ISSN:0975-9646

Lakshmanarao Battula et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (6) , 2015, 5334-5339

Secured Remote Client Authentication using Elliptic Curve Cryptography Algorithm

Lakshmanarao Battula[#1], Dr.P.Vamsikrishna raja[*2]

[1] *Asst.Professor,* [2] *Professor,*
[1,2]*Dept of Computer Science and Engineering, Kakinada Institute of Technology & Sciences.,*
Divili, Peddapuram (M), E.G.DT., AP,India

***Abstract*— the effectiveness of remote client-authentication schemes varies significantly in relation to today's security challenges, which include phishing, man-in-the-middle Attacks and malicious software. A survey of remote authentication methods shows how each measure up and includes recommendations for solution developers and consumers.**

INTRODUCTION

In today's world of distributed data sources and Web services, the need for remote authentication is ubiquitous. By "remote," we mean any infrastructure in which client and server are connected via some potentially insecure network—are it the Internet or a data connection using short-message service (SMS). Internet banking might be the prime example; with it comes the added challenge of a user base that's not necessarily technically savvy. Ease of use and high resilience against accidental misuse are thus of particular importance.

So far, researchers have proposed many remote authentication methods, including simple passwords, public-key infrastructures (PKIs), biometrics running on desktop PCs, smart cards, and mobile phones. Each has a reason to exist, depending on the design criteria for the overall usage scheme. The challenge therefore has become less one of inventing a working scheme, and more of deciding which scheme to choose given the design criteria. In this article, we assume that developers are addressing security as the foremost concern. Security can, however, conflict with business or Usability goals. It might be acceptable, for example, to deploy password-based authentication solutions when developers are more concerned with cost and minimal user training and support than with the threat of improper authentication.

Existing system

The security challenges is to use a security device with a display and embedded keypad that maintains a secure end to end connection with the server, by this it protects against various malicious software attacks.

Proposed System

The proposed system is provide a PKI based remote authentication scheme by this server usually stores either copies of the certificates or corresponding hash values that it can authenticate.

- Although still widely used, scratch lists are no longer state of the art as they can't withstand phishing and malicious software attacks.
- Challenge/response one-time codes or PKI-based schemes, combined with a secure device, should be the basis for any authentication solution.
- Because MITM attacks are increasing, developers should build solutions with a clear vision of how they might be extended to thwart MITM attacks.

Remote authentication schemes

Any remote authentication method's goal is to establish and secure an authenticated information channel by proving a user's identity through an associated security channel. For most methods, the information channel also serves as the security channel and—unless we state otherwise—we assume this is the case in our discussion. Terminology-wise, we also assume that it's always the client who connects to and authenticates with a server.

Static passwords

The oldest, most primitive remote authentication method is the use of static passwords, which typically change—at most—only every few months. With this method, a client presents a single static password to the server for each authentication; the server then matches it with the password stored for that client. Static passwords are still widely used in application domains where the environment is well controlled, the protected values are limited, or the potential risks are manageable. Example domains include authenticating a user locally with his or her personal computer (PC); remote authentication within local-area networks or intranets; or access control to an Internet bookstore. However, when it comes to highly sensitive data— such as information about financial institutions, their customers, and their transactions—researchers today deem static passwords an insufficient remote authentication method.1 we therefore exclude this approach from our discussion.

One-time codes

Remote authentication with one-time codes is based on the idea that both client and server share a secret. The client presents it to the server either as is (that is, the secret is the one-time code) or in a derived form according to some algorithm, possibly with additional data also known to the server. (An exception here is with systems based on one-

way hash functions—such as the S/KEY system.2 in these systems, rather than use a shared secret, the server authenticates the next code in a sequence based on the previous code.) In the One-time code approach, clients present each code to the server only once; codes can't be reused.

Scratch lists. A scratch list is the simplest form of a one-time code. A scratch list is typically given to the client once, in paper form, and usually contains about 40 to 100 codes. The server knows these codes, and clients use them sequentially or in an indexed form. So, the shared secret is the listed code and clients use it as is, without further derivation. If the client uses an indexed scratch list, the server decides which one-time code should be used next by specifying its index in the list; otherwise, clients typically have to track the used codes themselves. Either way, each code is used once and only once, and the server automatically sends the client a new list when only a certain number of codes are left. Figure 1a illustrates a scratch list scenario.

Short-time codes. With short-time codes, both client and server share one or more secrets exchanged in advance. They might, for example, use a symmetric key and derive one-time codes for authentication based on these shared secrets and the current time. They never actually exchange the shared secrets; just the codes derived via some derivation function $f(x)$ (see Figure 1b). Time granularity is typically on the order of a few Minutes—that is, the same code is derived during that time. This permits small time shifts between the client and server, and the server also usually accepts codes derived from times within the previous and next time slots.

Challenge/response codes. As Figure 1c shows, Challenge/response codes modify the short-time code concept by substituting a server-specified challenge for the current time. That is, client and server are again initially equipped with one or more shared secrets, such as a symmetric key and a counter value that's incremented after each authentication attempt. Then, for authentication, the server presents a randomly chosen challenge to the client. The client then responds with a code derived from the shared secrets and the challenge, while the server performs the same derivation; the counter value thereby prevents identical responses to the same challenge. Although there's no time-shift problem with this approach, the client can still inadvertently calculate response codes, potentially misaligning some secrets shared with the server. Equally, the server might be accidentally or intentionally triggered to send out challenges and calculate response codes, which again misaligns shared client-server secrets. Given this, the server typically doesn't accept one and only one response code; it also accepts codes neighboring the target code up to a certain limit (such as the five previous and following ones). If it detects a match, the server realigns its shared secrets accordingly. Alternatively, the server might send the challenge via a separate security channel that's authenticated by some other means (such as an SMS communication secured via the mobile phone network). Thus, the identity function can derive the response and no shared secrets are required. In this case, the resulting authentication's strength is inherited from the selected security channel.

PKI-based authentication

In contrast to one-time codes, authentication based on public-key cryptography doesn't rely on shared secrets. 3 Instead, each client is initially equipped with a private key (never to be exposed) and a matching public key. Furthermore, the server uses a PKI that issues a digital certificate to bind the client's identity to his or her public key. The certificate contains the client's public key, which is signed by one or more certificate Agency (CA) that the server trusts.

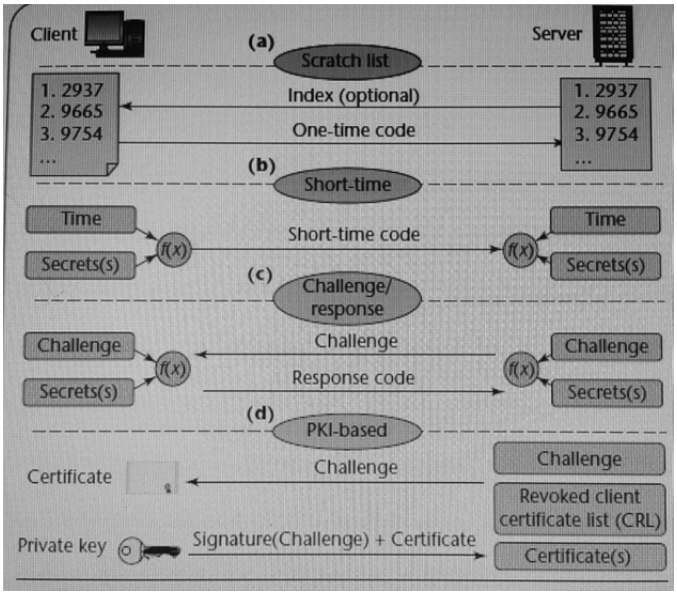

Figure 1. Remote authentication schemes. There are currently four primary approaches to advanced security: (a) a scratch list scenario, (b) short-time codes, (c) challenge/response codes, and (d) public-key infrastructure (PKI) authentication.

Although it's somewhat difficult to establish and maintain a PKI, the authentication itself is rather simple. The server presents a randomly chosen challenge and the client signs with its private key. (If both parties fail to use necessary safeguards to prevent well-known crypto-analytic attacks, such as the chosen-plaintext attack, however, then the authentication scheme can be broken.) As Figure 1d shows, both the signed challenge and the client's certificate are then returned to the server in response. The server thereupon ensures that: the client's certificate is valid (that is, it's signed by a trusted CA and the signature verifies), and the signature of the challenge verifies with the given client certificate. The server also maintains a list of revoked client certificates (CRLs) in case, for example, a client's private key is compromised and must be invalidated. During authentication, the server checks each client-presented certificate against the CRL and if it finds a match,

it denies the authentication. For a positive test, or if more than one server uses the same CA without necessarily authenticating the same group of clients, the server usually stores either copies of the certificates or corresponding hash values that it can authenticate. Furthermore, each certificate's lifetime is usually limited from a couple of months to a couple of years, after which the server issues a replacement certificate to the client.

ECDSA - Elliptic Curve Digital Signature Algorithm

Signature Generation

For signing a message m by sender A, using A's private key dA

1. Calculate e = HASH (m), where HASH is a cryptographic hash function, such as SHA-1
2. Select a random integer k from [1,n − 1]
3. Calculate r = x1 (mod n), where (x1, y1) = k * G. If r = 0, go to step 2
4. Calculate s = k − 1(e + dAr)(mod n). If s = 0, go to step 2
5. The signature is the pair (r, s)

Signature Verification

For B to authenticate A's signature, B must have A's public key QA

1. Verify that r and s are integers in [1,n − 1]. If not, the signature is invalid
2. Calculate e = HASH (m), where HASH is the same function used in the signature generation
3. Calculate w = s −1 (mod n)
4. Calculate u1 = ew (mod n) and u2 = rw (mod n)
5. Calculate (x1, y1) = u1G + u2QA
6. The signature is valid if x1 = r(mod n), invalid otherwise

Biometrics

Biometrics base remote authentication on "being something" instead of "knowing something" (such as a onetime code) or "having something" (such as a private key). In biometrics, the server matches one or more client biometric—such as a fingerprint, facial feature, iris pattern, or hand geometry—with the same information previously stored at the server during enrollment. During the enrollment process, the serving organization captures and stores each client's biometric information under well-controlled conditions. Then, for authentication, the client again captures that information and sends it and the claimed identity to the server for matching. The servers can thus reliably authentic clients given three assumptions. That is, that biometric information.

- can be reproducibly captured repeatedly,
- cannot be easily faked, and
- is sufficiently different between any two clients.

Unfortunately, clients can't capture biometric information reproducibly, but rather only in close approximation. A biometric match never returns a clear-cut yes or no result—it returns only a probability as to the verifiability of the client's identity. This coercively causes so-called "false rejects" (genuine clients aren't authenticated) and "false accepts" (im-poster are authenticated). Although developers can move a threshold to adjust the tendency as to which side a method will err on and with what probability, each biometric authentication is inherently prone to this type of misjudgment.

A significant drawback here is that it's possible to obtain some biometric properties, such as after physical contact (fingerprint on a water glass, for example) or by using a high-resolution picture of a person's eyes. This limits biometrics' value in scenarios where forgery of this type—say, plastic fingers or photographs presented during the authentication phase—are undetectab le. Also, like static passwords, clients can use biometric features repeatedly once they're obtained.

To prevent such misuse, we'd have to authenticate the biometric device used to capture the biometrics to ensure the authentication data's origin.4 This would introduce a whole new complexity level, making biometrics of limited value for remote authentication over insecure networks.

Client security devices

The most common client hardware is a standard desktopPC, which is also an easy platform to attack. People thus often additionally use a security device, such as a smart card (either in a standalone reader or one that connects to the PC), a mobile phone or PDA, or a smart memory stick. The security device's software then (at least partially) performs the authentication, preventing certain types of attacks against the PC. As we now describe, there are currently several commonly used client-security devices. Although this might make the security channel distinct from the information channel, we assume that the information channel is always Established between a client's PC and the server.

Smart cards

A smart card consists of a plastic card with a small embedded Microprocessor with various memory types (such as ROM, RAM, and Eeprom) and tamper- resistant properties (such as secure crypto-coprocessors for symmetric and public-key cryptography). Typically, smart cards have external power and clock, and communication is serial (contact-based or contactless). To communicate with the smart card, users need a reader that connects it with either a PC or standalone device. Standalone readers typically have at least a small display and a numeric keypad where users enter their PIN and commands. Readers providing a PC connection are commonly classified according to their capabilities, with class 1 readers simply providing connection, class 2 adding a pin pad, and class 3 readers offering a display and some programming capabilities.

Class 4 readers feature a separate security module and a virtual machine for custom application execution; we thus consider them secure execution platforms.

Given their resistance to tampering, smart cards are generally accepted as sufficiently secure to store sensitive data, particularly private keys. Ideally, private keys are generated on the smart card and are never directly exposed to the outside world.

Mobile phones and PDAs

Mobile phones or personal digital assistants (PDAs) are separate computing platforms with their own displays and keypads. Both can serve either as standalone devices (for storing and computing one-time codes, for example) or as a PC connection using Bluetooth, infrared, or USB to remotely authenticate users. Mobile phones also can use their mobile networks as security channels. Functionality-wise, mobile phones are increasingly general-purpose computing platforms that support more or less open software execution platforms. By far the most prevalent code execution platform is Sun's Java, which is used in millions of phones. Downloading and installing Java applications is simple; vendors have a huge commercial interest in making any software purchase (such as games) easy for mobile phone users. All versions of J2ME—the stripped-down "embedded" Java version run in mobile phones—lack desktop and enterprise Java security features such as the byte-code verifier, which performs static code and integrity checks.5 Without this on-device verifier, attackers can write "subversive" code and thereby access the data of other Java code—such as a banking application— residing on the same mobile phone. Only the most recent J2ME version (MIDP 2.0) includes the possibility of digitally signing code and thus tying code executed on the device to a trusted source that presumably performs proper off-device byte-code verification. Only devices that conform to this specification level meet the recommendation to only run code of known origin. It's highly dubious, however, that we'll be able to educate typical users enough to verify that a piece of code's digital signature refers to a trusted source. The underlying issue of checking certificate origin—that is, checking SSL certificates to avoid man-in-the-middle (MITM) attacks—has already proven intractable in PC-based online banking. Java's prevalence, combined with a nonexistent or difficult-to-verify code-origin check and a weak code security model, makes mobile phones far weaker than PCs when it comes to security. Storing secret information (such as private keys) to support banking applications on a mobile phone that lacks a security module—that is, one that uses *soft tokens,* or software only authentication implementations—is, in our opinion, far too dangerous and users should not even

Consider it. Instead, they should use a smart card as a tamper-resistant security module. Nearly all mobile phones have security interface modules (SIM), which are smart cards that, among other things, authenticate the mobile phone with the mobile network provider. Network operators can deploy additional applications on these SIMs, which can use the mobile phone network as a security channel. The new generation of mobile phones also has a second smart card that offers secure storage for applications running on the phone's runtime platform. The card also provides a runtime platform for secure applications. These new phones can connect smart cards to PCs using near-field communication (NFC),6 a short-range contactless communication interface and protocol based on ISO proximity card standards.7 Although it's convenient for users, such direct communication between NFC-enabled mobile phones and PCs requires them to add a contactless reader device to their PCs. To use the second smart card simply, that is, as a phone-based authentication with user interaction on the phone, is similar to using the SIM. The only difference is the chip's availability (which is rare compared to SIM) and accessibility (which is better than SIM). Still, short of the phone-internal chip that drives all external user interactions—through a SIM Application Toolkit (SAT)8 application, for example—using a security device helps protect only the secret data. It can't guarantee overall application integrity. If a user can be tricked into providing the password to the secret data, a malicious on-phone application can freely access the data, even if it's safely stored. Any application that uses a mobile phone as a security device must account for this fact when designing what on phone code can do with the secret information once it's unlocked. As an obvious example, it should be impossible for an application to copy out this data, even if the user presents the correct PIN code.

Smart memory sticks

Memory sticks equipped with a smart card—in, for example, a USB form factor for use with PCs—are a rather new development. The memory stick mounts as a write-protected volume (like a CD) and usually contains some immutable software for remote authentication. This software might be a Web browser, such as Firefox, that's restricted to connecting only to certain Web sites. For added convenience, users can Configure some or all of this software to automatically Launch whenever the memory stick is mounted. Any mutable state and any data that can't be exposed (such as a shared secret or a private key) are stored and processed on the embedded smart card. One of this technology's main challenges is how to create user-friendly updating of the read-only memory if, for example, security patches require installation of a new version of the memory stick's software. Furthermore, not all applications can operate from a read-only Device. As a result, they must be temporarily copied to the PC's hard drive prior to each execution. In principle, smart memory sticks can work with mobile phones—by inserting them into the secure digital (SD) slot, for example—as of now, however, we're not aware of any such products.

Attacks and countermeasures

All of the methods mentioned earlier authenticate a client with a server and are thus equivalent in terms of functionality. However, their power to resist attacks differs significantly. We must therefore understand the potential attacks on a remote authentication method before choosing one. Here, we focus on three types of attack: phishing, malicious software, and MITM. All three are attacks against the client, whose protection mechanisms are arguably less sophisticated than those typically found at a server. We don't discuss attacks Against the server, such as denial of service. Figure 2 offers an intentionally generic outline of the three attack types; we don't consider combination attacks or further relationships between the attacker and the client

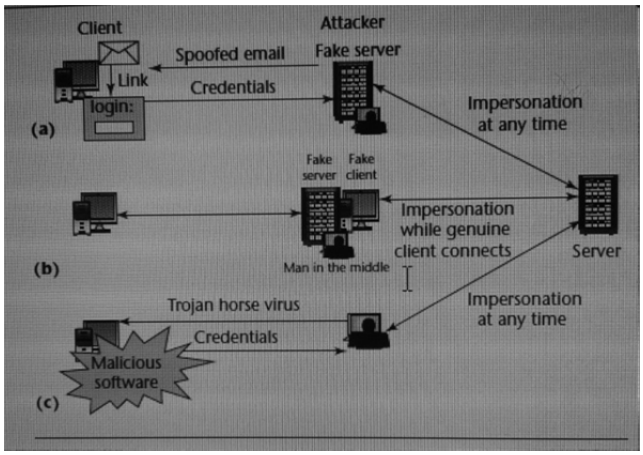

Figure 2. Three types of attack scenarios. (a) Phishing attacks trick users into revealing their credentials. (b) Man-in-the-middle attacks intercept communication between client and server to modify transactions or hijack the authenticated channel. (c) Malicious software invades PCs to fraudulently gather user credentials.

Phishing

As Figure 2a shows, phishing combines spoofed emails and mocked-up Web pages. An attacker, might, for example, hijack a well-known financial institution's trusted brand and trick users into entering their authentication credentials (such as a one-time code) into a faked Web form. Such trickery is commonly achieved through emails that look genuine if users only casually examine them; most users don't actually know how to reliably identify a genuine server. Phishes reportedly convince up to 5 percent of users receiving a spoofed email to respond and reveal their secrets.9 Scratch lists, even indexed ones, are inherently prone to phishing attacks because the one-time code's validity period is itself rather long (or even unlimited). Also, the code isn't specifically connected with a particular information channel. So, when attackers get a one-time code from a scratch list, they can use it with any information channel to the genuine server any time after an attack and prior to the legitimate user's next attempt to authenticate with that server. Biometric authentication is also conceptually vulnerable to phishing attacks for the same reasons. To make it more difficult for phishes to obtain secret Information stored on a mobile phone's security module, it might help to authenticate not just the user, but also the backend system to the secret data container. Also, the secret data container is unaware of the current time. By reliably provisioning it with the current time, we might prevent repeated/recurrent use of the secret data without the container's knowledge, and also possibly have it shut down if it becomes aware of an attack. In contrast, short-time codes at least limit an attacker's window of opportunity to a couple of minutes. Still, only challenge/response authentication effectively prevents phishing by strictly associating each response to a specific authentication attempt. Applying this line of reasoning, PKI-based authentication methods also prevent phishing attacks. In fact, we can consider the server challenge's digital signature as a response, much like a one-time code challenge/ response scheme, though the latter usually employ a simpler infrastructure than PKI.

Man in the middle

The infamous MITM is a network attack. Rather than trying to obtain a user's authentication credentials, the attacker covertly intercepts messages between the client and server, masquerading as the server to the client and as the client to server, respectively (see Figure 2b). Although virtually all of today's servers are authenticated via a public-key certificate when users establish an SSL/TLS session, users often naively ignore warning messages about invalid or untrusted certificates. This lets attackers hijack an authenticated information channel or silently modify transaction data. In contrast to phishing, however, an MITM attack doesn't necessarily compromise a user's credentials.

On the protocol level, the SSL/TLS protocol's client-authentication option can render MITM attacks impossible. Unfortunately, the SSL/TLS protocol doesn't support one-time code schemes for client authentication (although researchers have made a proposal in that direction10). On the application level, MITM attacks can be prevented only by challenge/ response one-time codes or PKI-based authentication methods—if both are extended to this end. To exclude an MITM, the client and server must uniquely Identify the information channel, and then

- use this identification as an additional input parameter when calculating the one-time code response, or
- Concatenate it with the data to be digitally signed.

The client and server could, for example, use the session-specific SSL/TLS protocol information—such as the handshake message's hash value—to identify the information channel. Such information would be different for client and server if, instead of one end to- end session, they had two sessions with an MITM. Consequently, either the client's one-time code response or the signed data wouldn't match, respectively. If the session identification is independent of the SSL/TLS connection—as in the use of cookies, for example—the session has no MITM resistance.11 all other remote authentication methods we discuss here are prone to MITM attacks. Unfortunately, recent research shows that while online services are becoming more resistant to phishing attacks—such as by moving from scratch lists to short-time password-generating hardware tokens—MITM attacks are increasing.12

Malicious software

Malicious software aims to fraudulently gather authentication Credentials by invading an insufficiently protected client PC by means of a virus or a Trojan horse (see Figure 2c). For example, once established, a Trojan horse could read and forward a private key stored on the PC's hard drive while monitoring keyboard activity to access the pass phrase used to decrypt the private key. Users can protect themselves against malicious software using security precautions—such as installing and maintaining a firewall and regularly updating antivirus software; applying OS and browser patches as needed; and configuring software appropriately—but few users strictly adhere to such procedures.

Considering most users' lack of attention to securing their PCs, server providers increasingly classify PCs as a

generally insecure client platform and refrain from using any soft tokens. This is why smart cards play an increasingly important security role: they not only store a client's credentials, but also perform any necessary computations.

A first step is a smart card directly connected to a PC by a class 1 reader. When the smart card runs a scheme to generate challenge/response one-time codes or a PKI-based authentication method, a client's credentials are no longer plainly available to malicious software. Even if a virus or Trojan horse obtains thesmart card's PIN via a keyboard logger, it can still trigger only the client PC's authentication method, rather than grab the credentials and use them on any PC.

Developers can achieve higher security by connecting the smart card to the PC using a class 3 or 4 reader, or a Bluetooth/NFC-connected mobile phone. This prevents malicious software from opening the smart card by itself and restricts it to triggering the authentication method once the user has entered the PIN. If each authentication method invocation requires some further user interaction on the reader, it becomes even harder because users must be tricked into explicitly accepting a remote authentication. Similarly, developers could use a reader to display application-related Information and authorize transactions.13

If both challenge and response can be easily transferred manually from one device to another, users can get a standalone device that totally eliminates malicious software attacks. Such a device can generate short-time codes, for example, and show them on a small display so the reader can view the code and enter it into a server Web form. If the standalone device has a keypad, it can generate challenge/response one-time codes and users can manually transfer both challenge and response between the device and their PCs. A keypad also lets users protect the device itself using a PIN, which renders it useless if lost. If the standalone device—such as a mobile phone—has network connectivity and can receive short messages, users can leverage this into a separate security channel and have the server send individual one-time authentication codes on demand. However, in such cases, the security channel's properties should be thoroughly evaluated. For example, with SMS communication, users must consider that messages can be delayed or lost, network operators can trace them, and a phone without PIN protection might be stolen.

In contrast to the one-time code approach, PKIbased authentication methods are slightly more difficult to implement on a standalone device. Manually transferring a digital signature from the device to a Web form is anything but practical. When combined with a mobile phone, however, users can separate the security and information channels. That is, the mobile phone network first sends a challenge to a client's mobile phone running the authentication software via SMS, for example. Next, users enter some authorization code into the mobile phone. The server might, for example, send that authorization code to the client via the information channel, thereby connecting it with the digital signature.14 finally, the PKI-enabled mobile phone sends the digital signature back to the server using the mobile phone network.

CONCLUSION

- Although still widely used, scratch lists are no longer State of the art as they can't withstand phishing and malicious software attacks.
- Challenge/response one-time codes or PKI-based Schemes, combined with a secure device, should be the basis for any authentication solution.
- Because MITM attacks are increasing,12 developers Should build solutions with a clear vision of how They might be extended to thwart MITM attacks.

REFERENCES

1. B. Schneier, "Two-Factor Authentication: Too Little, Too Late," *Comm. ACM*, vol. 48, no. 4, 2005, p. 136.
2. L. Lamport, "Password Authentication with Insecure Communication," *Comm. ACM*, vol. 24, no. 11, 1981, pp. 770–772.
3. R.E. Smith, *Authentication: From Passwords to Public Keys*, Addison-Wesley, 2002.
4. U. Waldmann et al., "Protected Transmission of Biometric User Authentication Data for Oncard- Matching," .*Proc. ACM Symp. Applied Computing*, ACM Press, 2004, pp. 425–430.
5. X. Leroy, "Java Bytecode Verification: Algorithms and Formalizations," *J. Automated Reasoning*, vol. 30, nos. 3-4, 2003, pp. 235–269.
6. *The Keys to Truly Interoperable Communications*, Near Field Communication Forum, 2007; www.nfc-forum. Org/resources/white_papers/nfc_forum_marketing _white_paper.pdf.
7. *Proximity Integrated Circuit Cards (PICCs), ISO/IEC 14443, parts 1-4*, http://wg8.de/sd1.html#14443.
8. *Specification of the SIM Application Toolkit (SAT)*, *3GPP Standard TS 11.14 v. 8.5.0*, www.3gpp.org/ftp/Specs/ archive/11_series/11.14/1114-850.zip.
9. *Report on Phishing*, Binational Working Group on Cross-Border Mass Marketing Fraud, US Department of Justice & Ministry on PublicSafety,Oct. 2006, www.usdoj.gov/opa/report_on_phishing.pdf.
10. M. Steiner et al, "Secure Password-Based Cipher Suite for TLS," *ACM Trans.*, vol. 4, no. 2, 2001, pp. 134–157.
11. K. Fu et al., "Dos and Don'ts of Client Authentication on the Web," *Proc. Usenix Security Forum*, Usenix Assoc., 2001, pp. 251–268.
12. *Semi-Annual Report*, Federal Office of Police, Swiss Reporting and Analysis Centre for Information Assurance (MELANI), 2007; www.melani.admin.ch/ dokumentation/00123/00124/01029/index.html? lang=en.
13. T. Weigold et al., "The Zurich Trusted Information Channel—An Efficient Defence against Man-in-the- Middle and Malicious Software Attacks," P. Lipp, A.R. Sadeghi, and K.M. Koch, eds., *Proc. Trust Conf.* (Trust 2008), LNCS 4968, Springer-Verlag, 2008, pp. 75–91.
14. The WPKI Non-Profit Association, *WPKI Main Specification*, v. 2.0, March 2006; www.wpki.net/files/WPKI%20Main%20Specification%202.0.pdf.
15. R. Thompson, "Why Spyware Poses Multiple Threats to Security," *Comm. ACM*, vol. 48, no. 8, 2005, pp. 41–43.
16. *US-Cert: Quarterly Trends and Analysis Report*, vol. 2, no. 2, U.S. Computer Emergency Readiness Team, June 2007; www.us-cert.gov/press_room/trendsandanalysis Q207.pdf
17. *Korea Phishing Activity Trends Report*, Korea Internet Security Center, Mar. 2007; www.krcert.or.kr/english _www/publication/8_1_publication_list.jsp?board Type=PUB.
18. R. Dhamija et al., "Why Phishing Works," *Proc. Conf. Human Factors in Computing Systems* (CHI), ACM Press, 2006, pp. 581–590.
19. A. Hiltgen et al., "Secure Internet Banking Authentication," *IEEE Security & Privacy*, vol. 4, no. 2, 2006, pp. 21–29.
20. F. Puente et al., "Improving Online Banking Security with Hardware Devices," *Proc. 39th Int'l Carnahan Conf. on Security Technology* (CCST), IEEE Press, p. 174–177.