RhythmLink: Securely Pairing I/O-Constrained Devices by Tapping

Felix Xiaozhu Lin
Department of Computer Science
Rice University
Houston, TX 77005 USA
xzl@rice.edu

Daniel Ashbrook, Sean White
Nokia Research Center
2400 Broadway
Santa Monica, CA 90404 USA
{daniel.ashbrook,sean.white}@nokia.com

ABSTRACT

We present RhythmLink, a system that improves the wireless pairing user experience. Users can link devices such as phones and headsets together by tapping a known rhythm on each device. In contrast to current solutions, RhythmLink does not require user interaction with the host device during the pairing process; and it only requires binary input on the peripheral, making it appropriate for small devices with minimal physical affordances. We describe the challenges in enabling this user experience and our solution, an algorithm that allows two devices to compare imprecisely-entered tap sequences while maintaining the secrecy of those sequences. We also discuss our prototype implementation of RhythmLink and review the results of initial user tests.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces—Haptic I/O, Input Devices and Strategies

General terms: Design, Human Factors, Security

Keywords: Device pairing, mobile devices, input, rhythm, security, taps

INTRODUCTION

As technology shrinks, the user interface for mobile devices is becoming the largest physical part of the device. Some input devices now consist solely of sensors [13, 14] or physical affordances [1]. As devices get smaller and cheaper, Weiser's vision of "'scrap computers'. . . that can be grabbed and used anywhere" [23] may extend to general input and output peripherals that can be connected on the fly to laptops, mobile phones, watches, and other personal, mobile devices.

User interaction with such peripherals has been explored, but a remaining challenge is the way in which these tiny, wireless devices are connected together or "paired". Consider a set of Bluetooth headphones: most have just a few buttons and no visual interface at all. In order to connect the headphones to a phone, the user must follow a multi-step procedure: retrieving the phone from wherever it is stored (pocket, bag, or purse, for example), scanning for devices and selecting the headphones, typing in the headphone's security code, and placing the phone back into the pocket or bag.

While these steps are familiar to many users of wireless technology, research indicates that this pairing process is often slow [10] and confusing [22]. Our research addresses the usability of device pairing, while at the same time maintaining security.

In this paper, we introduce our system "RhythmLink" which improves on prior pairing methods. RhythmLink allows users to securely pair a peripheral with a host device such as a mobile phone via rhythmic taps, entirely bypasses the usual discovery phase, needs just a single button or other binary input device to work, and does not even require the user to be in physical possession of the intended host device.

RhythmLink Use Example

The following example illustrates the fundamental ideas of our system. Alice has just purchased a new RhythmLink-enabled phone. As part of the setup procedure, the phone asks her to input a new pairing tap password. After considering for a moment, she taps out the rhythm of the chorus from "Let It Be" by The Beatles.

A few weeks later, Alice is at a friend's party. Her friend has some new RhythmLink-enabled wireless speakers, and invites Alice to play some tunes. She taps out "Let It Be" on the pairing button on the back of one of the speakers. In just a few seconds, the speakers have securely paired with Alice's phone—still sitting at the bottom of her purse—and started to play her music.

Contributions

Our research makes two major contributions. First, RhythmLink improves the *usability* of mobile device pairing, from both a speed and a user experience standpoint. It does so by a) removing the need to search for devices with which to pair; and b) allowing pairing to be initiated and completed from the peripheral, without having to hold, touch, or be physically near the phone.

Secondly, RhythmLink accomplishes the above while maintaining the *security* of mobile device pairing. As we will discuss, this is not a trivial problem: RhythmLink must compare a tapped sequence with the stored tapped password on surrounding devices, but without any of the devices revealing their tap data.

Design Challenges

For clarity, in the rest of this paper, we will use *phone* to indicate the host device—whether it be phone, music player, computer or television—and *peripheral* to identify the device that is to be connected to the host. In the example above, we would refer to the wireless speakers as the peripheral.

Because both usability and security are important features of wireless pairing, we have identified the following as desiderata for an improved pairing procedure:

Security Undesired peripherals must not be able to be paired without the phone owner's knowledge or consent. Likewise, the peripheral should only be paired to the intended phone.

Fast selection Selecting which of a number of peripherals to pair with should be fast and natural.

Fast authentication Authentication—the user verifying that indeed two devices should be paired—should be fast and natural.

Variety of input mechanisms No particular kind of input device such as keypads should be required: peripherals come in all shapes and sizes, so the pairing process should work with a wide variety of input mechanisms such as buttons or accelerometers.

Variety of output mechanisms Similarly, no particular kind of rich output should be assumed for either peripherals or phones.

Low power Some tiny devices may be very computationally- and battery-constrained: pairing shouldn't be slow or drain too much power.

As we discuss in the rest of the paper, RhythmLink attempts to fulfill each of these desiderata.

RELATED WORK

Wireless Pairing

In order to understand RhythmLink, it is helpful to have a basic understanding of wireless pairing and how it works. The essential goal of a pairing procedure is to establish trust between two devices that have no pre-existing information about each other. Because peripherals can expose private data (e.g., a GPS receiver) or control the host device (e.g., a phone headset), it is important to verify that the two device should really be connected. That is, the user should have confidence that only her phone can connect to her GPS, and that nobody else will be able to connect their headset to her phone.

Because wireless devices must operate over untrusted channels, their communication is susceptible to compromise by methods such as the *man-in-the-middle* (MITM) attack, during which an attacker is inserted in between the two devices, making them believe that they are securely communicating, whereas in fact the attacker is relaying—and possibly modifying—the messages.

In order to be secure against MITM attacks, the two devices should have a *shared secret*: some information that they each know, that is used to verify that the two devices are directly communicating. *It is vital that this shared secret is not given away to anyone*—that is, it is not sent over wireless either in the clear (because a third party could find it out) or sent encrypted to another device (because a MITM attack could already be underway). The only way the shared secret can work is if both devices *independently* know it.

The shared secret is often communicated between devices using *out-of-band* (OOB) information sent by a different channel than the primary wireless connection. Frequently, the OOB channel is human-perceptible: when the communication can be directly perceived by the user, an attacker cannot modify messages without being detected. Various OOB channels have been explored. Bluetooth, for example, can display a number on one device that is typed in on the other device. In this case, the human is the OOB channel.

The next sections discuss various authentication and pairing methodologies in more detail, and the advantages and drawbacks of each.

Rhythm-Based Authentication

RhythmLink's tap input is inspired by two rhythm-based approaches to authentication found in the literature. Westeyn and Starner's Prescott [24] uses song-based rhythmic eye-blinking patterns to authenticate a user. The system detects blinks using a camera and computer vision and compares them to a database using dynamic time warping. Due to the eye/camera line of sight, Prescott is intrinsically secure for fixed installation (the authors give the example of a door at an airport).

Wobbrock's TapSongs [25] extends the rhythm-based concept to any binary sensor such as a button. With a wired connection between the sensor and the computer, TapSongs compares the input rhythm to a database of taps using non-dynamic linear warping.

By showing that rhythmic input through sensors such as button or cameras is a feasible method for entering password information, Prescott and TapSongs provide a way to minimize the I/O requirements of peripheral devices. However, there are two reasons that neither system is suitable for wireless pairing. First, they both feature a database of stored passwords to which the candidate input is directly compared. Direct comparisons are not desirable in a wireless pairing scenario: until a trusted connection is established, neither party will want to reveal their passwords. Secondly, in Prescott and TapSongs, the system is trusted end-to-end: the input (camera or button) is coupled to the database of rhythms, so there is little possibility that someone will steal a password. On the other hand, wireless connections are susceptible to eavesdropping, so directly transmitting the blinked or tapped password would be insecure; even cryptographic techniques such as public key exchange still allow MITM attacks.

Bluetooth Pairing

The current state of the art for connecting wireless peripherals and hosts is Bluetooth, so any new pairing procedure should improve upon its operation. From the user's perspective, there are several steps to successfully completing pairing with Bluetooth:

1. (Optional for some peripherals) The user sets the peripheral to "discoverable" mode.
2. On the phone, the user initiates the pairing process.
3. The phone searches for discoverable peripherals and displays a list; the user selects the desired peripheral.
4. The user completes one of several security procedures [11] including: entering a Personal Identification Number (PIN), possibly hardcoded to a default such as "0000"; comparing a number on two displays; or using an *out-of-band* (OOB), non-Bluetooth-based mechanism such as RFID to communicate security information.
5. The devices are paired and ready to use.

There are a number of usability issues with the Bluetooth pairing process: discovery of available devices is often slow, and the found devices can have similar ("iPhone") or confusing ("1USA24881") names; both devices must be physically available to the user to confirm the pairing procedure (except with a special "just works" security protocol [11], which offers no MITM protection); and there is little support for truly secure pairing with I/O-constrained devices such as headsets.

Bluetooth's security is not adaptable to rhythm-based input. Bluetooth avoids revealing PINs by performing cryptographic calculations on each bit of the PIN; only if the results exactly match on each end do both devices know that they have matching codes. However, because of the variability of human rhythm input, this technique cannot work for RhythmLink: even a millisecond of difference between two taps will yield very different calculations on the two devices.

Pairing Research

Some of Bluetooth's usability issues with respect to pairing have been addressed in prior research. For an overview of many of these efforts, including evaluation of their usability, see the work of Kumar *et al.* [10]. Similar to our work, the aim of prior research is frequently to simplify pairing by removing discovery and/or simplifying authentication.

Some such systems use the proximity of two devices as OOB information [5, 12, 16, 20]; however, this approach requires that the devices be able to get near to each other, which prevents, for example, pairing a phone with a remote large-screen display or pairing a headset with a phone in a bag across the room.

Another solution is to use simultaneous input through shaking the devices together [7, 12] or concurrent button presses [15, 19] or gestures [4]. These solutions require proximity as well—either to hold both devices in one hand, or to access buttons on both devices at the same time. If one device is too large to hold, is remote (e.g., the aforementioned large-screen display), or is simply inaccessible in a bag or pocket, these methods will not work.

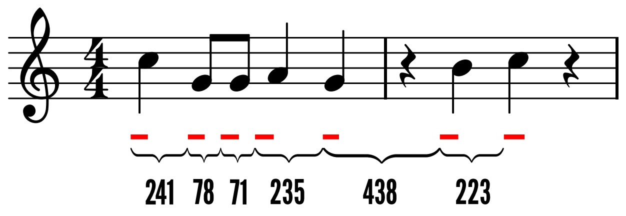

Figure 1: The tune "Shave and a Haircut, Two Bits". ♩ and 𝅘𝅥𝅮 represent one beat each, ♫ is a double beat, and each 𝄽 is a one-beat rest. The red lines represent the taps input for the same tune by one user. The numbers indicate the length of each interval between tap starts, in milliseconds.

RHYTHMLINK

RhythmLink builds on Prescott [24] and TapSongs [25] by using rhythm-based passwords for authentication. The idea is simple: the user starts by entering a tapped password—or "tapword" (a representation distinct from the TapSongs method in terms of model and comparison algorithm)—on the phone. This tapword is defined once per phone (or other host device) and does not need to be changed unless the user desires. When the user wants to connect a peripheral to the phone, she taps out the same tapword on the peripheral. The peripheral then—without revealing the tapword—queries all of the phones in the vicinity to see if any possess a similar tapword. If one does, that phone pairs with the peripheral.

RhythmLink uses tap-based input in order to require a minimum of I/O capability on the peripheral. A button is the simplest way to put in a tapword, but any sensor that can be binarized could be used: a microphone, accelerometer, capacitive or resistive touch sensor, or pressure sensor all provide miniature, "tappable" input.

Similar to Prescott and TapSongs, RhythmLink's tapwords are rhythmic. This is not due to any algorithmic constraint, but simply because—as Wobbrock points out in his TapSongs paper—people are rhythmic by nature. A simple example of a tapword is the rhythm from "Shave and a Haircut", illustrated in Figure 1.

RhythmLink addresses many of the usability problems encountered in earlier research:

- The difficulty of discovering and identifying the peripheral with which to pair: RhythmLink starts at the peripheral, and the tapword identifies and authenticates the phone.
- The necessity of having both the phone and peripheral in hand: RhythmLink only requires the peripheral to be held by the user; the phone can be anywhere within radio range.
- The need for rich input and/or output: in contrast to schemes requiring screens [3], keyboards [3], extra sensors [17], or even lasers [12], RhythmLink requires only a button or other binary sensor.

By design, RhythmLink is very simple from the user's perspective. However, comparing two tapwords in a secure way—without revealing a tapword to an untrusted device—is more complex.

Security Challenges

Using taps rather than alphanumerics for password input means having to do more complex comparisons in order to determine if the user has provided the right password. For normal textual passwords, there is a one-to-one mapping between a correct password and its stored analogue: "`pickle`" is the same every time a user enters it, and if it's not, it's not the right password—"`picklE`" is still incorrect.

Taps are more ambiguous, however, because they depend on timing. Humans cannot exactly reproduce timing, so each time a tapword is entered, there will some amount of error in the relative timing between notes. In fact, as Wobbrock [25] points out, Weber's law indicates that the longer the interval between taps, the more variable a person's timing will be [21]. This is why pattern matching techniques were used in Prescott and TapSongs: a candidate sequence is compared in turn to each stored sequence—yielding a closeness score, or distance—until a sufficiently close match is found. One challenge for wireless pairing therefore, is how to do a distance comparison in the vein of Prescott or TapSongs, but without revealing the devices' tapwords.

Another challenge is the limited computational power and battery available on tiny devices. A peripheral such as a pair of headphones will be very limited in its ability to compute and communicate, so any security algorithm must be lightweight. Computation is not only a concern from a battery life perspective. If the pairing process is so computationally intense that it takes many seconds to complete, it will be frustrating to the user.

Finally, tapwords share some weaknesses with text-based passwords. The shorter a tapword is, the more susceptible it will be to brute-force attacks (where an attacker tries to guess every possible combination). An ideal text password will contain a high degree of randomness (e.g., "`kS!U;jL%rq`"), whereas by their very nature tapwords are less random. Although Wobbrock found that rhythmic variation between individuals usually prevents an attacker from logging in with a stolen password, it would be ideal to provide the user an estimate of the strength of their password.

Comparing Tapwords

In order to calculate the similarity between two tapwords, RhythmLink needs a numeric representation of each. Although information such as pitch (the vertical location of each note in Figure 1) or pressure or acceleration (depending on the sensor) could be used as part of the tapword, RhythmLink currently uses only the interval between the starts of two tap events (rather than intervals, TapSongs [25] uses the time of down and up events, but we found no advantage with this approach). Therefore, the tapword "Shave and a Haircut, Two Bits" might be represented by RhythmLink as shown at the bottom of Figure 1.

As illustrated in Figure 2, a user will not tap in exactly the same way every time; so to enable non-precise matching, we want to capture some of this variability. We do this similarly to TapSongs: by having the user input their personal tapword multiple (5–10) times on the phone, and building a model based on the mean (μ) and standard deviation (σ) of the length of each interval. We then store, on the phone, a model of the n interval-long tapword: $(\mu_1, \sigma_1), \ldots, (\mu_n, \sigma_n)$.

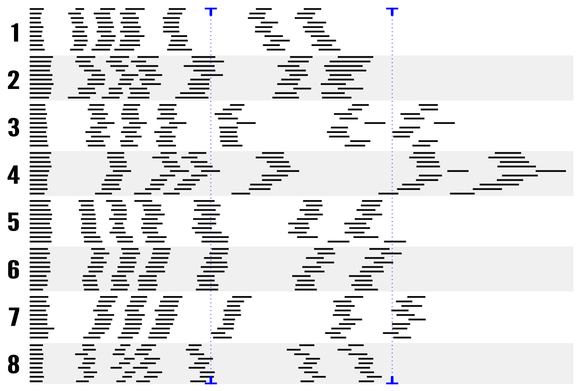

Figure 2: "Shave and a Haircut, Two Bits" for all eight participants in the pilot study. Each shaded band represents one user; within each band, the black lines show the duration of each press for ten repetitions of the sequence. The horizontal axis is three seconds long; the dotted blue vertical lines indicate one-second intervals.

As discussed earlier, the requirement for secrecy in pairing means that—unlike Prescott [24] and TapSongs [25]—RhythmLink cannot directly compare sequences. However, due to the properties of our encryption system (discussed in the next section), RhythmLink can calculate the Euclidean distance between two tapwords without revealing the tapwords themselves. This distance is defined, for a tapword x and model (μ, σ):

$$dist = \sqrt{\sum_i \left(\frac{\mu_i - x_i}{\sigma_i}\right)^2} \tag{1}$$

For each interval i, we divide the difference between its length x_i and the model's mean length μ_i by the standard deviation for that interval σ_i. Dividing by σ makes it easier to match with longer intervals, which—in accordance with Weber's law [21]—will have a higher standard deviation.

When tapword is entered on the peripheral, it communicates with each of the surrounding phones. Using the Euclidean distance, it compares its tapword with each phone's model; if the distance is less than some threshold, then the two devices know their tapwords match, and connect.

Security

In order to securely pair, neither device should reveal its tapword to the other. The challenge is how the Euclidean distance calculation can be performed without either client being able to directly interact with the other's tapword.

Terminology and Preliminaries Encryption of x with key K is denoted $E_K(x)$, while decryption of the same is $D_K(x)$. We therefore might have $D_K(E_K(x)) = x$.

RhythmLink uses elliptic curve cryptography (ECC) [8], a cryptographic scheme that is often used for embedded devices. ECC is *asymmetric*; that is, each party has a public key (K) and a private key ($\tilde{K}$). Anyone can use a public key to encrypt a message: $E_K(m) = \hat{m}$, but the public key cannot be used to decrypt $\hat{m}$. Only the private key can be used: $D_{\tilde{K}}(\hat{m}) = m$. When a number is encrypted, it looks like a random number; for example, the value "5" might encrypt to $E_K(5) = \texttt{1dcca23355272056f04fe8bf20edfce0}$.

One of the interesting properties of ECC is that two encrypted numbers can be added to each other without decrypting them first, and an encrypted number can be multiplied by an unencrypted number without decrypting it first. So, for a given numeric constant j, $E_K(j)+E_K(x) = E_K(j+x)$ and $j \cdot E_K(x) = E_K(jx)$. In RhythmLink, these two properties allow the phone and peripheral to *collaboratively* perform the Euclidean distance calculation on their tapwords without actually revealing the tapwords to each other.

Distance Calculation As described earlier, the user trains the phone for their tapword, yielding a model $(\mu_1, \sigma_1), \ldots, (\mu_n, \sigma_n)$. When pairing, the user inputs a tapword—a single time—on the peripheral. This tapword is represented as a set of intervals $x_1, \ldots, x_n$ (later we will discuss the situation where the tapword and the model are of unequal lengths). The goal of the peripheral is to find a phone with a matching model; so it communicates with each visible phone in turn. During this communication, the goal of the two devices is to determine if the Euclidean distance between the peripheral's tapword and the phone's model is under some threshold t: $dist <? \; t$. To collaboratively calculate the distance, Equation 1 can be squared and expanded:

$$dist = \sqrt{\sum_i \left(\frac{\mu_i - x_i}{\sigma_i}\right)^2}$$

$$\begin{aligned}
dist^2 &= \textstyle\sum_i \left(\frac{\mu_i - x_i}{\sigma_i}\right)^2 \\
&= \textstyle\sum_i \frac{\mu_i^2 - 2\mu_i x_i + x_i^2}{\sigma_i^2} \\
&= \textstyle\sum_i \frac{\mu_i^2}{\sigma_i^2} + \sum_i \left(\frac{-2\mu_i}{\sigma_i^2} \cdot x_i\right) + \sum_i \frac{x_i^2}{\sigma_i^2} \\
&= \underbrace{\textstyle\sum_i \frac{\mu_i^2}{\sigma_i^2}}_{ⓐ} + \left(\underbrace{\frac{-2\mu_1}{\sigma_1^2}}_{ⓑ} \cdot x_1 + \ldots + \underbrace{\frac{-2\mu_n}{\sigma_n^2}}_{ⓑ} \cdot x_n\right) + \textstyle\sum_i \left(\underbrace{\frac{1}{\sigma_i^2}}_{ⓒ} x_i^2\right)
\end{aligned}$$

This equation requires input from both sides to compute: the intervals x_i from the peripheral and the model (μ_i, σ_i) from the phone. However, because the phone and peripheral don't trust each other yet, they don't want to reveal this information. Therefore, they use encryption to hide the values, collaboratively computing the result of the distance equation.

The phone generates a public key PH and sends it to the peripheral. Using its key, the phone also encrypts the sum of its squared intervals ⓐ $E_{PH}(\sum \mu_i^2/\sigma_i^2)$ and, for each individual interval i, $E_{PH}(-2\mu_i/\sigma_i^2)$ (the ⓑs) and $E_{PH}(1/\sigma_i^2)$ (the ⓒs), and sends these to the peripheral.

The peripheral's goal is to generate $dist^2$. Recalling that, using ECC, we can multiply and add encrypted values and they will remain encrypted, the peripheral calculates the third term, using the encrypted ⓒs and its own x_is:

$$\begin{aligned}
\textstyle\sum_i \left(\frac{1}{\sigma_i^2} x_i^2\right) &= \frac{1}{\sigma_1^2} x_1^2 + \cdots + \frac{1}{\sigma_i^2} x_i^2 \\
&= E_{PH}\left(\frac{1}{\sigma_1^2}\right) x_1^2 + \ldots + E_{PH}\left(\frac{1}{\sigma_n^2}\right) x_n^2 \\
&= E_{PH}\left(\frac{1}{\sigma_1^2} x_1^2\right) + \ldots + E_{PH}\left(\frac{1}{\sigma_n^2} x_n^2\right) \\
&= E_{PH}\left(\textstyle\sum_i \frac{x_i^2}{\sigma_i^2}\right)
\end{aligned}$$

In a similar fashion, provided with n $E_{PH}(-2\mu_i/\sigma_i^2)$s (the ⓑs), the peripheral can compute the center term, yielding

$$E_{PH}\left(\textstyle\sum_i \frac{-2\mu_i x_i}{\sigma_i^2}\right)$$

Now the peripheral has, encrypted, all of the pieces of the equation, which it can add together:

$$\begin{aligned}
E_{PH}\left(\textstyle\sum_i \frac{\mu_i^2}{\sigma_i^2}\right) + E_{PH}\left(\textstyle\sum_i \frac{-2\mu_i x_i}{\sigma_i^2}\right) + E_{PH}\left(\textstyle\sum_i \frac{x^2}{\sigma_i^2}\right) &= \\
E_{PH}\left(\textstyle\sum_i \frac{\mu_i^2 - 2\mu_i x_i + x^2}{\sigma_i^2}\right) &= \\
E_{PH}\left(\textstyle\sum_i \left(\frac{\mu_i - x_i}{\sigma_i}\right)^2\right) &= E_{PH}(dist^2)
\end{aligned}$$

Now the peripheral can send $E_{PH}(dist^2)$ to the phone, which can decrypt it $D_{\tilde{PH}}(E_{PH}(dist^2)) = dist^2$ and compare it with the threshold: $dist^2 <? \; t^2$.

Assuming that the result is under the threshold, the phone now trusts the peripheral. However, the peripheral still does not trust the phone, so the phone must prove that its model is compatible. Since the phone trusts the peripheral, it can simply send its model to the peripheral, which can compare it directly. However, there is still the possibility of an ongoing MITM attack. To encrypt both the model and further communication, the two devices must establish a session key in a way resistant to MITM attacks. Using Password-Authenticated Key Exchange (PAKE) [2], such a key can be created using simple information known to both devices. In our case, the number of intervals (known *a priori* to each device) can be used to bootstrap secure key generation. Using this session key, the phone can encrypt its model and send it to the peripheral, which decrypts it and computes the Euclidean distance directly with its x_is. If there is a match, the peripheral knows it can trust the phone.

Unequal Tapword Lengths An issue is how to compare tapwords that are unequal lengths. The problem is twofold: first, the Euclidean distance works only on sequences of equal length, and second, to remain completely secure, neither the phone nor the peripheral should know whether or not their tapwords have the same number of intervals.

Our solution for this is to work with a standard length: we pick some average number of intervals n, and any tapword that is longer than n gets truncated. Any tapword shorter than n gets padded to n intervals with a pre-determined set of large filler values f_i. (In order to prevent guessing the number of filler values, each has a tiny random amount added to it, thus changing the encrypted value.) The values are universally agreed upon by RhythmLink-compliant systems, so that two tapwords of length m (where $m < n$) compare correctly with the Euclidean distance: the remaining f_i values will calculate as having (nearly) 0 distance, and only the m differing values will have any affect on the final calculation of $dist$. Tapwords of differing lengths will have $x_i - f_i$ included in the distance, yielding a large value that will not fall under the threshold.

Further Securing the Calculation A remaining issue is that the phone gains some information—$dist$—about how its sequence compares against the peripheral's, and could potentially use this to guess at its sequence. Ideally, either device would find out whether its sequence matches or not without ever learning $dist$. We can accomplish this goal through the use of a so-called "black box" function, which will allow the peripheral to perform the comparison $dist <? \; t$ without revealing $dist$.

Recall that, at the end of its calculations, the peripheral is in possession of $E_{PH}(dist^2)$. It generates a random value r and adds it: $E_{PH}(dist^2 + r) = E_{PH}(v)$ and sends the result to the phone. The phone can use its key and decrypt this message: $D_{\tilde{PH}}(E_{PH}(v)) = v$. Because the phone doesn't know r, it can't calculate $dist^2$ from v.

To compare $dist^2$ with t^2, the peripheral creates an encrypted, or "garbled", function [9]; similar to arithmetic computation with ECC, this function allows efficient *comparison* of two values without revealing the values to either device. The peripheral constructs and sends this function to the phone.

To make the comparison, the garbled function takes v and t^2 as input, internally performing the computation

$$\begin{aligned} f(v, t^2) &= v - r <? \; t^2 \\ &= (dist^2 + r) - r <? \; t^2 \\ &= dist^2 <? \; t^2 \end{aligned}$$

returning a one-bit Boolean value T to the phone. Because the garbled function is constructed by the peripheral, it can be confident that the phone is not trying to find out what $dist^2$ is. If T is true, the phone knows it can trust the peripheral. As above, it then sends the peripheral its model, which compares it with its intervals and determines that it can trust the phone, and they set up secure communications.

Choosing the Threshold Because a phone will be owned by a single user, but peripherals may pass between people, the phone should be the device that determines the threshold for matching. The threshold can be dynamically generated on the phone when the tapword is initially input by the user. Taking each of the N repetitions of the tapword, and finding the Euclidean distance to each of the other inputs will give us a mean μ and standard deviation σ; we can then set the threshold to $\mu \pm k\sigma$. We can pick k to be more or less restrictive; assuming a normal distribution, $k = 3$ will encompass 99.7% of the variation in a user's taps.

Efficiency

As discussed earlier, RhythmLink is intended to work with I/O-constrained devices. In a phone/peripheral pairing situation, it is possible that at least the peripheral will have a small battery and a low-power embedded processor. To maintain an acceptable user experience, delays in pairing should be minimized; thus, it is important that RhythmLink be efficiently executable on low-end CPUs and microprocessors.

The major power drain in RhythmLink comes from the computation involved in ECC. Therefore, if we can avoid doing cryptographic operations unless they are absolutely necessary, we can save both power and time. In an ideal situation, the entire tapword comparison process described above will only take place once: between the peripheral and the intended phone. However, the real world is rarely ideal—in a room full of phones, we want to avoid the peripheral having to attempt to authenticate with each one!

In order to encourage a closer-to-ideal situation, RhythmLink uses a *selector*, consisting of a minimal set of information about the tapword, to attempt to narrow down the number of potential matches between a peripheral and the surrounding phones. Each phone allows unencrypted viewing of its selector, and so a peripheral will only try to pair with a phone that has the same selector as itself.

The rationale behind the selector is twofold: first, it reduces the computational overhead of pairing by orders of magnitude. Without the selector, the peripheral must attempt secure authentication with every phone in the vicinity until it finds one that matches, incurring not only a significant power drain, but increased latency. Secondly, using a selector reduces the information about the tapword that is leaked during the course of normal usage: each failed authentication eliminates a possibility, making the tapword less secret. The selector eliminates most failed authentications; additionally, it allows the peripheral to treat phones that claim to have a matching selector but fail authentication as suspicious.

The challenge in choosing a selector is that it should disclose *just enough* information about the tapword to determine whether two devices should attempt authentication—but no more, to prevent guessing of the tapword (for example, using the number of intervals in the tapword would reveal information useful to an attacker). Additionally, the selector generation needs to work with the random timing errors present during human tapword input.

Currently, RhythmLink uses the first two intervals of a tapword as the selector. Before any authentication attempt, the phone reveals the mean (μ_i) and standard deviation (σ_i) of the lengths of the first two intervals of its tapword to the peripheral. The peripheral tries to match its first two intervals (x_i), and only proceeds if there is a reasonably close match. Because it has direct access to both devices' selectors, RhythmLink uses part of TapSongs' [25] matching process to determine whether there is a match or not:

$$|x_i - \mu_i| \leq? \; 3\sigma_i$$

We chose the selector this way for three reasons. First, the selector must be based on pattern matching, like the full authentication, because human timing errors make statistics based on the tapword as a whole unreliable. Secondly, by generating a selector based on the start of the tapword, a peripheral can determine whether each phone is a potential match before performing encrypted computations. Finally, after experimentation (see "Preliminary Experiments" section), we chose to use two intervals as a balance between keeping the tapword secret and efficiently pre-emptively rejecting non-matching phones.

Tapword Strength

Another important consideration in RhythmLink is how to encourage users to choose good tapwords. One way is to give feedback to the user on how "strong" a given tapword is. Similar to systems using text-based passwords that indicate whether a password will be resistant to guessing—for example, setting a password for online banking—we likewise want to assign a given tapword a strength score. There are a number of metrics we can use to indicate the strength of a tapword; most either assign or take away points from the total score:

length Is the tapword long enough? A minimum length can be required, or points can be deducted for shorter tapwords, or points assigned for longer ones.

speed A shorter—in terms of time, rather than number of taps—tapword might be harder for another party to overhear, and so may be more secure.

commonality Similar to how "`password`" is a common, and poor, password, "Shave and a Haircut, Two Bits" may likewise become a common and poor tapword. Given a built-in list of common rhythms, RhythmLink can use the Euclidean distance to prevent such tapwords from being chosen.

interval sizes Having a mix of long and short intervals will make a tapword harder to guess; therefore, counting the number of intervals larger or shorter than some percentage of the overall length (i.e., if the tapword is 1000 milliseconds long, counting intervals under 5%—50 ms—or over 60%—600 ms) can contribute to the score.

consecutive intervals The more heterogeneous the interval lengths are, the more secure the password will be. We can look at each set of two consecutive intervals, and assign a score based on the difference between them as compared to the overall length. For example, if the tapword is 1000 ms long, and we have two tap intervals at 100 and 110 ms, we can assign a score of $-(1000/(110-100)) = -100$.

specific rhythms Specific sub-rhythms can be looked for. For example, a (desirable) fast double-tap can be defined as an interval shorter than some fixed value (rather than a percentage of the total length) such as 50 ms, or a long interval can be similarly defined (e.g. 250 ms).

RHYTHMLINK IMPLEMENTATION

To evaluate the RhythmLink security protocol, we employed TASTY [6], a protocol compiler for performing secure cryptographic computation. We provide a high-level description of the RhythmLink protocol, and use TASTY to automatically generate programs that simulate the phone and peripheral. Once invoked, these programs execute RhythmLink over the network, and log the protocol overhead during execution.

We created a prototype RhythmLink system with three Nokia N900 mobile phones, where one phone (N900-P) acts the part of "peripheral" and the other two (N900-X and N900-Y) act as "phones". In our prototype, we used the touchscreen as input to RhythmLink. In an ideal RhythmLink implementation, devices would perform authentication over a low-level protocol, like Bluetooth; however, in the interests of rapid prototyping, our N900-based system communicates over a local 802.11 network.

To use our prototype, a user enters their tapword by tapping on the touchscreen of N900-P. N900-P then generates its selector according to the mechanism described in the Efficiency section, and requests selectors from all visible phones. N900-X and -Y respond with their selectors; if either is a match, N900-P will launch the TASTY-generated "peripheral" program to perform secure RhythmLink authentication with the "phone" program on the other device.

PRELIMINARY EXPERIMENTS

Data Collection

In order to test our algorithms, we asked eight volunteers to input tapwords. Participants were drawn from our lab, ranged in age from 24–40, and two were female. Using our Nokia N900 RhythmLink prototype to record the taps on the resistive touchscreen, we asked each volunteer to tap in a tapword of their own devising ten times. Then participants were requested to tap "Shave and a Haircut, Two Bits" (Figure 1) ten times. We recorded timestamps for tap down and release events. Figure 3 illustrates each volunteer's personally-chosen tapword, and Figure 2 shows "Shave and a Haircut" for each participant.

We asked the volunteers what rhythm they chose for a personal tapword; from 1–8, they used: "Day Glow" by Tow Head, "On" by Aphex Twin, a song from a Chinese movie, Mozart's "Requiem in D Minor", "Row Your Boat", "Mary Had a Little Lamb", "Jingle Bells", and an impromptu rhythm.

Selector Performance

As described earlier, the purpose of the selector is to prevent devices from having to authenticate if it is unlikely that they will have a match. To investigate the performance of the selector, we perform leave-one-out validation: we remove one instance of an initial tapword from the list of tapwords, and then build a model with those remaining. We then use the remaining tapword, and all other users' tapwords, to create selectors, and compare them with the model.

We tested using one, two, and three intervals for the selector. For each test, we recorded the true positive rate (how often the selector correctly indicated the two devices should connect) and the false positive rate (how often there would have been an unnecessary authentication attempt).

Using two intervals for the selector, the average rate of true positives for the users' custom rhythms was 98%: just two

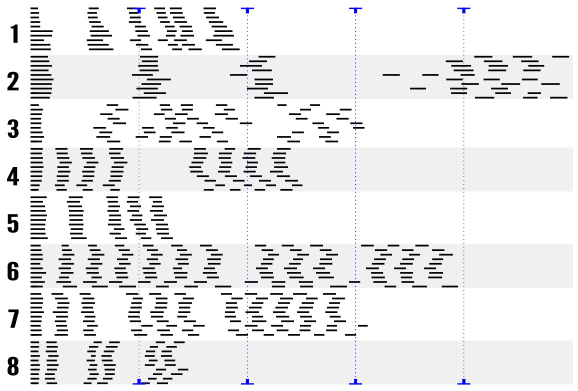

Figure 3: The custom tapword for all eight participants in the pilot study. Each shaded band represents one user; within each band, the black lines show the duration of each press. The horizontal axis is five seconds long; the dotted blue vertical lines indicate one-second intervals.

users had one out of ten tested selectors fail to indicate an authentication attempt should be made. The false positive rate was 9%. For "Shave and a Haircut", the average true positive rate was 94%, while the false positive rate was naturally higher at 32%.

For using the first three intervals to make up the selector, the average true positive rate for custom rhythms was 92%—lower than for two intervals! The false positive rate was just 3%. For "Shave and a Haircut", the true positive rate was still high at 93%, while the false positive rate was 14%.

These results highlight the impact of human error on the system. That is, the more taps the user makes, the more likely she is to make an error. Therefore, to err on the side of a satisfactory user experience—a successful connection on the first try—we use two intervals for the selector.

Tapword Distinguishability

One of the most important aspects of RhythmLink, of course, is that the tapwords themselves are secure; that is, that someone can't accidentally perform a successful authentication with the wrong tapword. To investigate this issue, we used RhythmLink to perform leave-one-out validation on our users' tapwords.

First, we compare the custom tapwords from each user. On average, the Euclidean distance between tapwords from the same user was 26.40, with a standard deviation of 5.10. Because only two users had tapwords with the same number of taps, differing tapwords were easily rejected. Truncating every user's tapword to the shortest number of intervals—four—the average distance between users' own tapwords was 1.27, while the mean distance between different users' tapwords was 20.52, a sufficiently large difference to easily reject wrong attempts.

Following Wobbrock in TapSongs [25], we next compare all users on the same rhythm. With the "Shave and a Haircut" tapword, the average distance within a single user was 18.45, with a standard deviation of 7.35. Between users, the average distance was 79.50, with the average minimum distance between two users' tapwords being 25.59. This is a promising result, showing us a very low probability of accidental overlap between two users with the same tapword. Indeed, Wobbrock addressed purposeful attempts to break security, and discovered that users found it very difficult to tap exactly the same rhythm as another user.

Tapword Reproducibility

In order to determine how well people can remember their tapwords, we re-collected data from each user after between five and twelve days after the initial test. We asked each volunteer to do the same thing as the initial study—to tap in their custom tapword, and then "Shave and a Haircut". For each user, we then created a model based off of their first day's tap input, and used each individual second day tap as input to RhythmLink.

We found that—with one exception—for their custom tapwords, users were in general able to remember and re-tap their rhythms. One user could not remember his rhythm. For the other users, there was an average difference of 6.72 between taps entered on the initial day and on the latter day.

Efficiency

Our N900-based prototype executed the RhythmLink protocol extremely quickly, with no user-perceivable delay. However, the N900 possesses a 600MHz processor—standard for a modern smartphone, but unrealistic for a peripheral device such as a headset or wristwatch. To validate RhythmLink on lower-capability devices, we measured the performance of ECC on an NXP LPC1343[1], a low-power embedded CPU. Processors of this type are commonly found in devices such as headsets, medical devices, and appliances.

For the LPC1343 running at 72MHz, optimized ECC takes .139 seconds per mathematical operation. On the peripheral, RhythmLink has an overhead of two operations per tap interval. Thus, each interval costs us .278 seconds. If our maximum tapword length is 12 taps, we have 11 intervals. Taking off two for the selector, we have 9 intervals involved in computation. So for this processor and maximum tapword length, we can expect a pairing time of around 2.5 seconds. Note that because the selector generation and communication takes place after the first three taps (two intervals), during the remaining taps, it takes no user-perceptible time.

With optimized hardware, RhythmLink can be made even more efficient. ECC encryptions, which dominate the computational overhead of RhythmLink, can be accelerated by several orders of magnitude with lightweight, inexpensive dedicated hardware. For example, an ECC engine based on a 9.9MHz FPGA [18] reduces the computational latency to 56μs per operation, representing a 100,000-times improvement over the general-purpose LPC1343 processor that we measured. Such hardware cryptography accelerators are becoming common in embedded chips, such as the AES (a standard for encryption) engine in many Bluetooth chips. Therefore, we envision that RhythmLink will greatly benefit from

[1]http://ics.nxp.com/products/lpc1000/lpc13xx/~LPC1343/

hardware ECC acceleration as it becomes available in off-the shelf components and devices.

Pairing Speed

RhythmLink requires much less pairing time than existing pairing methods, as it is bounded by computation rather than the speed of user input. As discussed above, with a maximum of nine intervals in a tapword, a RhythmLink pairing procedure will take under three seconds. This pairing time is shorter than almost all of the pairing methods described in the literature. Compare RhythmLink with, for example, the widely-used number-matching pairing method, which takes 8.6 seconds on average [10]. As hardware speed improves, RhythmLink's performance will likewise improve.

CONCLUSIONS

RhythmLink enables secure device pairing while improving usability: it is fast to execute, intuitive for the user, and does not require that both devices be physically co-present. By using rhythmic tap input, RhythmLink allows peripheral devices to have little to no visible input mechanism for pairing: just a button will do. The technique only requires input output sufficient to inform the user whether pairing worked, obviating the need for more complex displays.

In an informal post-study survey, user response to RhythmLink was generally positive, with pilot study participants indicating they would enjoy using such a system to pair their devices.

Compared to other methods, RhythmLink shifts the overhead in pairing from the user to the computational device. That is, the user interaction is less involved, but there is more computation involved. This is a decided advantage, as computing power continues to increase, while human capability remains relatively constant.

Our next steps are to implement RhythmLink on embedded hardware and test it during longer-term use. We are also interested in the impact of using different input modalities: does using a button lead to different rhythmic variability than the touchscreen, and does using a button on one device and a microphone on the other make the tapwords harder to match?

REFERENCES

1. D Ashbrook, P Baudisch, and S White, *Nenya: Subtle and eyes-free mobile input with a magnetically-tracked finger ring*, Proc. CHI, 2011, pp. 2043–2046.
2. S Bellovin and M Merritt, *Encrypted key exchange: password-based protocols secure against dictionary attacks*, Proceedings of IEEE Symposium on Research in Security and Privacy, 1992, pp. 72–84.
3. Bluetooth SIG, *Bluetooth core specifications*, `http://www.bluetooth.org`, April 2011.
4. MK Chong, G Marsden, and H Gellersen, *GesturePIN: Using discrete gestures for associating mobile devices*, Proc. MobileHCI, 2010, pp. 261–264.
5. T Falck, H Baldus, J Espina, and K Klabunde, *Plug 'n play simplicity for wireless medical body sensors*, Mobile Network Applications **12** (2007), no. 2-3, 143–153.
6. W Henecka, S Kögl, AR Sadeghi, T Schneider, and I Wehrenberg, *TASTY: tool for automating secure two-party computations*, Proc. ACM Conf. on Computer and communications security, 2010, pp. 451–462.
7. LE Holmquist, F Mattern, B Schiele, P Alahuhta, M Beigl, and HW Gellersen, *Smart-its friends: A technique for users to easily establish connections between smart artefacts*, Proc. Ubicomp, 2001, pp. 116–122.
8. N Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation **48** (1987), no. 177, 203–209.
9. V Kolesnikov, AR Sadeghi, and T Schneider, *From dust to dawn: Practically efficient two-party secure function evaluation protocols and their modular design*, Cryptology ePrint Archive, Report 2010/079, 2010, `http://eprint.iacr.org/`.
10. A Kumar, N Saxena, G Tsudik, and E Uzun, *A comparative study of secure device pairing methods*, Pervasive and Mobile Computing **5** (2009), no. 6, 734–749.
11. J Linsky, *Simple pairing whitepaper*, `http://mclean-linsky.net/joel/cv/Simple%20Pairing_WP_V10r00.pdf`, April 2011.
12. R Mayrhofer and H Gellersen, *Spontaneous mobile device authentication based on sensor data*, Information Security Technical Report **13** (2008), no. 3, 136–150.
13. C Metzger, M Anderson, and T Starner, *Freedigiter: a contact-free device for gesture control*, Proc. ISWC, 2004, pp. 18–21.
14. T Ni and P Baudisch, *Disappearing mobile devices*, Proc. UIST, 2009, pp. 101–110.
15. J Rekimoto, *Synctap: synchronous user operation for spontaneous network connection*, Personal and Ubiquitous Computing **8** (2004), no. 2, 126–134.
16. J Rekimoto, T Miyaki, and M Kohno, *Proxnet: Secure dynamic wireless connection by proximity sensing*, Proc. Pervasive, 2004, pp. 213–218.
17. R Roman and J Lopez, *KeyLED—transmitting sensitive data over out-of-band channels in wireless sensor networks*, Proc. MASS, 2008, pp. 796–801.
18. N Saqib, F Rodriguez-Henriquez, and A Diaz-Perez, *A parallel architecture for fast computation of elliptic curve scalar multiplication over gf(2m)*, Proc. Parallel and Distributed Processing Symp., 2004, pp. 144–152.
19. C Soriente, Gene Tsudik, and E Uzun, *Secure pairing of interface constrained devices*, International Journal of Security and Networks **4** (2009), no. 1/2, 17–26.
20. A Spahić, M Kreutzer, M Kahmer, and S Chandratilleke, *Pre-authentication using infrared*, Proc. Workshop on Privacy, Security and Trust within the Context of Pervasive Computing, 2005, pp. 1–7.
21. D Sternad, WJ Dean, and K M Newell, *Force and timing variability in rhythmic unimanual tapping*, Journal of Motor Behavior **32** (2000), no. 3, 249–267.
22. E Uzun, K Karvonen, and N Asokan, *Usability analysis of secure pairing methods*, Financial Cryptography and Data Security, 2007, pp. 307–324.
23. M Weiser, *The computer for the 21st century*, Scientific American (2002), 94–104.
24. T Westeyn and T Starner, *Recognizing song–based blink patterns: Applications for restricted and universal access*, Proc. Automatic Face and Gesture Recog., 2004, pp. 717–722.
25. JO Wobbrock, *Tapsongs: tapping rhythm-based passwords on a single binary sensor*, Proc. UIST, 2009, pp. 93–96.