Snap2Pass: Consumer-Friendly Challenge-Response Authentication with a Phone

Ben Dodson Debangsu Sengupta Dan Boneh Monica S. Lam
Computer Science Department
Stanford University
Stanford, CA 94305
{bjdodson, debangsu, dabo, lam}@cs.stanford.edu

ABSTRACT

This paper proposes a challenge-response authentication system for web applications called Snap2Pass that is easy to use, provides strong security guarantees, and requires no browser extensions. The system uses QR codes which are small two-dimensional pictures that encode digital data. When logging in to a site, the web server sends the PC browser a QR code that encodes a cryptographic challenge; the user takes a picture of the QR code with his cell phone camera which results in a cryptographic response sent to the server; the web server then logs the PC browser in. Our user study shows that authentication using Snap2Pass is easy to learn and considerably faster than existing one-time password and challenge-response systems. By implementing our solution as an OpenID provider, we have made this scheme available to over 30,000 websites that use OpenID today.

This paper also proposes Snap2Pay, an extension of Snap2Pass, to improve the usability and security of online payments. Snap2Pay allows a consumer to use one-time credit cards as well as the Verified by Visa or Mastercard SecureCode services securely and easily with just a snap of a QR code.

1. INTRODUCTION

Passwords are the predominant form of authentication system used by today's websites. It is not because the password system is secure; quite the contrary, they are known to have many problems. Passwords are vulnerable to dictionary attacks and can be easily phished using a spoofed web site. Moreover, since users tend to use the same password at many sites, a single server compromise can result in account takeover at many other sites. Despite these limitations passwords are widely used.

Over the years, many enhancements have been proposed, including smart cards, one-time password tokens (such as RSA SecurID) and challenge-response authentication. To date, none of these have been widely adopted on the Web.

Challenge-response is a good case study. While it prevents some attacks that defeat basic passwords, it is rarely used on the Web due to the cumbersome user experience. For example, a system called CRYPTOCard uses a smartcard with a screen and a keyboard where users key in the challenge and then copy the response to the desktop. Authentication using CRYPTOCard takes far longer than authentication using a simple password. As a result, CRYPTOCard is primarily used in corporate settings where the additional hardware cost and the extra inconvenience is acceptable.

This paper introduces a new technique called Snap2Pass that provides a convenient challenge-response system for logging into web sites from a PC. Snap2Pass requires no special hardware beyond a cell phone with a camera. There is no more memorization of passwords and the login process is faster and less error prone than with existing systems such as one-time passwords. Our user study confirms the Snap2Pass ease of use and speed.

1.1 Authentication on the Web

The web has become the dominant platform for modern applications. Perhaps the largest contribution to the web's success as a platform is the ability for users to visit any web page or application from a standard web browser, found on any modern computer today, with no configuration. Simply entering a unique name for that web application is enough to download the necessary code and launch the application.

A web application's authentication system must support this interaction — a user should be able to authenticate against a web application from any available browser, with no additional configuration. In particular, the authentication mechanism is restricted to using generic browser components combined with information supplied by the user. Thus, password-based authentication has remained dominant.

But passwords have known issues. For example, a recent breach at a large web site showed that close to 1% of users choose "123456" as their password [1]. Similarly, Florêncio and Herley found that a single password is typically used to access over five sites [8]. As more sites support email addresses as a username, this poses a significant risk–if an account is breached at one site, others are at risk as well. The study also indicates that on the order of 0.4% of users fall victim to a phishing attack each year.

Snap2Pass is our effort to improve web authentication. It provides improved security for web applications while supporting roaming users.

1.2 Fast Secure Login on the Web Browser

The phone is always with us and switched on. It is a personal device–we do not use others' phones, and nobody uses our phone, except in rare circumstances. In fact, the phone is an ideal device for keeping personal, private, information. In other words, it serves to identify the owner, and can be used as a second-factor authentication. Browsers on the PC, on the other hand, are not personal. We often drift between browsers, on different PCs, at home, at work and on the road. Since the PC has a larger screen and a big keyboard, we can make the best of both worlds by pairing the phone as a personal device with a generic PC browser.

At a high level, the user experience using Snap2Pass is as follows. The user navigates his PC browser to the login page of a web site. The login page displays a QR code containing a cryptographic challenge, among other things. The user takes a picture of the QR code using his cell phone camera. No other user interaction is needed to log in. Under the hood, a pre-shared secret key stored on the phone is used to compute a response to the cryptographic challenge which is then transmitted to the site via the cellular (or wi-fi) network. The site checks the response and if it verifies, triggers the PC browser to successfully complete the login process, and load secured pages. The use of both the phone and PC provides an added security benefit, as checking the co-location of these devices can mitigate man-in-the-middle attacks.

With a snap of the QR code, we have tied our phone to the PC browser, and now we can use these two platforms synergistically to enhance our browsing experience. We can browse on the bigger display of any generic PC available and use the phone to manage our personal identity. There is no setup necessary; we do not have to download any extensions to the browser. Capturing a QR code is easy and fast. Unlike for example pairing with bluetooth, we do not need to read any manuals to learn how to snap a QR code. Consumers already know how to take a picture; they just need to see it done once. The difficulty is identical to swiping a bar code.

1.3 Contributions

This paper makes the following contributions:

Snap2Pass: A consumer-friendly challenge-response authentication system. This technique is easy to use, requiring users to only take a picture of the QR code with a camera on their cell phones. The website displays a QR code that embeds a challenge. The cell phone sends the response to the challenge directly to the web server.

OpenID implementation. We have implemented a custom OpenID provider that uses Snap2Pass, and a mobile client for the Android environment. This provider can be used immediately to log onto over 30,000 existing websites that use OpenID today [18]. We demonstrate that our techniques can be implemented today with minimal changes to legacy services. No changes are required on browsers.

Payments. Beyond login and authentication, we discuss applications of Snap2Pass to payment systems on the web. We combine Snap2Pass with one-time credit card numbers to obtain a payment system providing some user privacy from online merchants. The Snap2Pass concept can also be used to improve the security and usability of the Verified by Visa or Mastercard SecureCode services.

User Study. Our user studies suggest that Snap2Pass is easy to use and is preferred to existing mechanisms like RSA SecureID and CRYPTOCard.

The Snap2 technology. Snap2 is a general technique based on the ability to create quickly a secure three-way connection between a server, a PC browser and a phone. The browser connects to the server with a web page visit, which is then connected to the phone via a QR code that embeds a session key. This enables a server to engage in secure sessions with the browser and the phone simultaneously. The server acts as a secure message router between the phone and the browser.

1.4 Paper Organization

Section 2 presents the threats addressed by this paper. Section 3 describes the core Snap2Pass algorithm based on both symmetric keys and public/private keys. Next, Section 4 describes how accounts for multiple web sites are managed. We describe how we simplify OpenID sign-on by extending it with Snap2Pass. Section 5 shows how the Snap2Pass concept can be extended to improve the experience and security of online payments. Sections 6 and 7 present some extensions and a security analysis, respectively. We describe our implementation in Section 8 and the results of our user study in Section 9. Section 10 presents related work and Section 11 concludes.

2. THREAT MODEL

Snap2Pass is an authentication system designed for ease of use while providing stronger security than traditional passwords or one-time passwords. Our design is intended to protect against the following types of adversaries:

- *Phishing.* Phishing targets users who ignore the information presented in the browser address bar. A phishing attacker sets up a spoof of a banking site and tries to fool the user into authenticating at the spoofed site. Furthermore, we allow for online phishing where the phishing site plays a man-in-the-middle between the real banking site and the user. The phisher can wait for authentication to complete and then hijack the session. One-time-password systems, such as SecurID over SSL, cannot defend against online phishing. With Snap2Pass this attack is considerably harder, as discussed in section 7.2.

- *Network attacker.* We allow the attacker to passively eavesdrop on any network traffic. Moreover, we allow for a wide class of *active* network attackers discussed in Section 7.

- *Phone theft.* Snap2Pass enables quick revocation in case of phone theft.

Snap2Pass does not provide security against malware on the user's machine. Indeed, a sophisticated transaction generator could, in principle, execute transactions on the user's behalf once authentication completes. A good example is the stealthy transaction generator described by Jackson et. al [9]. Similarly, Snap2Pass does not protect against malware on the phone itself.

3. THE Snap2Pass SYSTEM

We now present the core algorithms in the Snap2Pass system. Recall that challenge-response authentication comes in two flavors. The first is a system based on symmetric cryptography. It uses little CPU power and generates very short messages, however it requires that the server possess the user's secret authentication key. As a result, the user must maintain a different secret key for each server where she has an account. The second is a system based on public-key cryptography. It requires considerably more CPU power to generate responses to challenges, but the server only keeps the user's public-key. Consequently, the same user secret key can be used to authenticate with many servers.

We describe both challenge-response systems as implemented in Snap2Pass. We present the basic work flow, from account creation, login, to revocation.

3.1 Symmetric key Challenge Response Authentication

In a symmetric key based challenge-response system, the client(s) and web server communicate using a pre-shared secret key. Our implementation uses a key length of 128 bits. This key is used in the HMAC-SHA1 algorithm to compute responses to server-issued challenges. The challenges are 128-bit length nonces embedded in a QR code, while the responses are 160 bits long and are sent over the wireless network.

3.1.1 Account Creation

The account creation web page invites a new user to submit a username. Upon receiving an acceptable username, the server generates a shared secret for the account and then sends a QR code to the web page encoding the account information. The user launches the Snap2Pass application on the phone and selects the "Set Account" button to activate the camera, consume and decode the QR code. The web site then confirms that the account was created successfully. To avoid adding spurious entries in the provider's database, it should require a user login to complete the creation process.

Figure 1 shows the Snap2Pass application running on the Android phone. We create a QR code representing a created account by encoding the following contents:

```
{  protocol:   "V3"
 , provider:   "goodbank.com"
 , respondTo:
    "https://login.goodbank.com/response"
 , username:   "mr_rich"
 , secret: "2934bab43cd29f23a9ea"
}
```

Note that this process eliminates the need for a user to create and remember a password, which is not just cumbersome but extremely insecure as discussed in Section 1. As a matter of fact, it is not even necessary for the user to have a friendly user name; however it is important for the sake of addressing and reassuring the user that the website recognizes him.

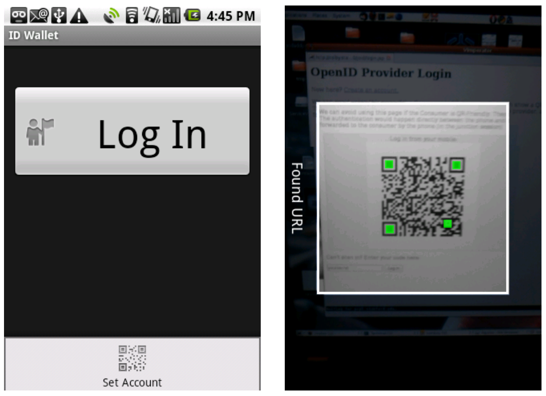


Figure 1: The mobile client running on Android. (a) The home screen, with a single button to log in and with "Set Account" accessible as a menu entry. (b), scanning into a browser session.

Instead of using a user-supplied password, our scheme allows the web server to generate a random key as a shared secret between the web site and the user. The shared secret is presented in a QR code and saved on the phone's password manager once scanned. The user can present it for subsequent logins without needing to know its value.

The QR code also specifies the endpoint where the phone will send responses to challenges as part of the login procedure. A sequence diagram showing the account creation protocol is shown in Figure 2.

In principle, account creation can be done entirely on the phone, without the need for an interaction between the PC and the phone. We chose not to use this approach in Snap2Pass since during account creation the user is often required to supply account details such as a physical address, email address, etc. Typing all this information on the phone can be cumbersome. Instead, with Snap2Pass the user enters all account details on the PC and uses a QR code to move the corresponding credentials to the phone.

3.1.2 Account Login

On the login page, a website displays a QR code and asks the user to snap the picture with his phone's camera to log in. Figure 3 shows a mock up of what a GMail login screen would look like using Snap2Pass. Note that the page provides an alternative login method in case the user's phone is not available.

The QR code on the login page, unique per session, encodes a random challenge nonce to be used in the symmetric challenge-response authentication. This is generally presented within the context of an SSL session between the browser and the web server. An example of the contents contained in a challenge QR code shown at the time of login:

```
{   protocol:   "V3"
  , provider:   "goodbank.com"
  , challenge:  "59b239ab129ec93f1a"
}
```

By binding the challenge nonce to the browser session, the server ensures that only one browser session can make use of its authorization.

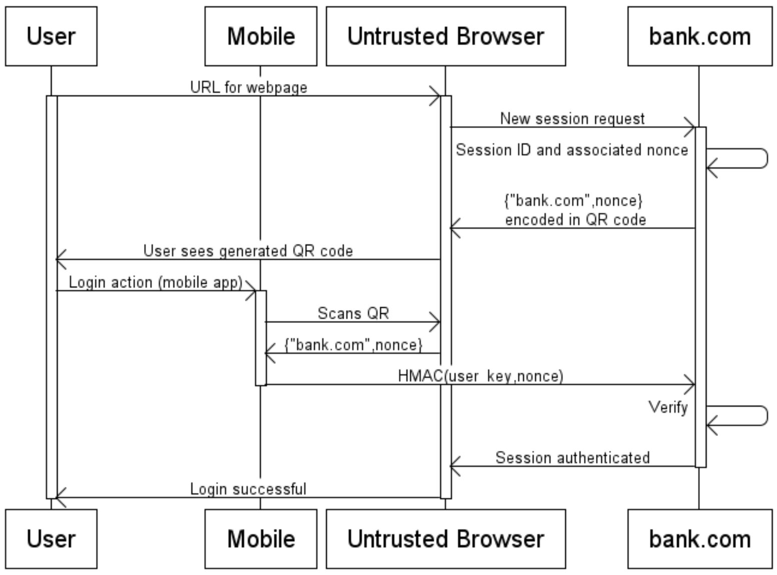


Figure 4: A sequence diagram for logging in to a web application using Snap2Pass.

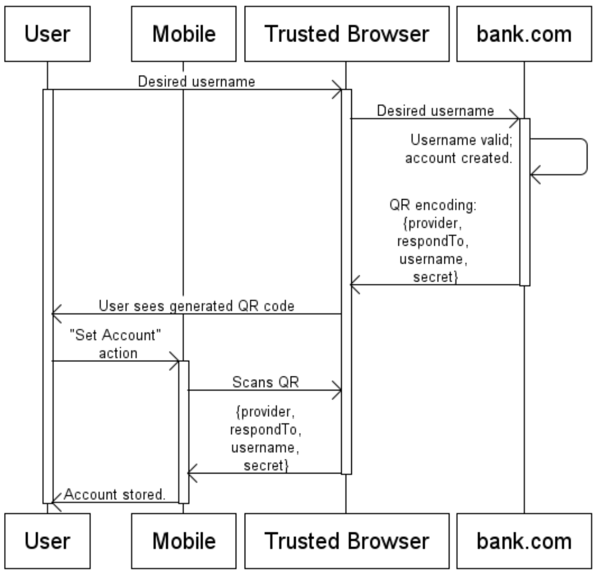
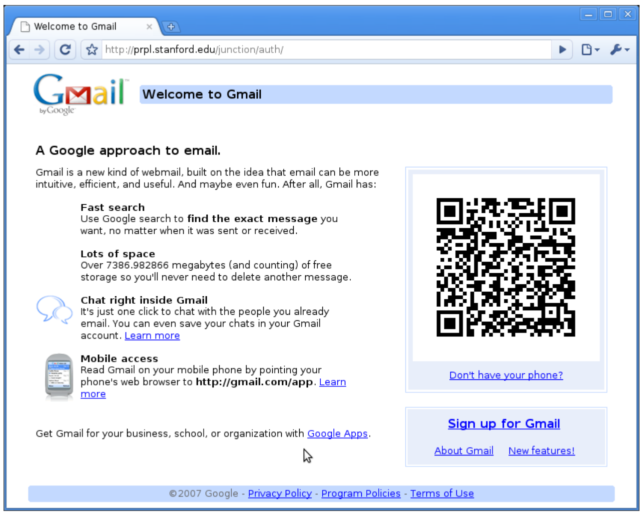


Figure 2: A sequence diagram for creating an account in Snap2Pass.

To log in, the user launches the mobile Snap2Pass application and selects "Log In". By using the phone's camera, the application consumes the challenge QR code and extracts the challenge within. The application finds a shared secret key and response endpoint that match the provider name and desired user account. It computes a response comprising of the HMAC-SHA1 hash of the entire challenge message using the pre-shared secret as key and sends it to the response endpoint, as well as the original challenge and account identifier. The provider verifies this response and,


Welcome to Gmail

http://prpl.stanford.edu/junction/auth/

Welcome to Gmail

A Google approach to email.

Gmail is a new kind of webmail, built on the idea that email can be more intuitive, efficient, and useful. And maybe even fun. After all, Gmail has:

Fast search
Use Google search to **find the exact message** you want, no matter when it was sent or received.

Lots of space
Over 7386.982866 megabytes (and counting) of free storage so you'll never need to delete another message.

Chat right inside Gmail
It's just one click to chat with the people you already email. You can even save your chats in your Gmail account. Learn more

Mobile access
Read Gmail on your mobile phone by pointing your phone's web browser to **http://gmail.com/app**. Learn more

Get Gmail for your business, school, or organization with Google Apps.

Don't have your phone?

Sign up for Gmail

About Gmail New features!

©2007 Google - Privacy Policy - Program Policies - Terms of Use

Figure 3: A mockup of a GMail login page using a QR code.

if successful, the browser session is authenticated with the appropriate account. An example of a response message is shown below:

```
{   protocol:  "V3"
  , challenge: "59b239ab129ec93f1a"
  , response:  "14432nafdrwe2443af"
  , username:  "mr_rich"
}
```

The challenge and response flows occur within SSL sessions. A sequence diagram showing the login protocol is shown in Figure 4.

3.2 Public-key Based Challenge Response Authentication

Key proliferation is a prevalent problem with a challenge response system that utilizes symmetric keys. The user needs to negotiate and manage a shared secret with each web site he visits. We describe a public-key based challenge response systems to combat this problem.

Instead of using symmetric keys, the Snap2Pass mobile application can generate private/public key pair for the user upon installation (and on-demand). The account creation step is modified such that the user presents his public key to the web site instead of having the site generate it. The challenge process proceeds as before. The Snap2Pass application generates a response by signing the challenge with the private key. The web server verifies the response by matching it against the user's public key. The user's public key can be used across all the sites that he wishes to sign in at.

There is an alternative solution to the key proliferation problem in symmetric challenge systems. The user can take advantage of a Snap2Pass-enabled OpenID provider and benefit from OpenID's single sign-on properties across multiple web sites. Thus, the user's Snap2Pass application needs to maintain a single shared secret between the user and his OpenID provider. The number of keys is limited to the number of OpenID providers he uses. He may even use the same private/public key pair across his OpenID providers enabling Snap2Pass to maintain fewer keys.

Note that this protocol requires no certificate authority (CA) infrastructure. Client certificates are entirely avoided in either solution, while the first solution also avoids a CA. The OpenID-based solution is a centralized component necessary to enable Snap2Pass use with unmodified websites while mitigating the key proliferation problem. The OpenID provider may use a CA; our scheme interoperates with this design but does not require it. Phone loss and recovery scenarios are address in Section 7.

4. HANDLING MULTIPLE WEBSITES

Section 3 describes the core Snap2Pass algorithm for logging onto one website. We now describe how we use a single mobile Snap2Pass client to log onto multiple websites, potentially with multiple personas. We also describe how by leveraging OpenID, we can enable the adoption of this technology immediately across a large number of existing websites.

4.1 Logging into Multiple Sites

In practice, we wish to carry only one Snap2Pass client on our phone to log onto multiple websites. The Snap2Pass client maintains a mapping from providers to accounts. It may also maintain multiple accounts per provider, allowing the user to select their desired identity during a login attempt. The response message from the phone to the web site contains the user's identity that is logging in along with the corresponding cryptographic response.

The biggest risk in extending to multiple web sites is a greater exposure to online phishing attacks. Now, a user has become accustomed to logging in to a variety of sites with the same mobile application, and they must be aware of the site at which they are authenticating. We associate a recognizable image with the web site and that image is displayed on the phone during login, as illustrated in Figure 5. The phone obtains the image at account creation time. Furthermore, recall that the cryptographic hash used to compute the response contains server information, so that the web site is certain that the mobile client was made aware of the correct login target.

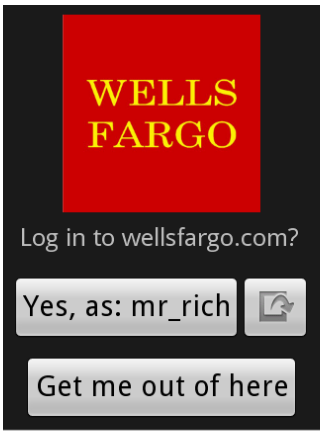


Figure 5: Confirmation of a Login

4.2 OpenID

We have implemented Snap2Pass as a custom OpenID provider. Many web sites today have adopted the use of OpenID, enabling single sign-on using their OpenID credentials. The key advantage is that all of the websites that support OpenID, known as the relying party, can enable Snap2Pass based login without requiring any code changes on their end. The user's credentials reside with an OpenID provider that uses Snap2Pass. We used Snap2Pass to log into several websites supporting the standard, including: Slashdot, ProductWiki, ccMixter, and LiveJournal. Upon typing in an OpenID account name to the web site of a relying party, the web page automatically redirects the user to the login page of the OpenID provider. In this case, our OpenID provider presents the user with the Snap2Pass QR code, and the login process proceeds as described in Section 3. Once the login process completes, the OpenID provider signals the result to the relying party web site.

4.3 OpenPass: Integrating Snap2Pass into OpenID

A benefit of Snap2Pass is its simple user interaction — a user no longer needs to type in any credentials at a participating website. Unfortunately, the first step of an OpenID login is to type in the user's OpenID address, so they may log in using their chosen identity provider. This defeats our goal of logging in with a single snap.

To address this issue we use a modified version of challenge-response. Now, the relying party is charged with creating the challenge. The phone sends its response to this challenge to a pre-configured identity provider, which then notifies the relying party of the transaction.

With OpenPass, a participating website presents the user with a QR code rather than an input field for an OpenID username. The QR code contains a challenge created by this relying party, as well as a response channel:

```
{    protocol:   "V3"
   , respondTo:
     "https://reliantparty.com/response"
   , challenge:  "e89c9fd66a5ec1a2d9"
}
```

The phone sends the response to this challenge to the user's pre-configured identity provider, and also forwards the reliant party's authentication endpoint:

```
{    protocol:  "V3"
   , challenge: "e89c9fd66a5ec1a2d9"
   , response:  "8fd6ef60acbe0d4193"
   , username:  "alice@myidprovider.com"
   , respondTo:
     "https://relyingparty.com/response"
}
```

The provider, possessing the user's shared secret, verifies the response and notifies the relying party at the requested endpoint. It indicates the user and challenge associated with the authentication attempt:

```
{    protocol:  "V3"
   , challenge: "e89c9fd66a5ec1a2d9"
   , username:  "alice@myidprovider.com"
   , provider: "myidprovider.com"
   , status: "OK"
   , token: "93aef90b7d0f5a7a96"
}
```

Finally, as in OpenID, the relying party verifies a token with the identity provider, using a shared secret. All the while, the user's browser waits for the transaction to complete, awaiting the relying party's response. In our implementation, we use XMPP's BOSH extension for this interaction. [23]

OpenPass realizes the goal of logging in with a single snap to any participating website. It re-introduces the tradeoff involving simple modifications at the relying party and the identity provider. To facilitate adoption, we integrate this technique into our implementation by modifying OpenID's Attribute Exchange extension and the identity provider protocol. The relying party login page now embeds the unique challenges within links to standard OpenID-based identity providers. The user selects his provider without needing to provide user credentials to the relying party. The user completes the OpenPass interaction with the provider, which signals the result to the relying party.

For scalability reasons, we assume that a pre-shared secret exists between the relying party and the identity provider similar to OpenID's stateful mode. The provider, in its response to the relying party, includes a token that authenticates all transaction data and is signed with the shared secret key. The relying party looks up with the shared secret using the provider's id as key and verifies the token. Alternatively, the identity provider maintains a verification time window for the relying party to verify the token at the cost of an extra round trip.

5. PAYMENTS

With Snap2Pass, we use the private storage of a mobile phone to create a secure payment experience. We can generalize the technique to improve our browsing experience in other ways.

5.1 Snap2Pay

When shopping at a small online retailer for the first time, the checkout page asks users to enter all their information (e.g. credit card number, billing address, shipping address, etc.) before the transaction can complete. This step is generally cumbersome and can cause shopping cart abandonment. In addition, there is some risk in sending all this sensitive information to a retailer the user has not seen before.

A generalization of Snap2Pass can help with both usability and security of online payments. The system, called Snap2Pay, functions as a digital wallet on the phone and interacts with the web site using QR codes. We first describe the user experience assuming the Snap2Pay application already has the user's payment information. We later explain how to automatically populate the phone with this data.

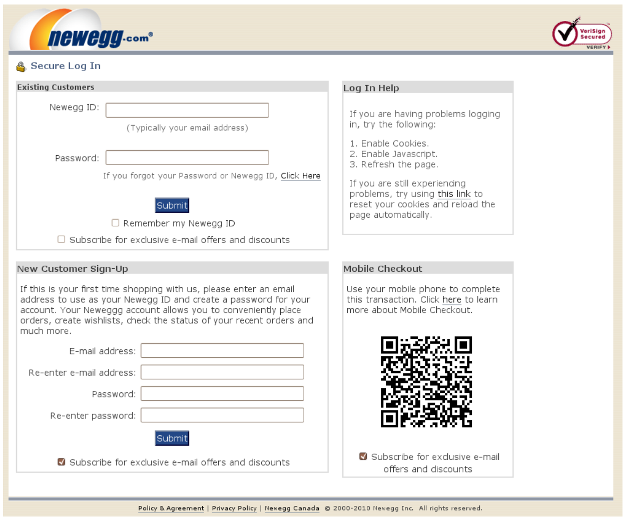
When making payments with Snap2Pay, the phone automatically contacts the user's bank and requests a one-time credit card number specific to the current retailer. This greatly reduces the risk of giving out the credit card number to an unknown retailer. Moreover, it enhances user privacy since it is more difficult for the retailer to track the user via credit card numbers. Combining this with other private browsing mechanisms, such as TorButton, gives the user a convenient way to shop online in private.

While one-time credit card numbers were introduced some time ago, they have had limited use primarily due to the manual labor required to generate them. With Snap2Pay one-time credit card numbers are built in and generated automatically by the system. As a result, the system is highly effective for interacting with small retailers or other questionable sites on the Internet.

Using Snap2Pay the checkout process works as follows:

- When the user's PC browser arrives at the retailer's checkout page, the page displays a QR code encoding transaction details, in addition to normal shopping cart information. The QR code encodes a response channel URL.
- Instead of manually entering personal information at the standard checkout page, the user can simply snap a picture of the QR code via her Snap2Pay phone application.
- Once the QR code is snapped the user is asked to confirm the transaction on the phone. Next, the phone securely obtains a one-time credit card number from the user's bank specific to that retailer.
- Next, the phone contacts the response channel URL on the retailer's site, and provides one-time payment information.
- The retailer completes the transaction, and redirects the user's PC browser to the transaction completed page. Our implementation uses XMPP's BOSH for this server-side push, as in Snap2Pass.

The Snap2Pay checkout process requires the user to a) snap a picture of the checkout QR code and b) confirm


Secure Log In

Existing Customers

Newegg ID:

(Typically your email address)

Password:

If you forgot your Password or Newegg ID, Click Here

Submit

☐ Remember my Newegg ID

☐ Subscribe for exclusive e-mail offers and discounts

Log In Help

If you are having problems logging in, try the following:

1. Enable Cookies.
2. Enable Javascript.
3. Refresh the page.

If you are still experiencing problems, try using this link to reset your cookies and reload the page automatically.

New Customer Sign-Up

If this is your first time shopping with us, please enter an email address to use as your Newegg ID and create a password for your account. Your Neweggg account allows you to conveniently place orders, create wishlists, check the status of your recent orders and much more.

E-mail address:

Re-enter e-mail address:

Password:

Re-enter password:

Submit

☑ Subscribe for exclusive e-mail offers and discounts

Mobile Checkout

Use your mobile phone to complete this transaction. Click here to learn more about Mobile Checkout.

☑ Subscribe for exclusive e-mail offers and discounts

Policy & Agreement | Privacy Policy | Newegg Canada © 2000-2010 Newegg Inc. All rights reserved.

Figure 6: A mockup of Snap2Pay in Newegg.com's checkout page.

the transaction on the phone. No other action is required. The main reason for doing this on the phone (as opposed to in the browser) is mobility: the user's payment data is available to use on any computer and any browser. No special hardware or software is required on the PC.

To mitigate phishing, we recognize that the above scenario is analogous to PC browsers, and similar solutions are applicable here. To mitigate phishing, the Snap2Pay phone application utilizes individual whitelisting and blacklisting to warn the user appropriately about malicious sites. If a retailer site does not appear on either list, the user is prompted to manually determine the retailer's validity for future transactions. By bootstrapping information about well-known retailers into the application, the user only needs to make such determination in the case of small or unknown entities in the long tail of retailers.

To complete the discussion of Snap2Pay, we explain how to populate the phone with the user's payment data. Past experience with digital wallets (e.g. Microsoft's Digital Wallet) suggests users do not take the time to enter their payment information into the wallet. Instead, with Snap2Pay, every time the user manually enters credit card information at an online retailer, the retailer displays a QR code containing that data. The user can simply snap the QR code to bootstrap the Snap2Pay database. Future transactions can use this data as explained above. We believe this process will make adoption much easier, but this can only be verified through a massive user study with the support of many online retailers.

5.2 Verified by Visa

Verified by Visa [21] and Mastercard SecureCode are, in effect, single sign-on services run by Visa and Mastercard that let merchants obtain user confirmation on requested transactions. When the user visits a merchant's checkout page, the browser is redirected to the user's bank where the user is asked to confirm the transaction with a password. The browser is then redirected back to the merchant where the transaction completes, provided a valid confirmation token is supplied by the bank. The resulting transaction is considered a "card present" transaction which is a strong incentive for merchants to adopt this system. This architecture is highly vulnerable to phishing and received much criticism [14].

Combining Snap2Pass with Snap2Pay can help improve the usability and security of Verified by Visa and Mastercard SecureCode. The mechanism is similar to how we integrate Snap2Pass with OpenID, as discussed in Section 4.2 and works as follows:

- In addition to standard transaction details, the merchant's checkout page includes a QR code that encodes the transaction amount plus a random challenge for a challenge-response protocol. The challenge also uniquely identifies the merchant.

- The user snaps the QR code with the Snap2Pass application and approves the transaction on the phone. The phone then sends a message to the user's bank containing the transaction amount, the random challenge from the merchant, and the response to that challenge (computed using the user's secret key stored on the phone). The message also includes account information such as the user's credit card number. Note that Snap2Pass is pre-configured at account setup to only send this message to the user's bank and nowhere else.

- The bank checks that the challenge from the merchant and the response from the phone, both contained in the message from the phone, are valid; namely, that the response from the phone is a valid response for the challenge. If so, it uses a merchant response channel URL (a well-known endpoint) to send to the merchant the verified by Visa confirmation token, which includes the random challenge contained in the message from the phone in addition to the standard fields.

- The merchant verifies the token from the bank and also verifies that the challenge in the token is the challenge that the merchant supplied in the QR code — this verification is needed to ensure that the phone answered the correct challenge. If all is valid, the merchant completes the transaction and transitions the browser from the checkout page to the transaction completed page.

Using this approach the random challenge is provided by the merchant (in the QR code), but is verified by the bank. The improved user experience is very simple: snap a picture of the QR code on the checkout page, confirm the transaction on the phone, and wait for the transaction to complete. Nothing needs to be typed in and no confusing redirections take place.

Since the user never supplies a credential to the merchant, this approach prevents offline phishing by a malicious merchant. Online phishing, discussed in Section 6.1, is still possible, but our geolocation-based defense described in that section applies here too.

6. EXTENSIONS

We now discuss several extensions to Snap2Pass to improve its security and to cope with the scenarios when the user forgets his phone, or forgets to log out.

6.1 Active Man in the Middle

The basic Snap2Pass does not prevent an active man in the middle attack such as online phishing. In an online phishing attack, the attacker creates a spoofed web site that constantly scrapes the target web site. The phisher lures users to the spoofed site and uses their responses to immediately login to their account at the target site. Once in, the phisher can take any action on the user's account. This attack easily defeats one-time password mechanisms and many phishing toolkits now work this way [6].

We minimize this attack vector by using geolocation information of the phone relative to the user's PC. Recall that in Snap2Pass the target web site communicates with the user's PC and with the user's phone. In normal use, the two are in close proximity. In an online phishing attack, the site communicates with the phishing server and the user's phone. The two are very likely to be far apart. Thus, the web site can use geolocation information to test if the two IP addresses it is seeing are in close proximity. If so, it allows the connection and if not it rejects it. Thus, for the phisher to succeed he must identify a victim user's location, find a compromised host close to the victim and place the phishing server there. While not impossible, in most phishing settings, this will be quite challenging for the phisher. Importantly, the phone's location measurement is not known to the web browser.

The above example works well when both the cell phone and user's PC operate are addressable, such as on wifi or wired networks. Commercial systems such as [11] offer geolocation databases claiming over 90 percent accuracy for resolving IP addresses to city locations. However, the cell phone is often not addressable, operating from the cellular provider's data network with an external gateway IP address. Cell phone IP addresses change frequently, and geographically diverse locations may operate under the same IP ranges. For example, a test user's Palo Alto, CA location resolved to one of T-Mobile's gateway IP addresses in Seattle, WA. The user's phone aids the geolocation system by providing GPS or cell tower ID data at transaction time. Furthermore, a complimentary approach involves exploiting application latency measurements to disambiguate cities operating under the same IP address range within a cellular data network [3]

We also may not have to rely on the IP network to determine the phone's location — most modern platforms can provide applications with relatively accurate location information. We expect the phone to cooperate with the authentication provider.

Another safeguard against the man in the middle is to require that sensitive transactions be verified on the mobile device. Here, the attacker gains access to the user's account and attempts to make a malicious transaction. The web site only allows this transaction to complete with confirmation from the phone, which the man in the middle cannot access. Using phones for transaction confirmation was previously studied in other projects [20, 9] and nicely complements Snap2Pass.

6.2 Logging in Without Phones

Although users will typically have their phones with them, an additional login method allows users with missing phones to gain entry to a webpage. This backup login method is treated as a password reset request. That is, to login without a phone requires solving a Captcha, responding to a selection of security questions, and retrieving a link sent to a primary email address.

6.3 Signing Out

It is difficult for a web site to know if a user has walked away from an authenticated session [19]. With Snap2Pass we can use the phone as a proximity sensor, powered by the device's location sensors or accelerometers. For example, when the phone detects motion above a threshold after authentication on the PC completes, it notifies the site. The site can then require re-authentication for subsequent requests. Thus, upon leaving an internet café, the user's session is immediately terminated. For web users on a moving train, the site may request one re-authentication and subsequently ignore motion notifications from the phone for the duration of the session.

More generally, with Snap2Pass a user can manually log out of all of her active sessions from her mobile phone, without returning to the abandoned terminal.

7. SECURITY ANALYSIS

We describe a number of attacks on the system and how they are addressed. Throughout the section, we assume that the login process and the subsequent session on the PC are served over SSL so that basic session hijacking (i.e. the attacker waits for authentication to complete and then hijacks the session) is not possible.

We first observe that with Snap2Pass, unlike passwords, a compromise at one web site does not affect the user's account at other sites. To see why, recall that in the symmetric scheme, Snap2Pass maintains a different shared secret with each site. In the public-key scheme, the site never stores the phone's secret key. Thus, in neither case does a compromise of one site affect another.

It is also worth noting that since the user never types in their password, Snap2Pass protects users against present day keylogging malware installed on the user's PC. Nevertheless, more sophisticated malware on the user's PC (e.g. [9]) can defeat Snap2Pass.

7.1 Offline Phishing

An offline phishing attack refers to a phisher who sets up a static spoofed web site and then waits for users to authenticate at the site. The term "offline" refers to the fact that the phisher scrapes the target web site's login page offline. For sites using password authentication, an offline phisher obtains a list of username/password that can be sold to others. We note that users who fall victim to this attack typically ignore information displayed in the address bar [7]. Consequently, the SSL lock icon or the extended validation colors in the address bar do not prevent this attack.

Snap2Pass clearly prevents offline phishing since the phisher does not obtain a credential that can be used or sold. In fact, the offline phisher gets nothing since the phone sends its response directly to the target web site. Recall that during account creation Snap2Pass records the target web site's address on the phone. During login, it sends the response to that address. Consequently, the offline phisher will never see the response.

7.2 Online Phishing and Active Man in the Middle

Online phishing is an example of an active man in the middle discussed in Section 6.1. The end result of the attack is that the phisher's browser is logged into the user's account at the target site. As in the offline phishing case, we cannot rely on security indicators in the browser chrome to alert the user to this attack. In section 6.1, we discussed how Snap2Pass uses geolocation to defend against this attack.

It is also worth noting that this attack is easily defeated using a PC browser extension. The extension would retrieve the SSL session key used in the connection to the web site (i.e. the phishing site) and embed a hash of this key in the QR code (if the connection is in the clear the data field would be empty) along with the extension's digital signature on the hash. The phone would verify the signature and then send the hash to the real site along with its response to the challenge. The web site would now see that the browser's SSL session key (used to communicate with the frontend of the phishing server) is different from its own SSL key (used to communicate with the backend of the phishing server) and would conclude that a man in the middle is interfering with the connection. The reason for the extension's signature on the hashed key is to ensure that the phisher cannot inject its own QR code onto the page with the "correct" key in it. An alternative to a digital signature is to place the QR code containing the hashed key in the browser chrome (e.g. in the address bar) where the phisher cannot overwrite it with its own data.

We chose to not implement this defense since Snap2Pass is designed to work with existing unmodified browsers.

7.3 Phone Theft and Key Revocation

If a phone is lost or stolen, that phone can potentially be used to impersonate the user at all websites where the user has an account. Snap2Pass mitigates this issue in two ways.

Firstly, the Snap2Pass application can require the user to authenticate to the phone before the application can be used. Rather than implement an unlock feature in Snap2Pass we rely on the phone's locking mechanism for this purpose. Users who worry about device theft can configure their phones to require a pass code before applications like Snap2Pass can be launched. This forces a thief to first override the phone's locking mechanism. Moreover, several phone vendors provide a remote kill feature that destroys data on the phone in case it is lost or stolen.

Secondly, when a phone is lost, users can easily revoke the Snap2Pass credentials on the phone by visiting web sites where they have an account and resetting their Snap2Pass credentials at those sites. This results in a new keying material generated for the user thus invalidating the secrets on the lost phone.

8. IMPLEMENTATION

Our implementation of Snap2Pass includes server-side code, called a provider, and a mobile client. The provider and the client provide a reference implementation for the server and client ends of the Snap2Pass protocol, respectively.

8.1 Provider

The provider is implemented as a custom OpenID provider and offers server-side challenge/response functionality as described above. OpenID is a popular protocol for federated identity management and single sign-on. With the addition of a layer of indirection, it enable tens of thousands of existing OpenID consumer web sites to use Snap2Pass without requiring modification of their server-side login protocols.

The provider implementation makes use of the Joid open source project, and is written in Java. It is loosely coupled to Joid; thus it can be plugged into other standard OpenID providers. The custom provider consists of a symmetric-key based challenge response system, account management and a web portal. The challenge response modules are written in Java using built-in crypto libraries. It includes modules for symmetric key generation, and HMAC-based challenge/response creation and verification. The account management modules manage user accounts, provide persistence and include a cache for fast lookup of incoming responses.

The web portal adds QR code features to the OpenID provider. It includes custom registration and login pages, implemented as Java Server Pages (JSP) to support the Snap2Pass account creation and login protocol. On completion of the login protocol, the web portal integrates with the provider backend to signal the result using the OpenID protocol. This enables existing OpenID consumer sites to support Snap2Pass with no code changes.

The server needs to notify authentication attempt results with the browser using either pull or push techniques. We chose a server-side push technique, using XMPP BOSH extension [23]. For this XMPP connection, we use the StropheJS Javascript library to connect the browser, and the Smack XMPP JAVA library for the server.

The provider module has approximately 1,600 source lines of code (SLOC).

8.2 Mobile Client

The mobile client is written in Java for the Android environment. It implements the client-side Snap2Pass protocol, and offers functionality for credential management and symmetric key challenge/response computation. We use Android's SharedPreferences API to store and manage credentials retrieved from the provider. In a production implementation, the credentials are managed using a secure credential manager. The login module uses built-in APIs to compute responses to challenges. We use Android's intent system and the ZXing project to scan and consume QR codes. For improved security, the scanning functionality will be embedded directly in the application. The mobile client has approximately 400 SLOC.

9. EXPERIMENTAL RESULTS

9.1 User Study of Snap2Pass

We compare our implementation against existing secure authentication mechanisms. Our goal is to provide a system that is more consumer-friendly than what is currently available without negatively affecting security. There are two main usability concerns: a) Snap2Pass is intuitive enough to be used by non-technical users and b) frequent logins are as fast and effortless as possible.

9.1.1 Using QR Codes

We presented our approach to new users to gauge their reaction. Our subjects ranged from savvy smartphone users to those who are considerably less technical.

We found that the UI design for pinpointing a QR code is important. The visual feedback on the phone's screen was helpful in guiding the user to a successful login. Sometimes, a user would believe a code was recognized by the software before it was actually able to locate it, causing some confusion on early attempts. Subsequent uses were much faster and free of this confusion. Usually two or three sample trials were enough for the user to become comfortable with the system. Most users, including the non-technical, agree that the approach is "very simple" and usable. Several even found it fun to use.

The amount of time required for the software to locate the barcode is important for the system's usability. We compared the user experience for the same application across three hardware platforms: the HTC G1, the Motorola Droid, and the HTC Nexus One. As the platform's power increased, the scanning process became noticeably quicker, thereby improving overall user experience. We did not experiment with other barcode formats or software solutions — scanning QR codes with the ZXing application proved sufficiently usable.

9.1.2 Comparing Authentication Techniques

We wanted to gauge both the usability and perceived security of Snap2Pass as compared to other authentication methods. We ran a user study across 30 users of different backgrounds. 15 of our users were female. 16 of our users were in a technical field.

Our study compared two web authentication methods enhanced with a cell phone — Snap2Pass, and a system much like RSA SecurID. With the SecurID system, participants used a mobile application to retrieve six randomly generated digits. They were told that, in deployment, this number would be synchronized between the phone and the authentication server. To log in to a web page, then, the participant entered their username, password, and this six-digit string.

We demonstrated the two systems to each user once before having them try for themselves, twice each. We then had them fill out a short survey about their experience.

To determine the perceived security of authentication systems, we asked how safe they would feel using a given system with their primary bank, from 1 (not safe at all) to 10 (completely safe). As a baseline, we asked about using a standard username/password scheme. The average score was 5.9. We then asked about SecurID and Snap2Pass; SecurID came out slightly better than Snap2Pass— 7.7 as compared to 7.3. Both were perceived to be noticeably more secure than username/passwords.

Of Snap2Pass, one user writes, "It was really cool. Felt very secure, like something out of a James Bond movie." She also agreed that SecurID felt secure, "but entering another field was very tedious, and I imagine would be pretty rough to have to do frequently."

To evaluate the usability of each system, we first asked how easily users found learning each system, on a scale from 1 (very difficult) to 10 (trivial). We found that users had no trouble learning either system; Snap2Pass was given an average of 9.2, and SecurID a 9.1.

We then asked how they found repeated logins with each system, from 1 (very tedious) to 10 (very easy). Here, Snap2Pass fared better than SecurID — 8.5 as compared to 6.9. A user who was skeptical of the system's security commented that "Snap2Pass does not seem to be safe but given the fact that it's sooo easy to use, I will be happy to use it as main login method."

Next, we asked how a participant would feel about using each system in practice. For the two systems, participants were asked, assuming they had a capable phone, if they would prefer to use each system with their primary bank as compared to username/passwords. They were given three choices: "I would not want to use this system," "I would use this system only if my bank made me," and "I would prefer to use this system." The resulting numbers are:

	Snap2Pass	SecurID
prefer the system over pwds:	15	13
would only use if required:	11	12
would prefer passwords:	4	5

Finally, we asked participants how they would feel about using a Snap2Pass-like system for making purchases on the web; in such a system, a user would point their phone at their shopping cart's checkout page to complete the transaction, without typing in billing information. We posed this question to 22 participants, and 13 indicated they would prefer such a system to entering billing details; 2 would do so only if required by their bank, and the remaining 7 would not want to use such a system.

9.1.3 User Concerns

By far, the most common concern with both demonstrated systems was an unavailable or stolen phone. Indeed, a system in deployment must be robust to such incidents, as we discuss in sections 6.2 and 7.3 respectively. Many users also stressed that they would only trust the security of Snap2Pass with suitable on-phone security. A participant writes, "(I am) concerned about phone security. I love the idea of using my phone to store/share personal info, but I do not trust the current security on my phone." Another says, "I feel that if someone stole your phone, they could easily gain access with this technology to whatever password protected account you have." For a system in deployment, this concern must be suitably addressed.

9.1.4 Comparison for an Experienced User

We also wanted a more quantitative comparison of secure authentication systems. We compared the login times required for Snap2Pass with the times for other authentication systems. Our focus was on comparing the amount of time required for an experienced user to complete a successful authentication. We compared with two security systems in active deployment: RSA SecurID and CRYPTOCard. We simulated the logins by providing phone interfaces for all three protocols.

For SecurID, our workflow was similar to a standard username/password login, with the addition of a "token" field. This token was generated on the mobile device, and the generation would be synchronized with the authentication server in deployment. Our CRYPTOCard simulation involved an extra step. First, the user submits his username to the authentication server. The server responds with a random 6-character hex string, which the user must enter into his mobile device. The mobile device then generates a 6-character hex response, which the user inputs into the web form, along with his password. In our scheme, a user visits the authentication page, which presents a QR code. The user runs the "Log In" routine on his phone and scans the QR code, completing the authentication.

For SecurID and CRYPTOCard, we entered a username and password that the user is familiar with. The username consisted of 8 lowercase characters, and the password was also 8 characters, consisting of upper- and lower-case letters, numbers, and punctuation marks.

For each scheme, we issued twenty logins and recorded the amount of wall clock time required, from the time the login page loaded to the time the secured content was displayed. We assumed that the second factor device was at hand and the required application can be launched with one button press. Figure 7 shows the results of the user study. The error bars indicated were computed by the standard formula $\frac{\sigma}{\sqrt{n}}$, where σ is the standard deviation and n is the number of inputs.

The study shows that an experienced user takes about 7 seconds to log in using Snap2Pass. Overall, Snap2Pass was approximately 2.5 times faster than the CRYPTOCard system, and 30% faster than RSA tokens. Most of the time required for logging in to both the SecurID and CRYPTOCard systems was spent inputting data into forms, and copying values between devices–a process that is limited by the human's typing and reading abilities. For Snap2Pass, most of the time was machine bounded. Our application required approximately two seconds on average to load the phone's camera module from an idle state. Most of the remaining time was spent in the ZXing libary trying to locate and decode the QR code. A visually apparent challenge for the library was auto-adjusting the aperture to read information from an active display as opposed to a passive one. These bottlenecks can be improved upon both in software and with upgraded device profiles, as opposed to end-user efforts. Notably, these tests were performed using an HTC G1, which newer devices like the Nexus One already outperform significantly.

Both the SecurID and CRYPTOCard systems involve human-entered form data, a process that is prone to errors. In our user study, one out of twenty logins for both systems resulted in a failed login, essentially doubling the amount of time required to log in. Once the user has authenticated with his phone, our scheme does not suffer this shortcoming, as all data is parsed, calculated, and submitted by the devices directly.

10. RELATED WORK

10.1 Mobile/Web Authentication

Using cell phone for authenticating people is an old idea. Wu et al. [22] and Oprea et al. [15] use phones to help establish a secure session on an insecure device. Snap2Pass addresses a different threat scenario where the user is contacting a remote server on his own PC. Our threat model is very common and is simpler than in [22, 15] which enables

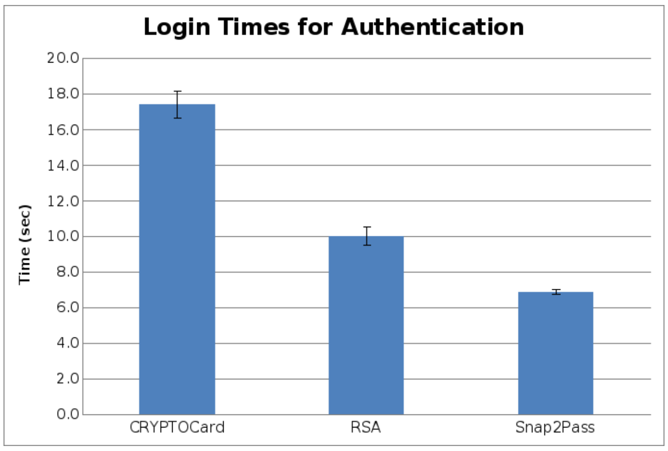


Figure 7: Average login time required across three secure authentication schemes.

us to design a more efficient solution that is easier for the user.

Phoolproof [16] is designed as an anti-phishing authentication mechanism. With Phoolproof, Parno et al. require custom software on the PC as well as a bluetooth connection. Snap2Pass in contrast, requires no modification to the PC and provides security using information available on smartphones that was not available at the time Phoolproof was designed.

Aloul and Zahidi discuss the use of one time passwords generated on a mobile phone for use with ATMs [2]. By using modern smart phones, we are able to provide a system that is easier to use without compromising security.

Bellovin et al. [4] propose EKE, a protocol for shared key exchange, which has been refined further. In contrast, we rely on QR codes over SSL to transfer the shared secret between the PC browser and the phone to manage account credentials.

10.2 Device Pairing

Mccune et al. [13] have previously combined phone cameras and two dimensional barcodes for transmitting public keys from one device to another. Other clever approaches to device pairing use the accelerometer for generating a shared key [12, 10] or for proving proximity [5].

In Snap2Pass and Snap2Pay, we present a novel system and applications that are based on using a visual communication channel, to present and transfer a challenge from one device to another. More importantly, the visual channel contains a reference to a communication endpoint that the mobile device uses to respond to the challenge. The combined effect is that the act of scanning a barcode becomes a user interaction with the site hosting that barcode.

In [17], Pierce et al. explore the possible uses of pairing a mobile with a PC, presenting several useful utilities. Their model uses a centrally managed account to discover services, as opposed to our ad-hoc pairing technique.

11. CONCLUSIONS

We described Snap2Pass, an easy-to-use authentication system that defeats many of the attacks on traditional password schemes on the Web. Snap2Pass is implemented as a custom OpenID provider, thereby immediately enabling usage on the tens of thousands of websites that accept OpenID-based authentication, without any server-side code changes. We have extended the OpenID protocol so that the user can simply snap a QR code presented by a relying party without having to enter user credentials on the login page.

We also presented Snap2Pay, which allows consumers to use one-time credit card numbers with just a snap of a QR code. One-time credit card numbers are useful for reducing the risk of interacting with small retailers or questionable sites on the Internet. Snap2Pay eliminates the manual labor involved, which has so far limited the adoption of the technique. Similarly, Verified by Visa and Mastercard SecureCode are single sign-on services which have not been adopted because they are highly vulnerable to phishing. We showed in this paper how the Snap2Pass technology improves both their usability and security.

Our user studies show that the system is fast to use and easy to learn. Even without much experience with smartphones, users can easily use Snap2Pass after seeing it done only once. The comparison with RSA tokens indicates that users tend to perceive more cumbersome techniques to be more secure; however, users care more about ease of use than security. Snap2Pass has the advantage that it is both secure and easy to use.

Snap2Pass can be used in an off-the-shelf PC browser with no modifications, and works well with all popular browsers today. We have an open-source implementation of Snap2Pass at `http://snap2.stanford.edu`.

12. REFERENCES

[1] T. I. A. D. C. (ADC). Consumer password worst practices, 2009. `www.imperva.com/download.asp?id=239`.

[2] F. Aloul and S. Zahidi. Two factor authentication using mobile phones.

[3] M. Balakrishnan, I. Mohomed, and V. Ramasubramanian. Where's that phone?: geolocating ip addresses on 3g networks. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 294–300, New York, NY, USA, 2009. ACM.

[4] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE SYMPOSIUM ON RESEARCH IN SECURITY AND PRIVACY*, pages 72–84, 1992.

[5] Bump technologies. `bumptechnologies.com`.

[6] M. Cova, C. Kruegel, and G. Vigna. There is no free phish: An analysis of "free" and live phishing kits. In *Proc. of 2nd Usenix wOOt '08*, 2008.

[7] R. Dhamija, D. Tygar, and M. Hearst. Why phishing works. In *Proc. of CHI 2006*, pages 581–590, 2006.

[8] D. Florencio and C. Herley. A large-scale study of web password habits. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 657–666, New York, NY, USA, 2007. ACM.

[9] C. Jackson, D. Boneh, and J. Mitchell. Transaction generators: Root kits for the web. In *proceedings of the 2nd USENIX Workshop on Hot Topics in Security*, 2007.

[10] D. Kirovski, M. Sinclair, and D. Wilson. The martini synch. Technical report, Microsoft Research Technical Report, MSR-TR-2007-123, 2007.

[11] Maxmind. `maxmind.com`.

[12] R. Mayrhofer and H. Gellersen. Shake well before use: Authentication based on accelerometer data. In *5th International Conference on Pervasive Computing*, volume 4480 of *LNCS*, pages 144–161, 2007.

[13] J. M. Mccune, A. Perrig, and M. K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *In IEEE Symposium on Security and Privacy*, pages 110–124, 2005.

[14] S. Murdoch and R. Anderson. Verified by visa and mastercard securecode: or, how not to design authentication. In *Proc. of Financial Cryptography*, 2010.

[15] A. Oprea, D. Balfanz, G. Durfee, and D. Smetters. Securing a remote terminal application with a mobile trusted device. In *Proc. of ACSAC'04*, pages 438–447, 2004.

[16] B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. In *Proceedings of the 10th International Conference on Financial Cryptography and Data Security (FC'06)*, 2006.

[17] J. S. Pierce and J. Nichols. An infrastructure for extending applications' user experiences across multiple personal devices. In *UIST '08: Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 101–110, New York, NY, USA, 2008. ACM.

[18] M. V. Rafter. A breakout year for openid, 2009. `technology.inc.com/security/articles/200902/openID.html`.

[19] B. Schneier. Unauthentication, 2009. `www.schneier.com/blog/archives/2009/09/unauthenticatio.html`.

[20] D. Steeves. Securing online transactions with a trusted digital identity. First TIPPI workshop, 2005.

[21] visa. Verified by visa, 2000.

[22] M. Wu, S. Garfinkel, and R. Miller. Secure web authentication with mobile phones. In *DIMACS Workshop on Usable Privacy and Security Software*, 2004.

[23] Xep-0206: Xmpp over bosh. `xmpp.org/extensions/xep-0206.html`.