08:52:04,323 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 3 completed.
2022-08-24 08:52:04,323 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 4: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 08:52:30,807 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 08:52:31,122 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2408:52:28202251001 was added to the Nonce Cache.
2022-08-24 08:53:32,782 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 08:53:34,835 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 4 completed.
2022-08-24 08:53:34,835 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 5: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 08:54:00,493 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 08:54:01,290 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2408:53:59202239168 was added to the Nonce Cache.
2022-08-24 08:55:02,807 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 08:55:04,516 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 5 completed.
2022-08-24 08:55:04,516 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 6: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 08:55:30,504 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 08:55:34,851 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2408:55:2920227510 was added to the Nonce Cache.
2022-08-24 08:56:32,833 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message successfully decrypted.
2022-08-24 08:56:34,953 - server_({'127.0.0.1', 52349}) - INFO - Client signature was VERIFIED. VALID CERTIFICATE ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 08:56:34,953 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 6 completed.
2022-08-24 08:57:00,878 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 7: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 08:57:01,854 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 08:58:02,890 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2408:56:59202215257 was added to the Nonce Cache.
2022-08-24 08:58:05,145 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 08:58:05,146 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 7 completed.
2022-08-24 08:58:31,690 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 8: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 08:58:32,569 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 08:59:32,907 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2408:58:30202257123 was added to the Nonce Cache.
2022-08-24 08:59:35,699 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 08:59:35,718 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 8 completed.
2022-08-24 09:00:01,847 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 9: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:00:02,779 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:01:02,934 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:00:01202244470 was added to the Nonce Cache.
2022-08-24 09:01:06,992 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:01:06,992 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 9 completed.
2022-08-24 09:01:33,111 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 10: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:01:35,989 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:02:32,948 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:01:32202240323 was added to the Nonce Cache.
2022-08-24 09:02:39,209 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:02:39,231 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 10 completed.
2022-08-24 09:03:05,786 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 11: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:03:06,561 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:03:06,561 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:03:04202250980 was added to the Nonce Cache.
2022-08-24 09:03:06,561 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).

```

Figure 42. Erinyes Server Log Part 1 ('127.0.1', 52349).

```

2022-08-24 09:04:02,946 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 11 completed.
2022-08-24 09:04:14,267 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 12: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:04:14,291 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:04:41,231 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:04:40202244083 was added to the Nonce Cache.
2022-08-24 09:04:41,905 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:05:32,955 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 12 completed.
2022-08-24 09:05:52,498 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 13: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:05:52,498 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:06:20,164 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:06:19202255218 was added to the Nonce Cache.
2022-08-24 09:06:20,170 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message successfully decrypted.
2022-08-24 09:06:20,676 - server_({'127.0.0.1', 52349}) - INFO - Client signature was VERIFIED. VALID CERTIFICATE ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:07:02,965 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 13 completed.
2022-08-24 09:07:40,840 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 14: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:07:40,840 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:08:15,701 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:08:1520227154 was added to the Nonce Cache.
2022-08-24 09:08:15,705 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:08:32,981 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 14 completed.
2022-08-24 09:09:13,969 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 15: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:09:13,969 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:09:52,018 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:09:52202212302 was added to the Nonce Cache.
2022-08-24 09:09:52,022 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:10:02,986 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 15 completed.
2022-08-24 09:10:45,266 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 16: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:10:45,266 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:11:21,773 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:11:21202235644 was added to the Nonce Cache.
2022-08-24 09:11:21,775 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:11:32,984 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 16 completed.
2022-08-24 09:12:18,426 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 17: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:12:18,426 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:12:54,219 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:12:54202234031 was added to the Nonce Cache.
2022-08-24 09:12:54,222 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:13:02,982 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 17 completed.
2022-08-24 09:13:50,264 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 18: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:13:50,265 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:14:19,423 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:14:19202240739 was added to the Nonce Cache.
2022-08-24 09:14:19,423 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:14:32,994 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 18 completed.
2022-08-24 09:15:14,943 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 19: server sent auth_request to client: ('127.0.0.1', 52349).
2022-08-24 09:15:14,943 - server_({'127.0.0.1', 52349}) - INFO - Encrypted auth_reply message received from client.
2022-08-24 09:15:35,847 - server_({'127.0.0.1', 52349}) - INFO - Auth_reply message nonce passed replay check. WedAug2409:15:3520225994 was added to the Nonce Cache.
2022-08-24 09:15:35,850 - server_({'127.0.0.1', 52349}) - INFO - Client FAILED verification. NO CERT WAS ISSUED TO CLIENT: ('127.0.0.1', 52349).
2022-08-24 09:16:02,992 - server_({'127.0.0.1', 52349}) - INFO - T_cycle 19 completed.
2022-08-24 09:16:02,992 - server_({'127.0.0.1', 52349}) - WARNING - Server closed connection to client: ('127.0.0.1', 52349).

```

Figure 43. Erinyes Server Log Part 2 ('127.0.1', 52349).

```

2022-08-24 08:40:42,446 - client_({'127.0.0.1', 52349}) - INFO - Client successfully connected to Erinyes Server.
2022-08-24 08:40:42,446 - client_({'127.0.0.1', 52349}) - INFO - ENROLLMENT PHASE: CLIENT SIDE INITIATED.
2022-08-24 08:40:42,446 - client_({'127.0.0.1', 52349}) - INFO - Gallery embeddings SUCCESSFULLY generated.
2022-08-24 08:40:42,526 - client_({'127.0.0.1', 52349}) - INFO - Encrypted enrollment message sent to server.
2022-08-24 08:40:43,224 - client_({'127.0.0.1', 52349}) - INFO - Encrypted enrollment ack received from server.
2022-08-24 08:40:43,230 - client_({'127.0.0.1', 52349}) - INFO - Enrollment ack nonce passed replay check. WedAug2408:40:43202221335 was added to the Nonce Cache.
2022-08-24 08:40:43,230 - client_({'127.0.0.1', 52349}) - INFO - UserId: 11847 was obtained from the server.
2022-08-24 08:40:43,230 - client_({'127.0.0.1', 52349}) - INFO - ENROLLMENT PHASE: CLIENT-SIDE COMPLETED.
2022-08-24 08:40:43,230 - client_({'127.0.0.1', 52349}) - INFO - BE KEY BINDING PHASE INITIATED.
2022-08-24 08:40:47,611 - client_({'127.0.0.1', 52349}) - INFO - New set of crypto keys were used to generate BE Public and BE Private key pair.
2022-08-24 08:40:47,611 - client_({'127.0.0.1', 52349}) - INFO - BE KEY BINDING PHASE COMPLETED.
2022-08-24 08:40:47,611 - client_({'127.0.0.1', 52349}) - INFO - CLIENT PUBLIC KEY VERIFICATION PHASE: CLIENT-SIDE INITIATED.
2022-08-24 08:40:47,611 - client_({'127.0.0.1', 52349}) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:41:04,921 - client_({'127.0.0.1', 52349}) - INFO - Encrypted verification message sent to server.
2022-08-24 08:46:02,704 - client_({'127.0.0.1', 52349}) - INFO - Encrypted verification ack received from server.
2022-08-24 08:46:02,712 - client_({'127.0.0.1', 52349}) - INFO - Verification ack: 11947Client Public Key Verified successfully decrypted.
2022-08-24 08:46:02,712 - client_({'127.0.0.1', 52349}) - INFO - Verification ack nonce passed replay check. WedAug2408:46:0120226336 was added to the Nonce Cache.
2022-08-24 08:46:02,712 - client_({'127.0.0.1', 52349}) - INFO - CLIENT PUBLIC KEY VERIFICATION PHASE: CLIENT-SIDE COMPLETED.
2022-08-24 08:46:05,252 - client_({'127.0.0.1', 52349}) - INFO - AUTHENTICATION: CLIENT-SIDE INITIATED.
2022-08-24 08:46:05,252 - client_({'127.0.0.1', 52349}) - INFO - Auth_req received from server.
2022-08-24 08:46:05,252 - client_({'127.0.0.1', 52349}) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:46:31,748 - client_({'127.0.0.1', 52349}) - INFO - BE Key Retrieval FAILED.
2022-08-24 08:46:35,748 - client_({'127.0.0.1', 52349}) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:46:35,764 - client_({'127.0.0.1', 52349}) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 08:47:34,392 - client_({'127.0.0.1', 52349}) - INFO - Auth_req received from server.
2022-08-24 08:47:34,392 - client_({'127.0.0.1', 52349}) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:47:57,190 - client_({'127.0.0.1', 52349}) - INFO - BE Key Retrieval FAILED.
2022-08-24 08:47:59,195 - client_({'127.0.0.1', 52349}) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:48:01,048 - client_({'127.0.0.1', 52349}) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 08:49:04,494 - client_({'127.0.0.1', 52349}) - INFO - Auth_req received from server.
2022-08-24 08:49:04,494 - client_({'127.0.0.1', 52349}) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:49:26,964 - client_({'127.0.0.1', 52349}) - INFO - BE Key Retrieval was SUCCESSFUL: auth_req was decrypted.
2022-08-24 08:49:26,964 - client_({'127.0.0.1', 52349}) - INFO - Auth_req nonce passed replay check. WedAug2408:49:04202240107 was added to the Nonce Cache.
2022-08-24 08:49:28,978 - client_({'127.0.0.1', 52349}) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:49:33,022 - client_({'127.0.0.1', 52349}) - INFO - Cert issue message nonce passed replay check. WedAug2408:49:3120224511 was added to the Nonce Cache.
2022-08-24 08:49:33,026 - client_({'127.0.0.1', 52349}) - INFO - Erinyes Certificate granted! Cert is valid From: Wed Aug 24 15:49:30 2022 To: Wed Aug 24 15:51:00 2022.
2022-08-24 08:50:34,429 - client_({'127.0.0.1', 52349}) - INFO - Auth_req received from server.
2022-08-24 08:50:34,429 - client_({'127.0.0.1', 52349}) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:50:57,088 - client_({'127.0.0.1', 52349}) - INFO - BE Key Retrieval FAILED.
2022-08-24 08:50:59,090 - client_({'127.0.0.1', 52349}) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:51:01,115 - client_({'127.0.0.1', 52349}) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 08:52:04,323 - client_({'127.0.0.1', 52349}) - INFO - Auth_req received from server.
2022-08-24 08:52:04,323 - client_({'127.0.0.1', 52349}) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:52:26,917 - client_({'127.0.0.1', 52349}) - INFO - BE Key Retrieval FAILED.
2022-08-24 08:52:28,924 - client_({'127.0.0.1', 52349}) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:52:31,092 - client_({'127.0.0.1', 52349}) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 08:53:34,835 - client_({'127.0.0.1', 52349}) - INFO - Auth_req received from server.
2022-08-24 08:53:34,835 - client_({'127.0.0.1', 52349}) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:53:57,444 - client_({'127.0.0.1', 52349}) - INFO - BE Key Retrieval FAILED.
2022-08-24 08:53:59,446 - client_({'127.0.0.1', 52349}) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:54:01,285 - client_({'127.0.0.1', 52349}) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 08:55:04,498 - client_({'127.0.0.1', 52349}) - INFO - Auth_req received from server.
2022-08-24 08:55:27,354 - client_({'127.0.0.1', 52349}) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:55:27,355 - client_({'127.0.0.1', 52349}) - INFO - BE Key Retrieval was SUCCESSFUL: auth_req was decrypted.
2022-08-24 08:55:29,359 - client_({'127.0.0.1', 52349}) - INFO - Auth_req nonce passed replay check. WedAug2408:55:04202210695 was added to the Nonce Cache.
2022-08-24 08:55:34,849 - client_({'127.0.0.1', 52349}) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:55:34,849 - client_({'127.0.0.1', 52349}) - INFO - Cert issue message nonce passed replay check. WedAug2408:55:31202250406 was added to the Nonce Cache.
2022-08-24 08:55:34,849 - client_({'127.0.0.1', 52349}) - INFO - Erinyes Certificate granted! Cert is valid From: Wed Aug 24 15:55:30 2022 To: Wed Aug 24 15:57:00 2022.
2022-08-24 08:56:34,924 - client_({'127.0.0.1', 52349}) - INFO - Auth_req received from server.
2022-08-24 08:56:34,924 - client_({'127.0.0.1', 52349}) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:56:57,816 - client_({'127.0.0.1', 52349}) - INFO - BE Key Retrieval FAILED.
2022-08-24 08:56:59,823 - client_({'127.0.0.1', 52349}) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:57:01,836 - client_({'127.0.0.1', 52349}) - INFO - Erinyes Certificate WAS NOT ISSUED!

```

Figure 44. Erinyes Client Log Part 1 ('127.0.0.1', 52349).

```

2022-08-24 08:58:05,133 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 08:58:05,133 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:58:05,378 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 08:58:30,385 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 08:58:32,528 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 08:59:35,699 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 08:59:35,699 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 08:59:59,017 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:00:01,022 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:00:02,761 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:01:06,973 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:01:06,974 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:01:30,283 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:01:31,291 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:01:33,980 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:02:39,199 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:02:39,199 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:03:02,765 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:03:04,771 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:03:06,527 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:04:14,237 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:04:14,237 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:04:38,451 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:04:40,457 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:04:41,986 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:05:52,498 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:05:52,498 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:06:17,825 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval was SUCCESSFUL: auth_req was decrypted.
2022-08-24 09:06:17,825 - client ('127.0.0.1', 52349) - INFO - Auth_req nonce passed replay check. WedAug2409:05:3420222515 was added to the Nonce Cache.
2022-08-24 09:06:19,835 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:06:20,713 - client ('127.0.0.1', 52349) - INFO - Cert issue message nonce passed replay check. WedAug2409:06:20202241478 was added to the Nonce Cache.
2022-08-24 09:06:20,713 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate granted! Cert is valid From: Wed Aug 24 16:06:20 2022 To: Wed Aug 24 16:07:50 2022.
2022-08-24 09:07:40,835 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:07:40,835 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:08:13,687 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:08:15,692 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:08:15,707 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:09:13,969 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:09:13,969 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:09:50,000 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:09:52,008 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:09:52,028 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:10:45,270 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:10:45,271 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:11:19,757 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:11:21,763 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:11:21,777 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:12:18,429 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:12:18,429 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:12:52,207 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:12:54,212 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:12:54,224 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:13:50,269 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:13:50,269 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:14:17,407 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:14:19,415 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:14:19,425 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:15:14,951 - client ('127.0.0.1', 52349) - INFO - Auth_req received from server.
2022-08-24 09:15:14,951 - client ('127.0.0.1', 52349) - INFO - Raw keystroke and raw mouse samples obtained.
2022-08-24 09:15:35,821 - client ('127.0.0.1', 52349) - INFO - BE Key Retrieval FAILED.
2022-08-24 09:15:35,838 - client ('127.0.0.1', 52349) - INFO - Encrypted Auth_reply sent to server.
2022-08-24 09:15:35,852 - client ('127.0.0.1', 52349) - INFO - Erinyes Certificate WAS NOT ISSUED!
2022-08-24 09:16:02,993 - client ('127.0.0.1', 52349) - WARNING - Server closed connection to client: ('127.0.0.1', 52349).
2022-08-24 09:16:02,994 - client ('127.0.0.1', 52349) - INFO - Metrics for this round of Erinyes protocol testing can be found at: /Users/gagehathaway/Desktop/NPS/Thesis/dev/code/erinyes/metrics/('127.0.0.1', 52349)/erinyes_metrics_2022-08-24 16:16:02.993528.txt.

```

Figure 45. Erinyes Client Log Part 2 ('127.0.0.1', 52349).

LIST OF REFERENCES

- [1] J. Amankwah-Amoah, Z. Khan, G. Wood, and G. Knight, “Covid-19 and digitalization: The great acceleration,” *Journal of Business Research*, August 11, 2021 [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8437806/>
- [2] DocuSign, “Digital Transactions: What are they?,” August 20, 2017 [Online]. Available: <https://www.docusign.com/learn/digital-transactions>
- [3] World Bank Group, “Covid-19 drives global surge in use of digital payments,” June 28, 2022 [Online]. Available: <https://www.worldbank.org/en/news/press-release/2022/06/29/covid-19-drives-global-surge-in-use-of-digital-payments>
- [4] Statista, “Digital payments - worldwide: Statista market forecast” [Online]. Available: <https://www.statista.com/outlook/dmo/fintech/digital-payments/worldwide>
- [5] U. Uludag, S. Pankanti, S. Prabhakar, and A. . Jain, “Biometric cryptosystems: issues and challenges,” in *2004 Proceedings of the IEEE*, vol. 92, no. 6, pp. 948–960, 2004 [Online]. Available: <https://doi.org/10.1109/JPROC.2004.827372>
- [6] D. Thakkar, “Biometric template - systems, components and Security,” Bayometric, July 22, 2020 [Online]. Available: <https://www.bayometric.com/biometric-template-security/>
- [7] Biometrics Evaluation and Testing (BEAT), “Biometrics evaluation and testing” [Online]. Available: <https://www.idiap.ch/webarchives/sites/beat-eu.org/>
- [8] Vandana and N. Kaur, “A Study of Biometric Identification and Verification System,” in *2021 ICACITE*, 2021, pp. 60–64. [Online]. Available: <https://doi.org/10.1109/ICACITE51222.2021.9404735>
- [9] C. Soutar, D. Roberge, A. Stoianov, R. Gilroy, and B. Vijaya Kumar, “Biometric Encryption using image processing,” in *Proceedings of SPIE*, 1998, vol. 3314, no. 1, 1998, pp. 178–188. [Online]. Available: <https://doi.org/10.1117/12.304705>
- [10] A. T. B. Jin and L. M. Hui, “Cancelable biometrics,” *Scholarpedia*, January 4, 2010. [Online]. Available: http://www.scholarpedia.org/article/Cancelable_biometrics
- [11] D. Arias, “Adding salt to hashing: A better way to store passwords,” *auth0*, February 25, 2021 [Online]. Available: <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>

- [12] A. T. B. Jin, D. N. C. Ling, and A. Goh, "Biohashing: two factor authentication featuring fingerprint data and tokenised random number," *Pattern Recognition*, vol. 37, no. 11, pp. 2245–2255, Nov. 2004 [Online]. Available: <https://doi.org/10.1016/j.patcog.2004.04.011>
- [13] N. Ratha, S. Chikkerur, J. . Connell, and R. Bolle, "Generating Cancelable Fingerprint Templates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 4, pp. 561–572, Feb. 2007 [Online]. Available: <https://doi.org/10.1109/TPAMI.2007.1004>
- [14] F. Quan, S. Fei, C. Anni, and Z. Feifei, "Cracking Cancelable Fingerprint Template of Ratha," *2008 International Symposium on Computer Science and Computational Technology*, 2008, vol. 2, pp. 572–575, Feb. 2007 [Online]. Available: <https://doi.org/10.1109/TPAMI.2007.1004>
- [15] S. W. Shin, M. Lee, D. Moon, and K. Moon, "Dictionary Attack on Functional Transform-Based Cancelable Fingerprint Templates," *ETRI journal*, vol. 31, no. 5, pp. 628–630, Oct. 2009 [Online]. Available: <https://doi.org/10.4218/etrij.09.0209.0137>
- [16] S. Tulyakov, F. Farooq, and V. Govindaraju, "Symmetric Hash Functions for Fingerprint Minutiae," *Lecture Notes in CS*, 2005, pp. 30–38. [Online]. Available: https://doi.org/10.1007/11552499_4
- [17] E. Maiorana, P. Campisi, J. Fierrez, J. Ortega-Garcia, and A. Neri, "Cancelable Templates for Sequence-Based Biometrics with Application to On-line Signature Recognition," *IEEE Transactions on Systems, Man and Cybernetics. Part A, Systems and Humans*, vol. 40, no. 3, pp. 525–538, May 2010 [Online]. Available: <https://doi.org/10.1109/TSMCA.2010.2041653>
- [18] G. Tomko, C. Soutar, and G. Schmidt, 2022."Fingerprint controlled public key cryptographic system," U.S. Patent 5541994, July 30, 1996 [Online]. Available: <https://uspto.report/patent/grant/5541994>
- [19] A. Cavoukian, and A. Stoianov, "Encryption, Biometric," *S.Z. Li, A. K. Jain (eds) Encyclopedia of Biometrics*. Springer, Boston, MA, July 3, 2015 [Online]. Available: https://doi.org/10.1007/978-1-4899-7488-4_63
- [20] M. Drahanský, "Hand-Based Biometrics - Methods and Technology - 5.2 History" *The IET*, 2018 [Online]. Available: <https://doi-org./10.1049/PBSE008E>
- [21] I. Azhar, M. Raza, M. Sharif, S. Kadry, and S. Rho, "Union is Strength: Improving face sketch synthesis by fusing outcomes of Fully-Convolutional-Networks and Random Sampling Locality Constraint," *Alexandria Engineering Journal*, vol. 61, no. 12, pp. 10727–10741, Dec. 2022 [Online]. Available: <https://doi-org.10.1016/j.aej.2022.04.007>

- [22] S. McCulley and V. Roussev, “Latent Typing Biometrics in Online Collaboration Services,” in *Proceedings of 2018 ACSAC*, Article 4, 2018 [Online]. Available: <https://doi.org/10.1145/3274694.3274754>.
- [23] A. Srivastava, “Google leads €6.2M series A funding for TypingDNA, which authenticates people based on their typing behaviour,” *Silicon Canals*, January 20, 2020 [Online]. Available: <https://www.siliconcanals.com/news/google-leads-funding-typingdna-authenticate-typing-behaviour/>
- [24] A. Morales, J. Fierrez, R. Tolosana, J. Ortega-Garcia, J. Galbally, M. Gomez-Barrero, A. Anjos, S. Marcel, “Keystroke Biometrics Ongoing Competition,” *IEEE access*, vol. 4, pp. 7736–7746, Nov.2016 [Online]. Available: <https://doi.org/10.1109/ACCESS.2016.2626718>
- [25] Allied Market Research, “Keystroke Dynamics Market to Reach \$754.9 Mn, Globally, by 2025 at 24.7% CAGR Says AMR,” February 6, 2019 [Online]. Available: <https://www.prnewswire.com/news-releases/keystroke-dynamics-market-to-reach-754-9-mn-globally-by-2025-at-24-7-cagr-says-amr-300790697.html>
- [26] D. Deb, A. Ross, A. K. Jain, K. Prakah-Asante, and K. V. Prasad, “Actions Speak Louder Than (Pass)words: Passive Authentication of Smartphone Users via Deep Temporal Features,” *2019 ICB*, pp. 1–8, Jun. 2019 [Online]. Available: <https://doi.org/10.1109/ICB45273.2019.8987433>
- [27] C. Olah, “*Understanding LSTM Networks*,” Colah’s Blog, blog, August 27, 2015 [Online]. Available: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [28] A. Acien, J. V. Monaco, A. Morales, R. Vera-Rodríguez, and J. Fierrez, “TypeNet: Scaling up Keystroke Biometrics,” in *IAPR/IEEE IJCB*, 2020 [Online]. Available: <https://doi.org/10.48550/arXiv.2004.03627>
- [29] J. G. Steiner, C. Neuman, and J. I. Schiller, “Kerberos: An Authentication Service for Open Network Systems,” in *USENIX Conference Proceedings*, 1988, pp. 191–202.
- [30] ManageEngine, “Windows event ID 4769 - A kerberos service ticket was requested” [Online]. Available: <https://www.manageengine.com/products/active-directory-audit/kb/windows-security-log-event-id-4769.html>
- [31] J. Petters, “Kerberos authentication explained,” Varonis, March 29, 2020 [Online]. Available: <https://www.varonis.com/blog/kerberos-authentication-explained#:~:text=Is%20Kerberos%20Obsolete%3F,protect%20passwords%20and%20authentication%20tickets>

- [32] J. Petters, “Kerberos attack: How to stop golden tickets?,” Varonis, March 29, 2020 [Online]. Available: <https://www.varonis.com/blog/kerberos-how-to-stop-golden-tickets>
- [33] M. Soria-Machado, D. Abolins, C. Boldea, K. Socha, “Protection from Kerberos Golden Ticket,” CERT-EU, 2014-007 ver 1.4, 2016 [Online]. Available: https://cert.europa.eu/static/WhitePapers/UPDATED%20-%20CERT-EU_Security_Whitepaper_2014-007_Kerberos_Golden_Ticket_Protection_v1_4.pdf
- [34] A. Green, “In Search of Kerberos’s golden ticket,” Varonis, September 5, 2014 [Online]. Available: <https://www.varonis.com/blog/search-kerberos-golden-ticket>
- [35] D. C. Neuman, S. Hartman, K. Raeburn, and T. Yu, “The Kerberos Network Authentication Service (V5),” RFC 4120, July 2005 [Online]. Available: <https://doi.org/10.17487/RFC4120>
- [36] V. Dhakal, A. M. Feit, P. O. Kristensson, and A. Oulasvirta, “Observations on typing from 136 million keystrokes,” in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018. [Online]. Available: <https://doi.org/10.1145/3173574.3174220>
- [37] TensorFlow. “Tensorflow addons losses: Tripletsemihardloss” [Online]. Available: https://www.tensorflow.org/addons/tutorials/losses_triplet
- [38] D. Chuda and P. Kratky, “Grouping instances in kNN for Classification Based on Computer Mouse Features” in *CompSysTech ‘15 Proceedings of the 16th International Conference on Computer Systems and Technologies*, 2015, pp. 214–220. [Online]. Available: <https://doi.org/10.1145/2812428.2812454>
- [39] A. Juels and M. Sunden, “A fuzzy vault scheme,” in *Proceedings of the IEEE International Symposium on Information Theory*, Lausanne, Switzerland, 2002, p. 408. [Online]. Available: <https://doi.org/10.1109/ISIT.2002.1023680>
- [40] A. Nasiri, F. Mahmood, and M. Z. Lighvan, “A New Approach to Alignment-Free Fingerprint Cryptosystem Using Fuzzy Vaults” *Journal of Electrical Systems and Signals*, Vol. 2, No. 2, Nov. 2015 [Online]. Available: <http://dx.doi.org/10.22067/ess.v2i2.47103>
- [41] M. Yasuda, and Y. Sugimura, “Biometric key-binding using lattice masking,” *Security Comm. Networks*, vol. 8, pp. 3405–3414, Nov. 2015 [Online]. Available: <https://doi.org/10.1002/sec.1267>

- [42] L. Wu, P. Xiao, S. Yuan, S. Jiang, and C. W. Chen, “A Fuzzy Vault Scheme for Ordered Biometrics” *JCM*, vol. 6., pp. 682–690, Dec. 2011 [Online]. Available: https://www.researchgate.net/publication/220520447_A_Fuzzy_Vault_Scheme_for_Ordered_Biometrics
- [43] Cryptography. “Use of openssl,” [Online]. Available: <https://cryptography.io/en/latest/openssl/>.
- [44] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, “Zero Trust Architecture. Special Publication (NIST SP),” National Institute of Standards, Gaithersburg, MD, August 2020. [Online]. Available: <https://doi.org/10.6028/NIST.SP.800-207>

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California