Engineering Gesture-Based Authentication Systems

Gestures are a topic of increasing interest in authentication, but successfully implementing them as a security layer requires reliable gesture recognition. This survey presents and analyzes different methods of gesture recognition and offers design considerations for gesture-based authentication systems.

Gradeigh D. Clark and Janne Lindqvist
Rutgers University

Authentication has become an essential component in daily life. Increasingly, it's the gateway to critical facets of the human experience including work, communication, and entertainment. To be effective, any authentication technique must be reliable, difficult to compromise, and, above all, easy to use when people are focused on the activity behind the gateway and not the authentication itself.

Gesture-based methods have advantages over currently popular authentication methods—such as text entry, PINs, biometrics—because gestures can be performed faster and are highly customizable,[1] easier to remember,[2] and potentially more secure.[1] Gestures also require lower concentration and accuracy compared to other methods, and thus have potentially lower chances of error when used by stressed or distracted people. For example, an incorrectly entered text-based password yields an automatic rejection, whereas some inaccuracy or deviation of the gesture password can still lead to a positive identification.

The difference between a recognizer and an authentication system is important here; the recognizer is one aspect of an authentication system, which can consist of several components (including the user interface). To the best of our knowledge, no comprehensive surveys of different recognition methods for gestures are available. Even more importantly, no critical discussion exists on how we might compare these proposals or use them to design robust and highly usable authentication systems. Prior studies have looked at basic issues, such as asking participants to generate "secure and memorable" gesture passwords with no other instructions.[1] However, gesture authentication is now at a stage where such work must be supplemented by a deeper understanding of usability and effectiveness. This isn't possible until a large, open dataset is created that will allow direct comparison of different gesture recognizing methods.

Until such a dataset is available, we can qualitatively analyze different approaches to gesture security. To this end, we offer here four contributions: a survey of common gesture recognizers; design considerations for gesture-based authentication systems; suggestions for comparing recognizers; and an evaluation of gesture-based authentication compared to text-based passwords.

Gesture Types

No universally accepted terminology exists for gesture types. Often, different names are used for the same type of gesture. From a top-level

Published by the IEEE CS ■ 1536-1268/15/$31.00 © 2015 IEEE

view, gestures are divisible into two categories: touchscreen gestures and motion gestures (see Figure 1). These two gesture classes can be freeform—that is, created without constraints and cues—or predefined by a recognizer's creator.

Touchscreen

Touchscreen gestures are those captured through a touchscreen. Single-stroke gestures use only one finger to perform a continuous input on the screen. Multistroke gestures are discontinuous and allow for multiple stroke attempts at the screen before completion. Multitouch gestures use more than one finger to perform a continuous gesture.

Motion Gestures

Motion gestures are performed in 3D and can be divided into sensor-based and camera-based gestures. Sensor-based gestures use sensors other than a camera or touchscreen (such as a smartphone's accelerometer). This division is motivated by the input techniques' challenges for recognizers, as well as the abundance of prior work. Camera-based methods represent the majority of gesture recognition publications, and thus warrant their own category.

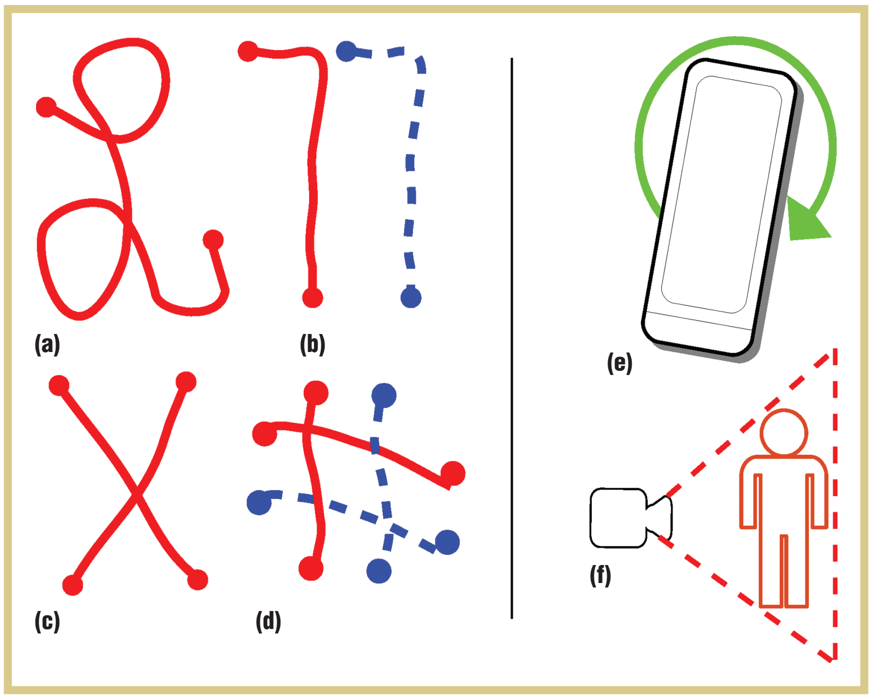


Figure 1. Different types of gestures. The example touchscreen gestures are (a) a single stroke gesture, (b) a multitouch gesture, (c) a multistroke gesture, and (d) a combination of multistroke and multitouch gestures (the hatch pattern can be drawn using two fingers and doing two strokes: one stroke with two fingers to get the horizontal portion, and a second stroke to form the vertical portion). Example motion gestures are (e) a sensor-based motion gesture, created by rotating a smartphone; and (f) a person being recorded for a camera-based motion gesture.

Threat Models: Attacks against Gestures

Authentication systems must be resilient against attacks. To successfully attack a gesture, an attacker must be capable of replicating the features accurately enough to fool the recognition algorithms into accepting the gesture as authentic.

A gesture password can be compromised in at least four ways: shoulder surfing, brute force, dictionary attacks, and storage leakage.

Shoulder Surfing

In shoulder surfing, an attacker tries to memorize a password or secret via line-of-sight. Basic shoulder surfing methods include the *standard* approach, in which the attacker observes the user from a vantage point that allows easy viewing of the user's gesture. Another option is *recording*, in which the attacker records and later observes the user's gesture. Finally, *multiple attackers* can work together from multiple vantage points to focus on specific parts of a password at different times and reconstruct it later.

Brute Force

Brute force attacking is done by repeatedly trying passwords to find the right one. A brute force approach can measure the susceptibility of a recognizer to algorithmic attack. As we discuss later, an equivalent attack on text-based passwords would be trying to guess the password without having access to password hashes.

Dictionary Attack

A dictionary attack is similar to a brute force attack except that the password attempts come from a set of more likely possibilities (such as datasets from user studies). To date, dictionary attacks have not been successfully demonstrated against gesture recognizers.

Storage Leakage

Storage leakage can be a problem depending on how the device stores the gesture password. To successfully steal a gesture based on stored data, a thief would have to know both the recognizer's structure and how to translate the stored values into gesture actions. Text-based passwords mitigate storage leakage by storing only the password's cryptographic hash. Storage leakage is a serious issue for gesture passwords given that no two inputs will be exactly alike, which makes comparing their hashes difficult.

Comparing Authentication Systems

Before discussing recognition in full, it's useful to outline aspects of gestures as an authentication scheme that make them viable as a replacement for (or supplement to) text-based passwords. The best way to do that is to evaluate gestures on three criteria:

- *usability* addresses a scheme's viability from the user's perspective,
- *deployability* examines the infrastructure a method requires to be usable, and
- *security* refers to the ability of a system to resist attacks.

These three metrics follow from an exhaustive survey of authentication methods,[3] which unfortunately doesn't discuss gestures. None of the known alternatives to text-based passwords offer the same range of features of such passwords,[3] and gestures are no exception. Decades of infrastructure and development have gone into making text-based passwords ubiquitous. Because of this, despite their many disadvantages, it's difficult for any scheme to match all the benefits of text-based passwords. However, advancing the development of alternative schemes could allow for a real challenger to text-based passwords.

Usability

Gestures are potentially more memorable than text-based passwords because human recall is better for pictorial concepts than for strings of text,[2] although no definitive measure exists for a password's memorability. We don't yet know whether complicated gesture passwords are more memorable than complicated text-based ones, but evidence from human psychology[2] and user studies[1] supports the assertion.

Gesture-based passwords are as easy to adopt as text-based passwords. Most people have used gestures to communicate silently or have drawn pictures to explain something to another person. As such, little additional training is required to teach people how to use them. The introduction of the Android 3 × 3 grid-based graphical password can be thought of as a primer to using gesture-based passwords on touchscreen devices.

Latency and error rates are natural usability related concerns; the former can be reduced with proper recognizer design, but error rates are harder to minimize. However, complicated gesture passwords could have better error rates than complicated text-based passwords.

Password recovery and the ability to reset passwords are necessary for usability. Gestures are equal to text-based passwords in this way—as we describe later, the same systems that recover text-based passwords can be applied to gesture passwords. The password recovery flow for gestures will be different than with text-based passwords because a gesture's features can be used to recover the password. A simple example would be to ask users to trace a set of characters and see how the result correlates to their past behaviors. Proper users can either be shown a picture of their gesture or given steps or hints as to how it can be replicated. The effort required to authenticate depends on the population under consideration. Given text-based passwords' ubiquity, user effort can be higher when starting out with gestures because more users are more comfortable using keyboards—though this might be less an issue for Android graphical password users. However, differently abled users (such as paraplegics) who can't naturally interact with a smartphone or touchscreen, or properly motion to a camera are at a disadvantage.

Deployability

With current technology, gesture passwords can't be used by all people who can use text-based passwords. The differently abled can face issues using gestures, especially if they suffer from a loss of sight or motor function.

Users don't require additional tools to authenticate. This doesn't mean that gestures are compatible with all current systems—rather that they integrate well into existing infrastructures. Although touchscreen tablets and phones are already prominent, laptops and monitors with touch capability are starting to become more commonplace as well. Alternatively, gestures could be generated with a mouse or using laptops' touch-interactive mousepads. A negative is that, for motion gestures, desktop users might require a separate webcam component.

Gestures aren't directly remote-login compatible; this is attributable to both their infancy as an authentication scheme and the proliferation of text-based passwords. However, some devices use biometrics as a master password, allowing integration with remote login servers. A gesture could be used as a master password in much the same way. This would improve remote-login compatibility, but it's not a perfect solution due to the inability to reliably compare the hash of two different gesture inputs. Gestures can be integrated into Web browsers. HTML5 or mobile websites can have a gesture capture area for touchscreen, while allowing a browser access to the camera would enable camera-based gestures. A mobile platform could observe a motion gesture on behalf of the browser. Using gestures might require additional hardware on desktops (such as a webcam or external sensors, but these are becoming default equipment for desktops, too).

Security

Gestures can be more resistant to shoulder surfing attacks than text-based passwords, depending on the amount of features used in recognition. Replication of the exact way a gesture is performed can be more difficult than assembling all the characters of a text-based password, depending on the password's length. Similar to biometric systems, personal knowledge doesn't yield clues that could reveal a user's gesture password. Comparing the security of text-based passwords to gesture passwords is an open problem. It's possible to quantify a gesture password's security based on a "surprisingness" factor.[1] This score allows for password creation policies similar to text-based passwords (such as rejecting simple passwords and instructing the user to try again).

If a system restricts the number of failed attempts, then it becomes difficult to compromise the password. As with text-based passwords, gesture passwords can be stolen if attempts are unlimited and the attacker is properly trained. The lower accuracy required for gestures—which is an advantage for usability—is a disadvantage here. This demonstrates the need for proper gesture-password creation policies like the ones used for text-based passwords.

Storage leakage is a problem because we currently lack a way to store gestures such that two similar inputs would have the same cryptographic hash. A hashing approach was explored in the Draw-A-Secret graphical password system,[4] where an input is a drawing over a 25-grid space. Inputs are compared by checking the order of grid boundaries that a drawing crosses (for example, "up in grid five, right in grid four") and concatenating those into a string. The hash function is then applied to this string. An approach like this does not work for free-form gestures, considering that the act of discretization causes a severe loss of information. This remains an open and important problem in gesture authentication and one that is well worth examining further.

Designing Recognizers for Authentication

Some design considerations can be gleaned from prior work on recognizers,[5] although such efforts focus on recognition, not authentication. Additionally, some aspects of reliable recognition[5]—such as location and scale invariance—don't translate to reliable authentication.

Our work extends previous research with eight design considerations for authentication systems.

Variable for Sampling

Different devices have different sampling rates. Additionally, natural variances occur in the speed and time with which a user performs gestures between authentication attempts. A recognizer should resample the input to obtain an accurate portrait of the gesture while keeping the number of samples constant.

Trainable

A good recognizer allows the design and learning to handle new inputs; it shouldn't use only predefined gestures. It should also differentiate between similar gestures (such as drawing a rectangle versus a square). Users must be allowed to create their own gestures to maximize usability and comfort while leveraging the full utility of the password space.

Adaptive

User behavior can change over time. For example, over time, users will likely perform their gesture password more rapidly. The recognizer should adapt to such changes. It should work through stored templates and features long after the initial training phase to figure out which templates are working and which ones aren't.

Computationally Efficient

When designing efficient recognizers, it's necessary to minimize the overall computation, memory, and delay that the algorithm introduces. The overall user experience is degraded if there's a noticeable pause with every login attempt.

Storage Conscious

Recognizers shouldn't make the system unusable by storing numerous templates or extracted features. The gestures should also be protected from theft by straightforward copying.

Configurable

A gesture recognizer should give users and developers options, such as control over the sampling rate and how many stored templates to use. Users should also be able to configure security settings based on their personal needs.

Attack Resistant

A recognizer must be efficient at rejecting false users. Gestures are represented as a collection of features. These features form layers that increase resistance to attacks. With more features, a recognizer increases its ability to exclude impostors. Examples of these features are pressure, speed, finger or arm length, body type, and path length. Recognizers with more than one layer can be considered attack resistant.

API Friendly

Although recognizers can be described in papers and with pseudocode, such descriptions might not be understandable to developers. A difficult, non-intuitive recognizer can have adoption issues. If implementing a recognizer is difficult, developers who might benefit from it—such as college students and those working for startups—would have trouble using it.

Common Gesture Recognition Approaches

A gesture recognizer uses algorithms to interpret human gestures. Designers have been creating new recognizers to accommodate neglected gesture types and support new features, which has led to numerous innovations in recognition for different gesture types. However, from a security engineering perspective, these efforts don't sufficiently consider authentication. Here, we analyze several popular algorithms; Table 1 summarizes how these algorithms correlate to our eight design principles.

TABLE 1
Design considerations for recognizers.

Recognizer	Features							
	Variable for sampling	Trainable	Adaptive	Computationally efficient	Storage conscious	Configurable	Attack resistant	API friendly
Geometric	Yes	Yes	Partial	Yes	Partial	Yes	Partial	Yes
Dynamic Time Warping	Yes	Yes	Partial	Yes	Partial	Partial	Partial	Partial
Hidden Markov model	Yes	Yes	Yes	Partial	No	Yes	Yes	No
Support vector machines	Yes	Yes	Yes	Partial	Partial	Yes	Yes	No
Wi-Fi	No	Partial	No	No	Yes	No	No	Partial
Capacitive	No	Partial	No	Yes	Yes	No	No	No
RFID	No	Partial	No	Yes	Yes	No	No	No

Geometric Methods

This family of recognizers performs distance-based comparisons on stored templates of coordinate pairs. For touchscreen gestures, the comparative measure is coordinate pairs in the plane; for motion gestures, the measure is either accelerometer or gyroscope data inside a 3D space.[6] The following discussion applies to several touchscreen recognizers, including \$1,[5] \$N,[7] and Protractor.[8] For motion gestures, we use the Protractor3D[6] extension. All four recognizers perform at least the first four of the following five steps:

1. The gesture is resampled to N points.
2. The resampled gesture is translated to the origin.
3. The size is normalized so the points are contained within a bounded cube.
4. The gesture is rotated until the angle that the sequence's first point makes with the gesture sequence's centroid is zero.
5. The gesture is iteratively rotated until an alignment is found that produces the optimal score with a given template.

A motion gesture recognizer doesn't need to consider rotation, so the fifth step above doesn't apply; instead, the concern is with the difference between successive accelerometer and gyroscope readings.

Geometric recognizers are sample-invariant and can be trained to identify new gestures. They are partially adaptable because they can store every successful attempt as a new template—which takes a toll on storage and efficiency—but the process also renders early templates meaningless. The algorithms are computationally efficient because templates are stored in a preprocessed form, requiring processing only for new attempts. Because they store little more than geometric features, but don't protect the data, they are partially storage conscious. They are configurable because they give users control over every step in the algorithm. Geometric recognizers are partially attack resistant because they have only one layer of features to breach. Finally, they are API friendly because they perform simple operations on coordinate pairs.

Dynamic Time Warping

Successive inputs tend to be mismatched because it's difficult for users to enter their gestures the exact same way every time. For example, in any given attempt, a user might round a corner more sharply or draw more slowly and carefully. From a top-level view, these attempts will often appear identical. Where they differ is in the details—one will have more sampling points than the other. The distribution of the inputs, when plotted against time, will often be very similar to each other; the difference is that they might appear as time-shifted versions of each other.

Dynamic Time Warping (DTW) transforms the gestures to make them directly comparable. To do this, DTW stretches gestures such that they are aligned perfectly in time by repeating sampling points in areas where one input is shorter than the other. So, an $N \times M$ matrix of path differences between gestures is constructed and traversed in a way that resamples each gesture to an equal length. After that, it can make point-wise distance comparisons and measure them against a threshold to determine authentication.[9,10]

DTW is sample invariant due to the alignment of time series. Distance-based measures are used to recognize stored templates after alignment, allowing for trainability. It's partially adaptable, using the same logic applied with the geometric methods above. It's partially configurable, because users can control the authentication threshold and the template count. Naive DTW implementation is computationally inefficient, but there are implementations that reduce the computation time without affecting recognition efficiency. DTW is partially storage conscious, requiring no extra data beyond a few templates, but it makes no provisions for data protection. It's partially attack resistant given that it's based on one layer of resistance (coordinate pairs). It's also partially API friendly; implementing path alignment and reducing complexity beyond the general implementation is not straightforward.

Machine Learning Techniques

Machine learning uses algorithms to teach computers how to perform tasks from data, which is a natural fit for performing gesture recognition. The most popular methods are hidden Markov models (HMMs) and support vector machines (SVMs).

Hidden Markov models. For HMM recognizers, the user's input appears as a collection of features (such as time, pressure, and distance) rather than as a known gesture. The sequence of states (the Markov process) that a gesture undergoes—such as up, left—can't be seen; only the measurable outcomes are visible (it lasted for *t* time and has *N* samples).

An HMM consists of a set of states; a transition probability matrix, which describes the chance to transition from one state to another; and an output probability function. Each individual gesture to be recognized by a system requires its own prespecified HMM. Given the input data, the recognition task is to figure out a sequence of state transitions to which a user's gesture might be mapped. More than one possible state sequence exists for any input. The most likely sequence (and thus, the gesture) is determined by evaluating the joint probability of the sequence and observations.[11]

Support vector machines. SVMs are a set of supervised learning algorithms that can be used for classification and recognition problems.[12] SVMs solve a binary classification problem: with gestures, the two classes would be a specific gesture (Class 1) and "not a gesture" (Class 2). A collection of gestures can be split into one of these classes, depending on what is being recognized. If plotted in a plane, there should be a visual way of drawing a boundary to separate Class 1 and Class 2 data points. The Class 1 and Class 2 data points that are closest to each other (and thus closest to the boundary) are the *support vectors*. These are the most difficult data points to classify, because they most closely affect the boundary's location and contour.

SVMs are useful because they can support many variables, representing them as vectors of features. However, the data must be mapped to a higher dimensional space proportional to the number of features. To create a decision boundary, a nonlinear equation must be solved. The training data is used to construct the boundary, and new user input is classified as a gesture depending on which side of the boundary it ends up on.

Sampling issues. Although more is often better, neither HMM nor SVM depend on a large number of samples. Both can be trained to accept any input. Because learning is a continuous process, both can adapt to user input over long periods. Computationally, they are the slowest algorithms we consider and are thus only partially computationally efficient. HMMs require many training examples due to the many different paths a gesture can take; this requires more space. SVMs can function properly with a low number of examples, depending on the basis set and complexity of the recognition space. Neither HMMs nor SVMs store training data in a way that protects it from leakage. As such, HMMs aren't storage conscious and SVMs are only partially conscious. HMMs and SVMs have more layers of resistance to attacks because they can use many different features (beyond coordinate and time data) to recognize gestures. They are, however, difficult to program and aren't API friendly.

Other Gesture Recognition Methods

A gesture could be identified by measuring very small, finite Doppler shifts between incoming and outgoing Wi-Fi signals.[13] If a user is wearing an RFID tag, an array of antennas could track the disturbance caused by a gesture.[14] A tag moving between the antennas generates readings, which are fed back wirelessly and analyzed to determine which gesture is being performed. Similarly, researchers have demonstrated methods for hand gesture recognition with capacitive proximity sensors. A person waving would generate readings of different magnitudes at different timestamps, from which the gesture can be determined.[15]

These methods aren't sample invariant—their recognition approach is based on discovering patterns in data, which sampling to a constant can affect. All three have some degree of trainability, because they are capable of learning some gestures. However, it is difficult to contend with possible resolution issues that might exist between recognizing two similar gestures using these methods. Also, there are no provisions for adaptivity. Although capacitive and RFID recognition are computationally feasible, Wi-Fi recognition is not, because its involved steps would require various transforms to analyze the input data. All three methods are neither configurable nor resistant; they have difficulty in distinguishing users and focus only on gestures. Finally, only Wi-Fi and RFID are partially API friendly because they use common, available equipment (such as routers and tags); in contrast, proximity sensors require special equipment and programming.

Cross Recognizer Comparisons

It's useful to understand how different recognizers reliably compare with each other. Typical measures of recognition performance are receiver operating characteristic (ROC) curves and equivalent error rates (EERs).

An ROC curve is a plot of a user's successful system login rate versus an attacker's successful login rate as the threshold to authenticate is varied from zero (accept any input as the password) to infinity (reject any input, even the real password). As the threshold increases, the rate of false positives (FPR) decreases and the rate of true positives (TPR) increases (up to a certain point, at which it then decreases). Varying this threshold generates a new (TPR, FPR) point at every step—together, the plot forms the ROC curve. EER is the rate at which the number of accepted attackers and rejected true users are equal. Perfect recognizers would have a TPR of 100 percent (all true user attempts accepted), an FPR of zero (all attacker attempts rejected), and an EER of zero (no true users rejected and no attackers accepted).

It seems that the recognizer with the lowest reported EER value would be the best one to use. However, that intuition fails because recognizers are rarely—if ever—computed across the same datasets; most designers generate new datasets to evaluate their system. In an authentication system, simply recognizing a gesture isn't enough; we need to know whose gesture it is.

Many recognizers focus on differentiating between a narrow vocabulary of gestures (such as a circle versus a rectangle) and report results based on that. For authentication, EER values must be computed based on attacks against a gesture, yet most recognizer designers don't do this. To justify the validity of recognition techniques, we must have a common reference point to compare the EERs.

Thus, to comprehensively compare recognizers, we need a large public dataset of gestures intended to test them. This isn't a unique problem. Optical character recognition (OCR) algorithms faced the exact problem that gesture recognizers face now. Researchers had no way to adequately compare all of the different methods until the US Department of Energy commissioned the large-scale creation of a comprehensive OCR dataset that developers could use to test their algorithms. A similar situation existed for speech recognition problems as well. The unifying theme here is that recognition algorithms aren't directly comparable without an extensive public dataset.

Many recognition algorithms depend on the input data type, complicating the process of creating a dataset. However, issues arise if not enough features are gathered from a gesture. As an example, a dataset compiled around motion gestures on a smartphone might collect time, coordinate, accelerometer, and gyroscope data. However, a new recognizer might need data from the gravity or magnetometer sensors. Thus, the dataset should be left entirely open so new features can be added when needed.

Also, the sets used for benchmarking should be categorized based on specific test criteria. Example sets include weak passwords, strong passwords, memorable passwords, and memorable and strong passwords. The gesture type is also an important variable.

Because we as yet have no public dataset, comparing recognizers for authentication purposes requires that we answer four key questions:

- What dataset are the recognizers being compared to?
- How do these recognizers compare to one another based on the eight design criteria?
- Were the EER values computed for the algorithm subject to gesture-based attacks?
- How do error rates vary as features are added or subtracted?

Providing answers to these questions clearly communicates to others both the advantages and disadvantages of a given dataset and the proposed recognizer for authentication.

Gesture-based authentication systems show potential for practical adoption. Although they have disadvantages, these can be attributed to the lack of development in recognizers and infrastructure. Text-based passwords work well for users who are comfortable with physical keyboards, yet touchscreen keyboards are becoming more common with the proliferation of mobile devices. Eventually, touchscreen keyboards might replace physical keyboards altogether. On those interfaces, gestures become a more natural authentication choice than a text-based password, because it is more difficult to type character strings. However, the ubiquity of text-based passwords makes it exceedingly difficult for alternative methods to gain traction. The potential benefits of gestures should not be ignored because of the current limitations.

ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under grant numbers 1223977 and 1228777. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

1. M. Sherman et al., "User-Generated Free-Form Gestures for Authentication: Security and Memorability," *Proc. 12th ACM Int'l Conf. Mobile Systems, Applications, and Services*, 2014, pp. 176–189.
2. A. Paivio, T. Rogers, and P.C. Smythe, "Why Are Pictures Easier to Recall Than Words?" *Psychonomic Science*, vol. 11, 1968, pp. 137–138.
3. J. Bonneau et al., "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes," *Proc. IEEE Symp. Security and Privacy*, 2012, pp. 553–567.
4. I. Jermyn et al., "The Design and Analysis of Graphical Passwords," *Proc. 8th Conf. USENIX Security Symp.* (SSYM 99), 1999; http://dl.acm.org/citation.cfm?id=1251422.
5. J.O. Wobbrock, A.D. Wilson, and Y. Li, "Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes," *Proc. 20th ACM Symp. User Interface Software and Technology*, 2007, pp. 159–168.
6. S. Kratz and M. Rohs, "Protractor3D: A Closed-Form Solution to Rotation-Invariant 3D Gestures," *Proc. 16th Int'l Conf. Intelligent User Interfaces*, 2011, pp. 371–374.

7. L. Anthony and J.O. Wobbrock, “A Lightweight Multistroke Recognizer for User Interface Prototypes,” *Proc. Graphics Interface*, 2010, pp. 245–252.

8. Y. Li, “Protractor: A Fast and Accurate Gesture Recognizer,” *Proc. SIGCHI Conf. on Human Factors in Computing Systems*, 2010, pp. 2169–2172.

9. N. Sae-Bae et al., “Biometric-Rich Gestures: A Novel Approach to Authentication on Multi-Touch Devices,” *Proc. SIGCHI Conf. Human Factors in Computing*, 2012, pp. 977–986.

10. J. Tian et al., “Kinwrite: Handwriting-Based Authentication Using Kinect,” *Proc. ISOC Network and Distributed System Security Symp.*, 2013; http://internetsociety.org/sites/default/files/10_2_0.pdf.

11. L.R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” *Proc. IEEE*, 1989, pp. 257–286.

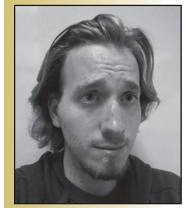
12. V. Vapnik, “The Support Vector Method,” *ICANN*, 1997, pp. 263–271.

13. Q. Pu et al., “Whole-Home Gesture Recognition Using Wireless Signals,” *Proc. Int’l Conf. Mobile Computing and Networking*, 2013, pp. 27–38.

14. P. Asadzadeh, L. Kulik, and E. Tanin, “Gesture Recognition Using RFID Technology,” *Personal Ubiquitous Computing*, vol. 16, no. 3, 2012, pp. 225–234.

15. R. Wimmer et al., “Thracker—Using Capacitive Sensing for Gesture Recognition,” *Proc. IEEE Conf. Distributed Computing Systems*, 2006, pp. 64.

the AUTHORS



Gradeigh D. Clark is a graduate student in the Human-Computer Interaction Group and WINLAB at Rutgers University. His research interests include security engineering, alternative authentication, privacy, social & mobile computing, cryptocurrency, and crowdsourcing. Clark has a BS in electrical and computer engineering from Rutgers University. Contact him at gradeigh.clark@rutgers.edu.

Janne Lindqvist is an assistant professor of electrical and computer engineering and a member of WINLAB at Rutgers University, where he directs the Human-Computer Interaction Group. His research interests are at the intersection of security engineering, human-computer interaction and mobile computing. Lindqvist has a D.Sc. in computer science and engineering from Helsinki University of Technology, Finland. Contact him at janne@winlab.rutgers.edu.