Secure Smart Card Signing with Time-based Digital Signature

Hossein Rezaeighaleh*, Roy Laurens**, Cliff C. Zou***
Department of Computer Science
University of Central Florida
FL, USA
rezaei@knights.ucf.edu*, rlaurens@knights.ucf.edu**, czou@cs.ucf.edu***

Abstract— **People use their personal computers, laptops, tablets and smart phones to digitally sign documents in company's websites and other online electronic applications, and one of the main cybersecurity challenges in this process is trusted digital signature. While the majority of systems use password-based authentication to secure electronic signature, some more critical systems use USB token and smart card to prevent identity theft and implement the trusted digital signing process. Even though smart card provides stronger security, any weakness in the terminal itself can compromise the security of smart card. In this paper, we investigate current smart card digital signature, and illustrate well-known basic vulnerabilities of smart card terminal with the real implementation of two possible attacks including PIN sniffing and message alteration just before signing. As we focus on second attack in this paper, we propose a novel mechanism using time-based digital signing by smart card to defend against message alteration attack. Our prototype implementation and performance analysis illustrate that our proposed mechanism is feasible and provides stronger security. Our method uses popular timestamping protocol packets and does not require any new key distribution and certificate issuance.**

Index Terms— **Time-based Digital Signature, Java Card, Secure Time Stamp, Terminal Attack, DER Decoder.**

I. Introduction

Online documents and applications became common and one of the main security challenges is trusted electronic or digital signing. In the majority of cases, a user should identify himself and prove that he is really a human and signs the document. On the other hand, attackers will try to fool a user into signing a rogue document. Currently, the main countermeasure mechanism is authentication by username and password and many popular services like DocuSign use that and trust the user's terminal. Even though user authentication is necessary, anybody besides the actual user can impersonate the real user and sign the documents. Therefore, in a more mission-critical application, each user has a unique asymmetric key pair (public and private keys) to digitally sign a document. Usually, the user must have a security device such as personal smart card which stores her private key and she can access it by entering her PIN. When the user approves content of a document, her terminal computes the hash of the document and sends it to her smart card. Then the user enters her PIN and the smart card generates secure digital signature using her private key.

At first glance, with this mechanism, the system can be sure that only the owner of the smart card can sign the document using the unique user's private key, but that is not always the case. The main security challenge of a smart card comes from its non-direct user input/output, as a smart card does not have monitor/keyboard and a user must rely on her terminal to perform input/output operations. Since terminals such as personal computers, laptops, tablets etc. are not secure at all, an attacker can compromise and breach these terminals to mislead a user's smart card operations. For example, the attacker can replace smart card driver library in a terminal to send an arbitrary content to smart card for digital signing. Thus, in a normal digital signature process, the user enters her PIN to digitally sign her approved content, but the attacker's fake driver sends another content to the smart card and the smart card cannot detect the attack because it relies on the terminal. This attack is applicable in all smart cards that are used to digital sign documents on untrusted terminals.

In this paper, we start by investigating and implementing two well-known attacks to smart card terminals. Then, we will present a novel protocol to secure digital signing on smart card with untrusted terminal. Our contributions in this paper are as follows:

- Proposing time-based digital signature with smart card
- Moving entire digital signature process from untrusted terminal to trusted smart card
- Negating the need to use new packet format and keys by utilizing existing keys, certificates and timestamping protocol
- Designing a new technique which we call onetime-scanning for DER (Distinguished Encoding Rules) decoding which significantly reduces processing time on smart card
- Developing a fully-functional prototype using Java card, and putting the code and library online as an open source project.

II. Smart Card Terminal Vulnerabilities

A. Threat Model

Several authors[1] proposed a threat model for smart card from abstract view, but in real world we can trust smart card chips and firmware because of their high security standards [2].

In addition, card issuer, key manager and Certificate Authority are trusted parties and use secure methods to issue smart cards. Therefore, we assume that the smart card has been manufactured, issued and delivered to user securely, and the main security challenges occur during usage.

There are several entities involved during smart card usage, including service provider (server), terminal (client) and card reader. We assume that card reader is trusted, and service provider is more secure than terminal because of common network security mechanisms such as firewall, IDS, Antivirus and so on, which are managed by professional administrators. On the other hand, the terminal is usually a general-purpose computer or mobile device that an attacker can compromise and alter. Consequently, the least secure part of the system is the terminal and in the rest of this paper, we assume that the terminal is not secure at all.

We summarize our assumptions of the threat model as follows:

- The following entities in smart card signing process are trusted: smart card manufacturer, smart card issuer, smart card chip, service provider, and user.
- Terminal is not trusted, i.e., an attacker can compromise and alter the terminal by installing malware.

B. Two Well-Known Attacks

In this section, we explain two possible well-known attacks to smart card terminal, including sniffing smart card's access PIN and altering digital signature just before signing.

1) Sniffing smart card PIN

A smart card should receive its user's password (PIN) to gain access to keys on the card. The main security challenge is that a smart card doesn't have direct input device and must use the terminal's keyboard, mouse etc. to get the PIN from its user. In this situation, an attacker can compromise terminal and install a key logger or another malware to capture the user's PIN. Then, the attacker can use this PIN to authenticate himself to the card without the user's authorization.

2) Altering user's digital signature

Another prevalent usage of smart card is digital signing. A user, such as government employee, company staff, individual etc. may use her smart card to sign emails, PDF files and other digital documents. The regular digital signature mechanism is as follows: a user sees the content of document in an online application and if she approves it, she signs it using her smart card. In this process, a cryptographic library, as part of the application or part of the operating system, computes the hash of the document and sends this hash value to smart card for signing. The challenge is that a malware can change the hash value just before transmitting it to the smart card, resulting in the user signing an unwanted content with her private key.

C. Implementation of Smart Card Attacks

We implemented the mentioned attacks on Windows, but they are applicable on other operating systems, too. The attack code is called MinidriverSpy. We used Personal Identity Verification (PIV) card [3] in our test. PIV is a smart card standard which is supported with built-in drivers from Windows 7 SP1, from OpenSC 0.11.1 (in Linux), and from Mac OS Sierra 10.12.

Microsoft Windows uses a software stack to communicate with smart card and conduct cryptography operations, and its important module is minidriver [4]. Windows has a built-in minidriver for PIV smart card which is MSCLMD.DLL. We implemented a spyware "MinidriverSpy" as a hooking DLL and replaced MSCLMD.DLL. Right side diagram in Figure 1 shows our change on attack. The only permission we need to do this action is file copy permission.

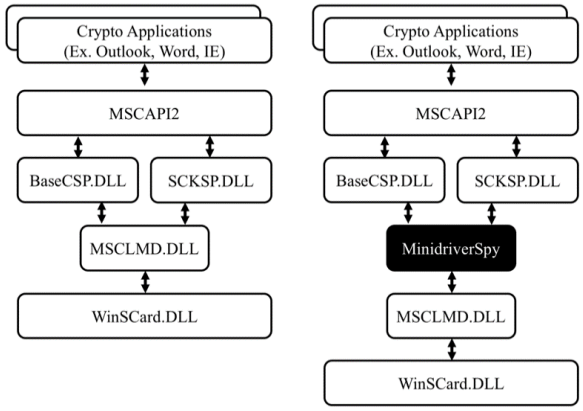


Fig. 1. Original Windows smart card software stack (left) and modified software stack after an attacker installs MinidriverSpy (right)

Original minidriver (MSCLMD.DLL) has only one entry point "CardAcquireContext". This function returns a set of function pointers of smart card minidriver. We added "CardAcquireContext" in our MinidriverSpy and pass these pointers from original minidriver to the caller, with some changes to implement our attacks. To sniff the smart card PIN, MinidriverSpy alters pointer of "CardAuthenticateEx" function and copy this PIN value before sending it to the original minidriver. To alter digital signature, MinidriverSpy modifies pointer of "CardSignData" function to change hash value just before sending it to smart card, and with this attacking tool, a user will be tricked to sign a fake data using her private key on smart card.

To test our attacks, we used two types of smart card including embedded smart card in USB token and traditional ID-1 sized smart card (credit card size) with USB card reader. We tested our MinidriverSpy successfully on YubiKey 4, PIVKey T600 USB Tokens and PIVKey C910 PKI Smart Card on Windows 7 Service Pack 1 64-bit and Windows 10 64-bit. We published basic parts of our MinidriverSpy as open-source software at GitHub [5].

In the rest of this paper, we focus on altering digital signature attack. For prevention of PIN sniffing attack, there are practical solutions in the market such as card readers with numerical pad [6] and smart card with embedded fingerprint scanner and match-on-card capability [7]. These methods prevent PIN sniffing but do not prevent the digital signature alteration attack.

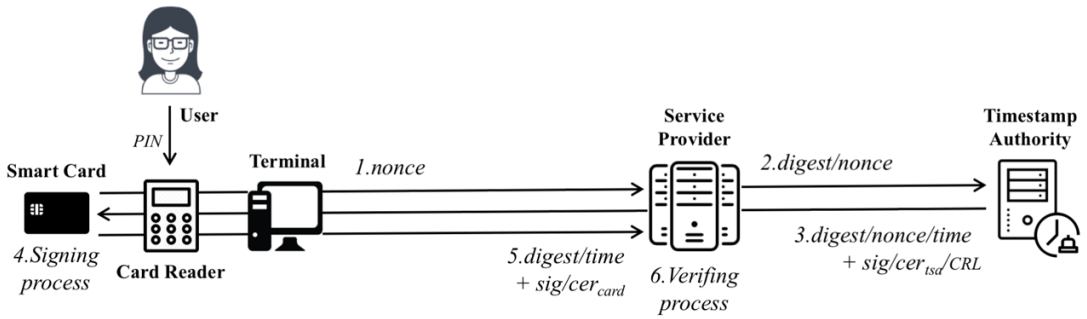


Fig. 2. Our proposed time-based digital signature using smart card

III. Time-based digital signature protocol

We propose a mechanism to prevent altering digital signature just before sending message to the smart card to sign. As explained in previous section, we assume that the smart card and the service provider are trusted and the terminal is untrusted. To deal with mentioned security challenge in the smart card digital signature process, we move all critical steps from terminal to smart card and transform the terminal into a simple transmitter and receiver. To do that, we add a trusted authority, and the smart card doesn't sign any digest unless this trusted authority has signed it beforehand. In other words, at first, trusted authority "pre-signs" the digest, then the smart card signs this digest if and only if it can verify the digest's signature. Therefore, an attacker cannot make changes in the digest because of trusted authority's signature, and any alteration will be detected by the smart card. This pre-signing mechanism follows the principle that a smart card signs a message if and only if a trusted authority has signed it before.

Even though pre-signing guarantees the digest's integrity, signature verification is not a simple action on a smart card. To verify a signature, a smart card should decrypt the signature and compare it with the digest. Furthermore, the smart card must verify the digital signature of the signer's certificate with CA's public key and check that the certificate is currently valid and has not expired. Another important step is to check certificate revocation by Certificate Revocation List (CRL) [8], which entails checking the CRL's digital signature and its time validity too. The big challenge in performing these steps on smart card is that checking certificate's and CRL's validity requires a trusted time but a smart card does not have any internal clock.

To solve time problem, we propose using Timestamp Authority (TSA). A TSA [9] receives a digest, then adds time to it and signs digest and time together. Therefore, when a smart card receives a timestamp packet, it verifies signature and can trust the time extracted from the timestamp packet. The smart card verifies the TSA's certificate by internal trusted CA's public key so that an attacker cannot use a fake TSA. Meanwhile, because an attacker can use an old or expired TSA, we propose that a smart card should insert received time in its response packet and sign digest and time together. A service provider accepts or rejects a packet which is signed by smart card by checking its time. Furthermore, before starting the signing process, a smart card generates a random nonce to guarantee packet freshness, and after receiving signing request, it verifies this nonce before signing.

Additionally, smart card, service provider and timestamp authority store a common Certificate Authority's certificate which has issued TSA's and smart card's certificate and CRL. The final process is illustrated in Figure 2 and is as follows:

Protocol 1: Time-based digital signing by smart card:
Input: Document that service provider requests user to sign
Output: Digital signature generated by user's smart card

1. Service provider gets a nonce from smart card.
2. Service provider sends the document's digest and nonce to TSA.
3. TSA signs digest, nonce and time and combines them with TSA's signature, TSA's certificate and up-to-date CRL. Then TSA sends the resultant packet to the smart card via service provider.
4. Smart card executes the following signing process:
    a. Verifies PIN
    b. Checks nonce
    c. Verifies signature of packet
    d. Extracts time from packet
    e. Verifies TSA's certificate by CA's public key
    f. Checks TSA's certificate validity time
    g. Verifies CRL's signature by CA's public key
    h. Checks CRL validity time
    i. Checks TSA's certificate with CRL
    j. Signs digest and time
5. Smart card sends the response packet to service provider.
6. Service provider executes the following verifying process:
    a. Verifies signature of packet
    b. Verifies smart card's certificate
    c. Verifies signature's time with TSA's time

In this mechanism, terminal doesn't do anything except sending and receiving packets, and if an attacker changes the contents of a packet, the smart card and the service provider can detect it and will not accept that packet.

IV. Security Analysis

To analyze the security of our proposed mechanism, we assume that timestamp authority, service provider and smart card are trusted. In addition, the actual user is authenticated to the service provider and wants to sign a document. On the other

hand, the terminal is untrusted and an attacker can install malicious programs (such as rogue driver), and intends to digitally sign an arbitrary document using the real user's private key which is stored on a smart card. In addition, we assume that the user uses a secure way to enter her PIN number such as PIN pad smart card reader.

Message alteration attack: As we mentioned before, under our proposed protocol, an attacker cannot mutate the document digest before sending it to a smart card, because the TSA has pre-signed it and the smart card verifies TSA's digital signature before signing, and hence, can detect any document modification by the attacker.

Fake TSA attack: If an attacker attempts to build a fake TSA to sign his rogue digest, the smart card will discard signing request because the smart card verifies TSA's certificate and the fake TSA's certificate is not issued by the trusted CA, whose public key is stored on the smart card.

Compromised TSA: Considering the case where a TSA server has been compromised and CA has revoked certificate of the TSA and put its serial number in CRL. The attacker who hacked this TSA server can pre-sign a request and send it to a smart card. To bypass revocation checking, he can also provide an old CRL. In this case, the smart card signs the document with provided time and, in verification phase, the service provider determines that TSA was revoked at that time.

Expired TSA: If the attacker uses an old expired TSA to pre-sign request, the smart card will detect it by checking time validity of TSA's certificate. Additionally, if the attacker sends a past time to the smart card, the smart card signs the document with that time and service provider will detect that the timestamp is not the current time and discard digital signature.

Replay attack: Attacker can conduct replay attack by sending pre-signed packet which is generated for one smart card to another smart card; since this pre-signed packet has TSA's signature, the second smart card can possibly accept it. The use of nonce in our protocol will prevent this replay attack. In the first step, the TSA must receive a nonce from the smart card and sign digest, time and nonce together, and when the smart card receives a packet, it checks the embedded nonce before signing, and after that resets its internal nonce value.

In order to implement the aforementioned countermeasure, the verification of the given TSA's certificate requires the current time on the smart card. A straightforward solution is to leverage the embedded clock in the smart card chips to provide current time, but the internal clock has not been traditionally built in these devices because they lack internal power. Considering the existing challenges, the current time of the TSA which has been inserted in the received packet will be considered as current time, and the smart card inserts this time to its signature packet to inform the service provider about current time which the smart card has used. Therefore, the service provider decides to accept or reject the smart card's digital signature based on whether this time is current or not.

V. IMPLEMENTATION

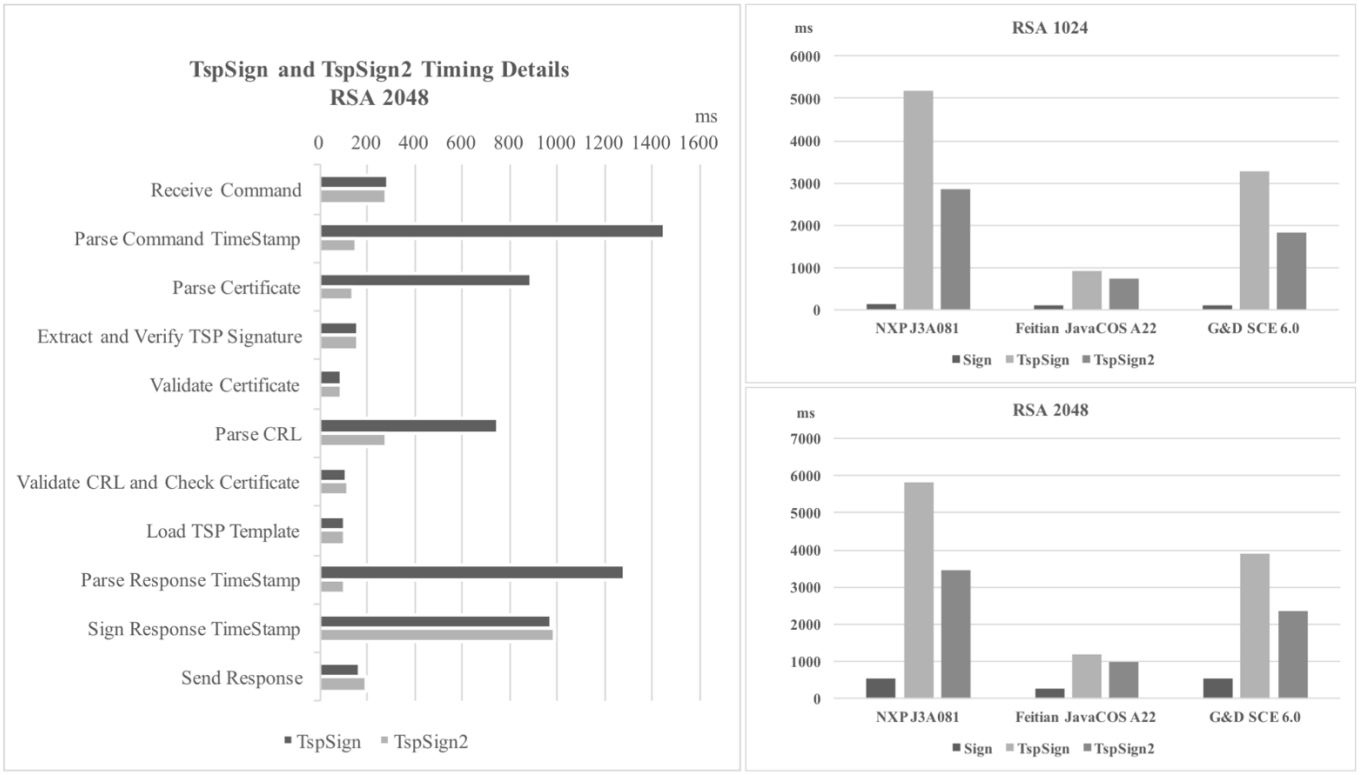
To implement our proposed mechanism, we employ existing timestamp protocol [9]. A timestamp token includes message digest, nonce, time and signature with attached signer certificate and CRL. Although we use timestamp token as defined in its protocol [9], we have changed its messaging sequence in our protocol to better serve the specific needs of smart card digital signature. In regular timestamp messaging, a requester sends a timestamp request containing message digest and optional nonce to the TSA, then the TSA returns timestamp response. In our proposed protocol, we use same messaging between the service provider and the TSA, but the service provider redirects TSA's response to a smart card, and the smart card generates another timestamp response and returns this to the service provider again. Since we still use standard timestamp message format, our proposed protocol does not need any change in the TSA.

We developed a prototype of our proposed method as proof-of-concept. We implemented this program with Java Card [10]. Java card is a smart card that executes limited Java bytecodes on smart card chip and the program code is called java card applet. We have to develop parsing libraries from scratch because there are no built-in features in existing java card API or open source projects. For this reason, we developed our own codes on DER decoder and X.509 certificate, X.509 CRL [8], and timestamp response [9] parsers.

The main challenge that we met in developing our codes is that a smart card has very limited resource (1 to 8 kilobyte memory) and decoding certificate, CRL and timestamp packets usually requires a significant amount of memory. This is because these packets usually are ASN.1 encoded content [11] and parsing them requires recursive back-and-force traverse in a binary-tree method, which is used by all open source projects like [12]. In [13], a file system has been proposed for smart card while also claiming the implementation of a DER decoder. However, the proposed algorithm and its impact on the performance has not been presented.

```
0x00,//TimeStampResp
    0x01, //PKIStatusInfo
    0x02, //TimeStampToken (ContentInfo)
        0x01, //ContentType
        0x02, //content [0]
            0x01, //SignedData
                0x01, //CMSVersion
                0x02, //DigestAlgorithmIdentifiers
                0x02, //EncapsulatedContentInfo
                    0x01, //ContentType
                    0x02, //eContent [0]
                        0x01, //octetStringTypeIndex
                            0x01, //TSTInfo
                                0x01, //version
                                0x02, //TSAPolicyId
                                0x02, //MessageImprint
                                    0x01, //hashAlgorithm
                                    0x02, //hashedMessage
                                0x02, //serialNumber
                                0x02, //genTime
                                0x02, //nonce
                0x02, //certificates [0]
                    0x01, //certificate
                0x02, //crls [1]
                    0x01, //crl
                0x02, //SignerInfos
                    0x01, //SignerInfo
                        0x01, //CMSVersion
                        0x02, //SignerIdentifier
                        0x02, //DigestAlgorithmIdentifier
                        0x02, //signedAttrs [0]
                        0x02, //SignatureAlgorithmIdentifier
                        0x02 //SignatureValue
```

Fig. 3. Sample DER content template for timestamp response



(a) (b)

Fig. 4. (a) TspSign and TspSign2 Java Card applets' detailed process time, and (b) their performance test results in comparison with classic sign on three different Java Cards for RSA 1024-bit and 2048-bit key lengths

Thus, we developed a novel approach using a so-called ***one-time scanning*** technique to implement these steps on a smart card with limited resource. In this technique, we use a pre-defined pattern (template) to decode a DER-Encoded content. Our pre-defined pattern of timestamp response is illustrated in Figure 3. We store the it in a one-dimensional array, while considering one movement in each single element. The values of 0x00, 0x01, and 0x02 indicate no movement, visit the child, and visit the brother, respectively. In this way, we don't parse unnecessary nodes whose data are not required by the applet. This one-time scanning approach can significantly save the smart card resource. Therefore, our code parses certificate, CRL and timestamp response with pre-defined templates, and builds four flat arrays offsets, tags, lengths and value indexes. As a result, indexes of fields are fixed and the applet accesses them quickly.

One-time scanning technique works for mandatory fields, but there are some optional fields like revoked certificate serial numbers in CRL. We use classic DER decoding just for this part of content. We published our java card applet and developed parsing libraries as an open source project in GitHub [14] so that other researchers can use and improve the project.

VI. Performance Evaluation

We developed our code based on Java Card 2.2.2 [10] specification and tested it on Java Card 3.0 classic edition [15] too. Furthermore, we developed a simulator for service provider and TSA as a regular computer program to test our applets. We implemented java card applet of our proposed mechanism with classic DER decoding technique and called it "TspSign". Then, we analyzed on-card process and measured the processing time in each of the step listed in Protocol 1, and its results are illustrated in Figure 4(a) with dark bars. As shown in the chart, the majority of time is consumed for parsing steps. To reduce the time consumed by parsing, we developed one-time scanning technique to decode DER-Encoded contents (described in previous section) and called it "TspSign2". By using TspSign2, we measured the processing time of steps again, and its results are illustrated in the same Figure 4(a) with bright bars. It is clear that there is a drastic performance improvement with this technique. We also developed a simple applet which only signs a message digest regularly with no security enhancement to compare the performance of implementation of our prototypes with it. We call it "Sign" applet in our experiments.

In addition, while normal sending/receiving buffer in smart card is 255 bytes, our applet needs to transmit a significantly longer data. To achieve this, we used extended length commands and chaining protocol methods. To generate digital signature, we used PKCS#1 PSS encoding with RSA 1024 and 2048-bit keys and SHA-1. We tested our code with simple RSA key and CRT RSA key [10] which requires primitives of private key, and because of performance considerations we used CRT version in all tests.

We used two different configurations to test 1024 and 2048-bit key lengths. To test 1024-bit key length, we generated a self-signed CA, TSA and card certificates with 1024-bit RSA keys

and to test 2048-bit length, we generated all of them with 2048-bit RSA keys. Therefore, we did not mix 1024-bit and 2048-bit keys in our tests.

For "Sign" applet, the test tool sends 20 bytes SHA-1 hash value to the smart card and receives 128/256 bytes response which is 1024/2048-bit RSA signature value. For TspSign and TspSign2 applets, the test tool sends 1501/2022 bytes which includes a timestamp response conveying nonce, time, message digest, TSA certificate and CRL, and receives 1131/1523 bytes which includes timestamp response generated by smart card conveying message digest, time and card certificate for 1024/2048-bit RSA keys.

To evaluate the performance, we loaded our applets to three popular Java Cards from three different brands including NXP J3A081, Feitian JavaCOS A22 and G&D SmartCafe Expert 6.0. Also, we used OmniKey Card Man 3121 USB card reader with T=1 communication protocol for all tests. We repeated each test case 10 times and calculated their average values. Thus, we loaded our three applets "Sign", "TspSign" and "TspSign2" to all three smart cards for both 1024 and 2048-bit configurations, and totally, we executed 180 successful tests. The results are illustrated in Figure 4(b) top and bottom charts.

VII. Related Works

There is a popular security mechanism to provide confidentiality and integrity for smart card transactions which is called Secure Messaging [16]. In secure messaging mode, smart card and terminal encrypt some parts of or entire smart card's command and response, and any intermediate entity cannot sniff or change data between terminal and smart card. But secure messaging cannot prevent terminal attacks because data is unencrypted in terminal.

Authors of [17] proposed to use a smart card that has LCD monitor to directly display information to a user. It's clear that this solution is not practical because there is no widely-used smart card with this special feature. Additionally, this type of monitor has limited capability and cannot show full text or handle any complicated digital document format.

Authors of [18] introduced a secure digital signature which is a two-phase signing process. In this mechanism, a user signs data by her smart card on an untrusted terminal and has a limited time (deadline) to reject it in a trusted terminal, otherwise the signature becomes committed. It is similar to a credit card which user can cancel his unwanted transactions. In other words, a user can sign data on any untrusted terminal, but should check the data in a trusted terminal such as his personal computer after the signing and cancel unknown signatures. The disadvantage of this scenario is that there should be at least one trusted terminal.

VIII. Conclusion

In this paper, we proposed a novel protocol to secure smart card signing with time-based digital signature. At first, we explained our threat model and illustrated our implementation of two well-known attacks against smart card terminal. Then, we proposed our new protocol to prevent altering digital signature just before signing and presented our security analysis. In addition, we developed a prototype with one-time scanning DER decoder to improve performance due to smart card's limited resource, and our test results show that this technique has significantly reduced processing time in smart card.

References

[1] B. Schneier, A. Shostack, "Breaking up is hard to do: modeling security threats for smart cards", USENIX Workshop on Smart Card Technology, USENIX Press, 1999, pp. 175-185.

[2] "FIPS PUB 140-2: Security requirements for cryptographic modules", National Institute of Standards and Technology, December 2002

[3] "SP 800-73-3. Interfaces for Personal Identity Verification", National Institute of Standards and Technology, February 2010

[4] "Windows smart card minidriver specification", Version 7.07, Microsoft corporation, February 2016

[5] "MinidriverSpy", https://github.com/hosseinpro/MinidriverSpy

[6] ACR83 PINeasy Smart Card Reader, http://www.acs.com.hk/en/products/34/acr83-pineasy-smart-card-reader/

[7] Thumbs Up: Mastercard Unveils Next Generation Biometric Card, https://newsroom.mastercard.com/press-releases/thumbs-up-mastercard-unveils-next-generation-biometric-card/

[8] R. Housley, W. Polk, W. Ford, and D. Solo, Internet X.509 Public Key Infrastructure certificate and certificate revocation list (CRL) profile, Request for Comments RFC 3280, 2002

[9] C. Adams, P. Cain, D. Pinkas, R. Zuccherato, Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), Request for Comments RFC 3161, 2001

[10] "Java Card Runtime Environment (JCRE) Specification," 2nd Edition, 2002

[11] Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), ITU-T Recommendation X.690, 2002

[12] https://www.bouncycastle.org

[13] M. Imam, M. Sobh, "Novel File System with ASN.1 Support for Java Card Applications", 23rd IEEE International Symposium on Industrial Electronics (ISIE), June 2014

[14] "TspSign", https://github.com/hosseinpro/TspSig

[15] "Java Card Runtime Environment (JCRE) Specification," 3rd Edition, 2011

[16] "ISO/IEC 7816: Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange", International Organization for Standardization, 2013

[17] S. Li, A.R. Sadeghi, S. Heisrath, R. Schmitz, J.J. Ahmad, "hPIN/hTAN: a lightweight and low-cost e-banking solution against untrusted computers", 15th International conference on financial cryptography and data security, volume 7035 of Lecture Notes in Computer Science, Springer, 2011

[18] I.Z. Berta, L. Buttyan, I. Vajda, "A framework for the revocation of unintended digital signatures initiated by malicious terminals", IEEE Transaction on dependable and secure computing, September 2005.