Fundamental Security Flaws in the 3-D Secure 2 Cardholder Authentication Specification

Francisco Corella Karen Lewison

October 28, 2019

Abstract

The 3-D Secure protocol, introduced by Visa in 1999 and adopted by other credit card networks, provides security for online credit card transactions by redirecting the cardholder's browser to the web site of the issuing bank, where the cardholder authenticates with an ordinary password or a one-time password. This however creates friction that may cause transaction abandonment. Over the last few years the credit card networks have been developing 3-D Secure 2, where the issuing bank assesses fraud risk based on information received from the merchant through a back channel and waives authentication for low-risk transactions. In previous work we have shown that 3-D Secure 2 has serious privacy and usability issues and proposed an alternative protocol that provides strong security without friction for all transactions by cryptographically authenticating the cardholder. Here we show that 3-D Secure 2 has fundamental security flaws that may allow the merchant to impersonate the cardholder, when used in a configuration where the cardholder uses a native app rather than a browser to access the merchant's site. These are not minor flaws that can be fixed by minor changes; fixing them would require a different approach to cardholder authentication in that configuration.

Contents

1 Introduction

Providing strong security for online credit card payments has been an open problem since the World Wide Web started to be used for electronic commerce in the mid nineties. Securing payments was the main motivation for the introduction of SSL by Netscape in 1994 [1]. SSL and its current successor TLS 1.3 [2] make it possible for the cardholder to send credit card data to the merchant over a secure channel, protecting it from eavesdropping on the Internet. The data printed on a credit card, which includes the credit card number, the expiration date, the security code and the cardholder's name, has a fair amount of entropy; but it is not a secret, since it is known by all merchants to whom the cardholder has made payments online or in-store, and it may be available to fraudsters on the dark web after having been exposed by security breaches. Hence authentication of the cardholder based on his or her knowledge of credit card data does not provide strong security.

A serious effort to provide strong security was made in the late nineties as Visa and MasterCard collaborated with major computer industry players on the design of the Secure Electronics Transactions (SET) protocol [3]. SET was a novel cryptographic protocol, which attempted to provide cardholder privacy in addition to security by restricting the amount of cardholder information provided to transaction participants on a need-to-know basis. But the effort failed and SET was abandoned.

The simpler 3-D Secure protocol [4] was then introduced by Visa in 1999. It was adopted by other credit card networks and is still being used today. In 3-D Secure version 1 (3DS1) the merchant redirects the cardholder's browser to the web site of the issuing bank where the cardholder authenticates, traditionally with a password, today more commonly with a one-time password sent by email or text messaging. This provides security but creates friction that may cause transaction abandonment. For that reason deployment of 3DS1 has been limited; it is rarely used in the US and unevenly used in other countries.

To address the problem of friction the credit card networks have tasked EMVCo with the specification of a new version of 3-D Secure, called 3-D Secure 2 (3DS2). The latest version of the specification (2.2.0 as of this writing), comprising a Protocol and Core Functions Specification (hereinafter the "Protocol Specification") and an SDK Specification, can be found in [5]. Figure 2.1 of the Protocol Specification shows the components of 3DS2 and the communications that take place between them.

3DS2 introduces a "frictionless flow" where the merchant sends transaction and cardholder information to the issuing bank through a back channel that circumvents the cardholder's browser. The issuing bank uses the information to assess the risk that the transaction may be fraudulent and waives authentication for low risk transactions. If a transaction is deemed high risk, the frictionless flow is followed by a “challenge flow” where the cardholder’s browser is redirected to the issuer’s web site, which authenticates the cardholder as in 3DS1. The terms “authentication flow”, “processing flow” and “authentication process” are used interchangeably to refer to the frictionless flow conditionally followed by the challenge flow.

3DS2 also has provisions for allowing the cardholder to use a native bank app and/or a native merchant app if made available by the issuing bank or the merchant.

In previous work [6] we have shown that that 3-D Secure 2 has serious privacy and usability issues, and we have proposed a cryptographic protocol for cardholder authentication, very different from 3-D Secure, that can be used to secure all transactions, both low and high-risk ones, without creating any friction.

Here we show that 3-D Secure 2 has fundamental security flaws, some of which may allow the merchant to impersonate the cardholder, when it is used in a configuration where the cardholder uses a native app to access the merchant’s site rather than a web browser. These are not minor flaws that can be fixed by minor changes; fixing them would require a different approach to cardholder authentication in that configuration.

The rest of the paper is organized as follows. Section 2 explains how 3DS2 works in a configuration where the cardholder uses a merchant app. Section 3 analyzes and dispels three misconceptions that are apparent in the 3DS2 specification and seem to be the root cause of the security flaws. This analysis may be of interest in its own right, as the fact that these misconceptions have not been challenged until now, after the specification has been under development for several years and has presumably been reviewed by many banks and merchant processors, seems to indicate that they are widespread. Section 4 describes the security flaws and Section 5 recapitulates.

2 Use of a merchant app in 3-D Secure 2

When the cardholder uses a merchant app to shop on the merchant’s site the frictionless flow proceeds very much as when the cardholder uses a web browser, but the challenge flow is very different. The merchant app communicates directly with issuing bank, or more precisely with an “Access Control Server (ACS)” in the “Issuer Domain” that provides an interface between the bank and the 3DS2 infrastructure. The merchant app mediates a challenge-response interaction between the cardholder and the ACS, interacting with the cardholder using either a native user interface (UI) or an HTML UI implemented in a web view created by the app, and interacting with the bank through a sequence of Challenge Request (CReq) and Challenge Response (CRes) messages whose contents depend on the method that the bank uses to authenticate the cardholder.

The 3DS2 specification does not prescribe specific authentication methods, but envisions and provides examples of two methods: (i) security questions, and (ii) a security code used as a one-time password (OTP), sent to the cardholder in an email message or a text message. In both methods the merchant app initiates the interaction by sending a CReq message that requests a challenge. In the security questions method the ACS responds with a CRes messages that conveys a question. The merchant app displays the question to the cardholder, as shown in the Sample Native UI Template of Figure 4.11 of the Protocol Specification, obtains the answer from the cardholder, and sends the answer to the ACS in a CReq message. Additional questions and answers are sent in subsequent CRes and CReq messages. (Confusingly, the questions are sent in Challenge Response messages and the answers in Challenge Request messages.) In the OTP method the ACS responds to the initial CReq message with a CRes message that asks the cardholder how the OTP should be sent, as shown in the Sample Native UI Template of Figure 4.10. The merchant app sends the cardholder's choice in a CReq message. The ACS sends the OTP by email or text messaging as chosen by the cardholder and responds to the CReq message with a CRes message that prompts the cardholder for the OTP, as shown in the Sample Native UI Template of Figure 4.9. The cardholder enters the OTP and the merchant app then sends it to the ACS in a CReq message. (Confusingly again, the prompt is sent in a Challenge Response message and the OTP in a Challenge Request message.)

Since the merchant app obtains authentication information that the cardholder uses to authenticate to the bank, and the merchant app is provided and controlled by the merchant, a question comes to mind: *does that authentication information allow the merchant to impersonate the cardholder* vis-à-vis *the bank?*

The merchant is required to implement the 3DS2 functionality by embedding an EMVCo-approved "3DS SDK" in the merchant app. (The term Software Development Kit (SDK) is used in the specification to refer to a code library, presumably made available to merchants as part of a development toolkit.) It is the SDK that implements the challenge-response interaction described above, obtaining the authentication information from the cardholder and sending it to the issuing bank. This means that an honest merchant who complies with the requirement to use an approved SDK will not see the authentication information. However we shall see below that a malicious merchant will be able to obtain the authentication information in several ways.

3 Security misconceptions

The security flaws in the configuration of 3DS2 where the cardholder uses a merchant app seem to be due to three security misconceptions that are apparent in the specification.

3.1 Misconception 1

One misconception is the idea that a "3DS Integrator" trusted by the payment networks can enforce the use of an EMVCo-approved 3DS SDK embedded in the merchant app to implement the 3DS2 functionality. The 3DS Integrator is responsible for supplying the SDK and for provisioning a "3DS Server", defined as follows in Table 1.3 of the Protocol Specification:

> *3DS Server: Refers to the 3DS Integrator's server or systems that handle online transactions and facilitates communication between the 3DS Requestor and the DS.*

(The term "3DS Requestor" is used in the specification to refer the merchant or to the merchant's site, and the acronym DS refers to a Directory Server, provisioned by one of the payment networks. The 3DS Server, the 3DS Requestor and the DS can be seen on Figure 2.1 of the Protocol Specification along with other components of the 3DS2 architecture.)

This misconception can be seen on page 46 of the Protocol Specification, where there is a requirement labeled “Req 7”, stating that

> *The 3DS Server shall: ... Verify the authenticity of the 3DS SDK as defined in Section 6.2.1.*

Here is what Section 6.2.1 says:

> *3DS Requestors deploy an EMVCo-approved 3DS SDK embedded in their App and are required to have a mechanism to authenticate the 3DS Requestor App to the 3DS Requestor, including confirmation that the embedded 3DS SDK has not been changed.*

Notice the sleight of hand: Section 6.2.1 is about code authentication to the 3DS Requestor, i.e. to the merchant, whereas Req 7 is about code authentication to the 3DS Server owned by the 3DS Integrator. A mechanism usable by the merchant to authenticate its own app would not be usable by the 3DS Integrator to verify that a malicious merchant has not modified the 3DS SDK.

In fact it is impossible to implement Req 7 in today’s mobile app ecosystem, where code integrity is ensured by code signing. The authenticity of a native app is verified by a trusted app store using a signature provided by the developer of the app; then, after the app has been downloaded to a mobile device, the operating system of the device is responsible for maintaining the integrity of the app code. The signature provided by the developer covers the entire app. There is no way for a supplier of a code library embedded in the app to sign the library and have the signature verified by the app store. In theory it would be possible for the operating system to verify a signature on a code library used by a native app, but mobile operating systems such as iOS or Android do not provide such functionality. Thus there is no way for the 3DS Server, or for the 3DS Integrator that provisions the 3DS Server, to verify that the 3DS SDK code library has not been modified by a malicious merchant.

Furthermore, it would be useless for the 3DS Integrator to verify that the approved SDK is embedded in the merchant’s app and has not been modified, without also verifying that it is the approved code that is actually used to implement the 3DS2 functionality. A malicious merchant could embed in the app both the approved and unmodified SDK and a maliciously modified SDK or some other malicious code, and use the malicious code instead of the approved code to implement the 3DS2 functionality. It could also use the approved code most of the time but use the malicious code some times, e.g. when trying to capture the authentication information of a particular cardholder. It could also embed and use only the approved code in the original version of the app, but later embed malicious code in an app update and use the malicious code from time to time, with obfuscation to prevent detection. To detect such attacks, the 3DS Integrator would have to review the code of the app as originally placed by the merchant in the app store and the code of every subsequent update to the app. In the age of agile development, this would be utterly impractical.

3.2 Misconception 2

Another misconception is the idea that an image displayed in a web view can provide protection against phishing attacks. This misconception can be seen on page 100 of the Protocol Specification, where Figure 4.18 shows an HTML UI Template, rendered by a web view, that asks a security question and displays an image above the question with the instruction "if the image to the right is correct please answer the question below".

Some web sites display a registered image that the user chooses when creating an account, as part of some scheme for authenticating the site to the user and providing protection against phishing attacks, even though it has been repeatedly shown that such schemes are ineffective or counterproductive.

In one such scheme, the user is asked for a username, then the image associated with the username is displayed, then the user is asked to enter the password after verifying the image. The motivation for this is that a user who is sent to a password-phishing site by a phishing email, and does not notice that the address bar of the browser does not show the URL of a legitimate site that the user expected to access, may notice instead that the image is missing or incorrect and refrain from entering the password. But it has been shown that most users fail to notice the absence of the registered image [7]; and, more importantly, it has been pointed out that, if the user does not pay attention to the address bar, a simple man-in-the-middle attack allows the attacker to obtain the registered image from the legitimate site and display it on the phishing site [8], thus helping the attacker by making the phishing site look trustworthy even though the URL of the site is not the expected one.

A more complicated anti-phishing scheme called SiteKey, which combined a registered image and/or phrase with a browser cookie and a Flash cookie, was used by large financial institutions but was discontinued after it was shown to be vulnerable to a similar man-in-the-middle attack [9].

The HTML UI Template of Figure 4.18 is an adaptation of the ill-conceived idea of using a registered image to authenticate a web site to the case where a user accesses an Internet site using a web view instead of a browser. A web view does not have an address bar, so, if one ignores the possibility of a man-in-the-middle attack, using an image to authenticate a site seems a clever way of dealing with the fact that the URL of the site is not shown. But when a man-in-the-middle attack takes place, a user who sees a registered image is misled into thinking that the phishing site is legitimate while being unable to see the URL of the site and thus having no means of detecting the attack.

Furthermore, presenting the registered image as a means of authenticating a site, as is done in Figure 4.18 by asking the user to answer the question only if the image is correct, gives the user the false impression that he or she is visiting a web page of a site, and that any information entered into the page will only be seen by the site. This is true when a site is accessed through a web browser, which enforces the same-origin origin policy of the Web. But it is not true when the site is accessed through a web view. As we shall see in the next section, the native app that hosts the view has access to any information entered into the view, and can thus capture such information without having to set up a phishing site.

Of course a browser can also see any information entered by the user in a web page; but the browser is trusted by the user for all purposes, whereas a native app should only trusted for some specific purposes. In particular, a merchant app may be trusted to see credit card data submitted to pay for a purchase, but should not be trusted to see information such as answers to security questions. Luring a cardholder into installing and using a malicious merchant app to capture information used by the cardholder to authenticate to the issuing bank could be viewed as a new kind of phishing attack, made possible by 3-D Secure 2. Displaying an image registered by the cardholder with the bank in a web view would facilitate such an attack.

3.3 Misconception 3

A third misconception is the idea that the use of a web view by the 3DS SDK somehow "isolates" the non-SDK code of the merchant app from the challenge-response interaction described above in Section 2. This misconception can be seen in the following paragraph on page 96 of the Protocol Specification:

> *A secure communication channel is established between the Consumer Device and the ACS for routing all the network communications. By defining all interaction with the web view in terms of the SDK, it clearly indicates that the SDK defines and owns the UI, and that the 3DS Requestor's App is isolated from the challenge interaction.*

There is indeed a secure channel between the Consumer Device and the ACS, or more specifically between the SDK and the ACS, set up by an elliptic curve Diffie-Hellman key exchange described in Section 6.2.3 of the Protocol Specification. This channel is used to encrypt the CReq and CRes messages of the challenge-response interaction with an authenticated encryption algorithm as described in Section 6.2.4 of the Protocol Specification; and the encrypted CReq and CRes messages are further protected by being sent over TLS connections as described in Section 6.1.4.1 of that same document.

At each step of the challenge-response interaction, the contents of the web view are determined by a CRes message received from the issuing bank, and the information entered into the view by the cardholder is transmitted to the bank in a CReq message. The above-quoted paragraph seems to argue that, from the facts that the CRes and CReq messages are encrypted and the 3DS2 specification "[defines] all interactions with the web view in terms of the SDK", it follows that the "3DS Requestor's App", i.e. the merchant app, or more precisely the non-SDK code of the merchant app, is "isolated from the challenge interaction". But this is a *non sequitur*. Even if all the interactions with the web view described in the 3DS2 specification are performed by an EMVCo-appproved SDK and the CRes and CReq messages are encrypted and further protected by TLS as specified, the non-SDK portion of the code of a malicious merchant app may very well be able to access the contents of a web view after they have been received in a CRes message and decrypted, and the information entered by the cardholder into the web view before it is encrypted and sent out in a CReq message, thus breaking the claimed isolation of the non-SDK code from the challenge-response interaction.

This is indeed the case. Both iOS and Android allow a native app to run JavaScript code inside a web view by calling the `evaluateJavascript` method of the web view object. That JavaScript code has access to the Document Object Model (DOM) that defines the HTML contents of the view, and is therefore able to read the contents of the view and any information entered by the user into input elements of the view. The JavaScript code can then exfiltrate any view contents or entered information using a callback function.

This requires access to the web view object, but the non-SDK portion of the merchant app does have such access. Indeed, as shown in the HTML UI Templates of Section 4.2.6 of the Protocol Specification, the web view is the second component of a layout, whose first component (which contains the "Cancel" button and the "SECURE CHECKOUT" title seen in the templates) is "managed by the 3DS Requestor", i.e. by the non-SDK code of the merchant app. Thus, while the web view is populated by the SDK, it is created by the non-SDK code as part of that layout. Therefore the non-SDK code has a reference to the web view object and can run JavaScript code inside the view by calling the `evaluateJavascript` method of the object.

That JavaScript code can exfiltrate cardholder authentication information, such as security questions and answers or an OTP, and provide it to the non-SDK code, thus making it available to the merchant.

4 Security flaws

When used in a configuration where the cardholder uses a merchant app, 3-D Secure 2 has security flaws that enable attacks by a malicious merchant, or by malicious personnel working for the merchant. Variations on the first two attacks described in this section can also be carried out by a malicious developer of merchant apps, or by a hacker with remote write access to the code of the merchant app before it has been signed.

4.1 The merchant can capture answers to security questions and use them to impersonate the cardholder

3-D Secure 2 envisions authenticating the cardholder by asking a sequence of security questions. The specification gives two examples of such questions, shown in Figures 4.11 and 4.18 of the Protocol Specification:

- "What cities have you lived in?", which is answered by selecting names of cities from a list, and
- "What is the name of your first pet?", which is answered entering the name of the pet in a text input field.

According to the specification, the EMVCo-approved SDK is responsible for displaying the questions to the cardholder and obtaining the answers, using either a native UI, or an HTML UI implemented by a web view.

When a native UI is used, a malicious merchant can capture the answers by modifying the SDK so that it leaks the answers to the merchant, or by leaving the SDK as is but sometimes using code other than the SDK to conduct the challenge-response interaction. The specification has a formal requirement that a 3DS Server, provisioned by the same 3DS Integrator that supplies the approved SDK to the merchant, must verify that the SDK has not been modified; but as discussed above in connection with Misconception 1, this requirement is impossible to satisfy in the context of current mobile technology. The specification does not have a formal requirement that the approved SDK, and not some other code, is always used by the merchant app to conduct the challenge-response interaction.

When a web view is used, a malicious merchant can similarly obtain the answers using code other than the approved SDK to conduct the challenge-response interaction. It can also use the approved SDK to conduct the interaction, but exfiltrate the answers by running JavaScript code in the web view as discussed above in connection with Misconception 3.

Having captured the answers to the security questions, the malicious merchant can use them to impersonate the cardholder and transact with honest merchants using the cardholder's credit card data. If the same security questions are used by the bank for cardholder authentication as for locked account recovery, the merchant may also be able to use the captured answers to take control of the cardholder's account with the bank.

4.2 The merchant can capture one-time passwords and use them to impersonate the cardholder

3-D Secure 2 also envisions authenticating the cardholder by means of a one-time password (OTP) sent in a text message or an email message, as illustrated in Figures 4.10 and 4.14 of the Protocol Specification. A malicious merchant can capture an OTP in the same way as it can capture answers to security questions, and can use it to impersonate the cardholder as follows:

1. The merchant waits for the cardholder to initiate a transaction on the merchant's site using the merchant app.

2. The merchant app simulates the transaction without actually interacting with the issuing bank, using code other than the approved SDK. This requires knowing a few digits of the cardholder's phone number and a few characters of the cardholder's email address, to be able to ask the cardholder how the OTP should be sent as shown in Figure 4.10 of the Protocol Specification. To that purpose the merchant app uses digits and characters exfiltrated from an earlier *bona fide* transaction as described above in connection with the third misconception.

3. The merchant, purporting to be the cardholder, performs some action that requires authentication to the bank with an OTP. When the merchant is asked to choose how the OTP is to be sent, it makes the same choice made by the cardholder in the simulated transaction.

4. The action performed by the merchant causes the bank to send an OTP to the cardholder. The cardholder receives the OTP and enters it into the merchant app, believing that it has been sent for use in the simulated transaction.

5. The merchant app captures the OTP and sends it to the merchant, which uses it to complete the action where it is impersonating the cardholder.

6. The merchant app terminates the simulated transaction by displaying an error message to the cardholder stating that the transaction could not be completed.

One action that the merchant may attempt to perform purporting to be the cardholder is to login to the cardholder's account at the bank, guessing any additional information that may be required to log in besides the OTP. A usual two-factor authentication method consists of asking for username and password plus a OTP. Knowing that many web users reuse their passwords at multiple sites, the malicious merchant may entice customers to create an account at the merchant's web site with authentication by username and password, promising future loyalty benefits if they do so. If the cardholder has created such an account, the merchant may be able to log in to the cardholder's account with the bank using the same username and password that the cardholder uses with the merchant account in conjunction with the captured OTP.

Another action that the malicious merchant may attempt to perform is a transaction with an honest merchant using the cardholder's credit card data. The cardholder may become suspicious of the malicious merchant once he or she sees the fraudulent transaction with the honest merchant on the credit card bill with the same date as the terminated transaction with the malicious merchant, but the malicious merchant may have gone out of business by then and disappeared, after defrauding many honest merchants by impersonating many different cardholders.

4.3 The merchant can add itself to a whitelist of merchants trusted by the cardholder without cardholder consent

By the same methods by which the merchant app can read authentication information entered by the cardholder in the course of a challenge-response interaction, the merchant app can also modify information entered by the cardholder before it is sent to the bank. One piece of information entered by the cardholder, shown in Figure 4.14 of the Protocol Specification, is an answer to the question:

Would you like to add this Merchant to your whitelist?

There is no explanation in the specification as to how this whitelist is used, but clearly the merchant should not be able to add itself to it. However it can to just that by modifying the cardholder's answer to that effect before it is sent to the bank.

4.4 Fundamental nature of the flaws

It should be clear that the above flaws are not minor flaws that can be fixed by minor changes. A malicious merchant can capture information that the cardholder uses to authenticate to the bank by modifying the 3DS SDK, or by using alternative code to perform a challenge-response interaction, or by exfiltrating information from the UI used to perform the interaction. As discussed above, it is easy to exfiltrate information from a web view by using the `evaluateJavascript` method of the view to run JavaScript inside the view. Exfiltrating information may be more difficult when using a native UI to conduct the interaction, but should also be possible in that case, using methods dependent on the operating system of the cardholder's device and the programming language used by the merchant app. Authentication information cannot be kept away from the merchant if it is entered by the cardholder into the merchant app. The above flaws cannot be fixed without using a different approach to cardholder authentication in the configuration where the cardholder uses a native app to access the merchant's site.

5 Conclusion

3-D Secure 2 is an attempt to reduce the friction caused by the original version of 3-D Secure, by waiving cardholder authentication for transactions deemed to be low risk.

In previous work [6] we have shown that 3-D Secure 2 has serious privacy and usability issues and proposed a protocol that provides security for all transactions without friction by authenticating the cardholder cryptographically. Here we have shown that 3-D Secure 2 has fundamental security flaws in the configuration where the cardholder uses a native merchant app rather than a web browser to access the merchant's site.

The merchant is required to use an EMVCo-approved SDK to implement a challenge-response interaction by which the cardholder submits authentication information to the issuing bank. But a malicious merchant can capture the authentication information by circumventing this requirement or, in the case where the interaction takes place in a web view, by running JavaScript code in the web view to exfiltrate the information entered by the user. The specification envisions the use of a sequence of security questions or an OTP as authentication information. Capture of the security questions may allow the malicious merchant to take control of the cardholder's account with the bank if such questions are also used for locked account recovery. Capture of an OTP may allow the malicious merchant to conduct fraudulent transactions with honest merchants using the cardholder's credit card data, by timing the fraudulent transactions to coincide with attempts by the cardholder to conduct transactions with the malicious merchant. Capture of an OTP may also allow the malicious merchant to log in to the cardholder's account at the bank with authentication by username and password plus OTP, if the cardholder uses the same username and password to log in to the bank as to an account with the malicious merchant.

Such flaws cannot be fixed without using a different approach to authenticating the cardholder in the case where the cardholder uses a native app to access the merchant's site.

References

[1] Eric Rescorla. *SSL and TLS—Designing and Building Secure Systems.* Addison-Wesley, 2001.

[2] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. https://tools.ietf.org/html/rfc8446.

[3] SET Secure Electronic Transactions LLC. SETCo Main Page. Archived at https://web.archive.org/web/20020802134102/http://www.setco.org/.

[4] Visa. Visa 3-D Secure 1.0. https://technologypartner.visa.com/Library/3DSecure.aspx.

[5] EMVCo$^{\text{TM}}$. EMV® 3-D Secure.
https://www.emvco.com/emv-technologies/3d-secure/.

[6] Francisco Corella and Karen Lewison. Frictionless Web Payments with Cryptographic Cardholder Authentication. In C. Stephanidis, editor, *Proceedings of the 21st HCI International Conference, HCII 2019*, volume Late Breaking Papers, pages 468–483. Author's version available at https://pomcor.com/techreports/frictionless-cardholder-authentication.pdf.

[7] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The Emperor's New Security Indicators. In *IEEE Symposium on Security and Privacy*, pages 51–65, 2007.

[8] Robin Wood. Web App Mutual Authentication Fail. June 2017.
https://digi.ninja/blog/mutual_auth.php.

[9] Jim Youll. Fraud Vulnerabilities in SiteKey Security at Bank of America. July 2006. Archived at https://web.archive.org/web/20161231004055/http://cr-labs.com/publications/SiteKey-20060718.pdf.