Uni-directional Trusted Path: Transaction Confirmation on Just One Device

Atanas Filyanov*, Jonathan M. McCune†, Ahmad-Reza Sadeghi‡, Marcel Winandy*
* *Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany*
{*atanas.filyanov, marcel.winandy*}*@rub.de*
† *CyLab, Carnegie Mellon University, USA*
jonmccune@cmu.edu
‡ *Center for Advanced Security Research Darmstadt / Technical University Darmstadt, Germany*
ahmad.sadeghi@cased.de

***Abstract*—Commodity computer systems today do not include a full trusted path capability. Consequently, malware can control the user's input and output in order to reveal sensitive information to malicious parties or to generate manipulated transaction requests to service providers. Recent hardware offers compelling features for remote attestation and isolated code execution, however, these mechanisms are not widely used in deployed systems to date. We show how to leverage these mechanisms to establish a "one-way" trusted path allowing service providers to gain assurance that users' transactions were indeed submitted by a human operating the computer, instead of by malware such as transaction generators. We design, implement, and evaluate our solution, and argue that it is practical and offers immediate value in e-commerce, as a replacement for captchas, and in other Internet scenarios.**

***Keywords*-security; transaction confirmation; trusted path; trusted computing;**

I. Introduction

For decades a significant challenge in computer security has been realizing a mechanism for establishing a full *trusted path*, i.e., a mechanism that (i) isolates the input and output channels of different applications to preserve the integrity and confidentiality of data exchanged with the user, (ii) provides a technical means that assures the user of a computer system that she is truly interacting with the intended software, and (iii) assures running applications that user inputs truly originate from the actions of a human (as opposed to being synthesized or injected by other software).

Software vulnerabilities in the increasingly complex software stack on commodity computing devices offer a large attack surface that attackers can exploit to inject malicious Trojan horse software, such as keyloggers [1] or transaction generators [2]. These software-based attacks can eavesdrop on user input or modify transactions in a malicious way, e.g., waiting until the user has legitimately authenticated to the web server, and then issuing illegitimate transactions using this channel (e.g., by faking or scripting user input).

Some trusted path proposals suggest the use of an external *verifier* device. However, the increasing use of mobile devices for sensitive online transactions and e-commerce applications make these solutions inconvenient, especially given today's trend in device convergence (e.g., smartphones including cameras, web access, GPS, etc.). Using one's mobile phone to verify one's laptop may be reasonable in some scenarios, but there are no verifiers available if the transaction itself is taking place on the mobile device while on-the-go. Thus, the challenge remains to somehow provide a trustworthy mechanism on *just one device* that can defend against transaction generators and malicious scripting that imitates actions intended to be performed only by humans.

An additional disincentive for existing solutions is that end users are rarely exposed to the liability for fraud. In the US and much of Europe, credit card companies, banks, and online merchants absorb this liability. Thus, it is these institutions that have the incentive to make online transactions more resilient to attacks.

We design, implement, and evaluate a secure transaction confirmation architecture that can provide assurance to a remote server that the user of a client system has indeed confirmed a proposed action. With this property, a service provider can ensure that it only commits transactions on the user's behalf if the user has actually confirmed the transaction. We are able to realize such a system by providing the properties of a trusted path *in one direction only*.

Restricting ourselves to a uni-directional trusted path enables us to realize a significantly more practical implementation, remaining compatible with legacy operating systems and applications. Our goal is to provide remotely-verifiable evidence to service providers of events taking place on the physical user-centric I/O devices. That is, *we wish to prove to a remote server that an actual user typed something or saw something displayed on-screen*. This makes it possible to achieve trusted transaction confirmation on *just one device*. Online service providers using our system are able to detect and reject transaction generators. In contrast to a full trusted path, we do not provide local, user-verifiable evidence of the genuineness of output, i.e., malicious code may still fake the confirmation process from the perspective of the user.

When implemented properly, the client-side user experi-

ence need not deviate from what users expect today. This same technology also enables a more usable captcha [3] mechanism, since our framework enables client platforms to generate evidence of events originating on user-controlled peripherals. We consider application scenarios where human users can perform actions (e.g., enroll for services or issue transactions) on Internet services that require a verification of the user's intent. E-commerce web sites, online banking, e-government services, and even e-voting are all examples.

Based on a uni-directional trusted path, we architect a trusted confirmation and optionally authentication (§III-D) agent that executes in CPU-provided isolation. The trusted computing base (TCB) of the trusted agent is relatively small compared to prior works (a few thousand lines of code), and remains compatible with users' existing operating system and application environments. We evaluate the performance and security of our implementation, and provide an estimation of the efforts needed to port our solution to other hardware architectures, such as smartphones.

II. Problem Definition

A. Background: Trusted Path

Several approaches for realizing a full trusted path have been proposed over the years, though none enjoy widespread adoption on commodity systems. We briefly review some representative designs, and their drawbacks.

One approach is to use a full trusted OS and window manager, where a dedicated area of the screen is reserved for the exclusive use of a trusted software component that shows the identity and status of the current application [4]. While this concept has been implemented in some research systems [5], [6], the goal of widespread adoption has remained elusive.

A closely related variant leverages the notion of a secure attention sequence, e.g., "Press Control-Alt-Delete to log on." The assumption here is that the OS kernel remains uncompromised, and will always be the first software layer to process keyboard input. Thus, any spoofed login dialog box will be immediately overwritten by the legitimate box. However, users must be taught to always press the necessary key sequence. Another approach for indicating a trusted state of a computer system is based on its ability to display a "secret picture" [7] (or any other human-recognizable secret). The system is architected such that the image data can only be decrypted if the system is running approved software, and the user must diligently remember to look for her designated image whenever she performs a security-sensitive operation. A final approach is to use some form of dedicated additional hardware as an axiomatically trustworthy indicator, in the limit something as simple as a dual-color LED [8]. This design is compelling as it still enables full screen applications, which must otherwise be disallowed given their ability to spoof other security indicators.

Unfortunately, none of these designs enjoy widespread deployment for online transactions. While additional hardware costs are sometimes a factor, we believe a significant barrier to adoption on commodity systems is a lack of interest from users. Vendors and financial institutions absorb the majority of the risk in fraudulent online transactions today. A confirmation system can be constructed based on a uni-directional trusted path that does not strictly require any additional work from the user, beyond reading a transaction summary before finalizing a transaction (an action that is already a standard part of online purchases).

B. Adversary Model

The model we consider involves four parties: (1) the user, (2) the user's computing device, (3) the service provider, and (4) an attacker. The attacker has complete control over the network between the service provider and the user's device. Thus, the attacker may try to impersonate the service provider to trick the user. Moreover, the attacker is able to install malware on the user's computing platform or modify any existing software there. The goal of the attacker is to issue transactions to the service provider illegitimately on behalf of the user.

C. Assumptions

While the adversary has control over the user's software environment, we make the following exception. We assume the device has some form of secure execution environment that is protected against software-based attacks, and that the hardware is correctly implemented and protects the integrity of the secure environment. Note that AMD SVM [9] and Intel TXT [10] are examples of widely deployed hardware capabilities that can provide the necessary properties. Moreover, we assume that the attacker cannot gain physical access to the user's platform. We assume that the service provider is honest, and that the service provider's servers are secure. We do not consider denial-of-service attacks.

D. Security Objectives

Our main security requirements are the following:

1) Mutual authenticity of the user and service provider: The user must somehow authenticate herself to the service provider, and the user's device must somehow authenticate the service provider on behalf of the user.

2) Integrity of the transaction: Authenticated users can engage in transactions with the service provider. Transaction requests must be integrity-protected both during final confirmation on the user's computing device and during transmission to the service provider.

3) Intention of the human user: The service provider must gain assurance that a requested transaction is indeed submitted by a human user, and not by malware. For this, the service provider must be able to reliably verify that a certain action (e.g., receive keypress events from a keyboard) has been performed in a secure execution environment.

4) Binding between user and machine: The service provider must be able to verify the binding between a confirmation and a particular transaction request from a certain machine.

The last requirement prevents proxy attacks, where confirmations are re-directed to other machines to get other users perform the required action. However, where privacy is a concern, the *real* identity of the user's hardware device should be appropriately anonymized, e.g., via a trusted third party or cryptographic protocols [11].

These requirements are typical for e-commerce scenarios. We prioritize application scenarios' integrity requirements over their secrecy requirements (excluding authentication). For example, in e-commerce or online banking applications, a Trojan horse plug-in that resides in the web browser may eavesdrop on account balances or what items have been recently ordered. However, the malware is prevented from generating or modifying transactions.

III. Architecture and Design

In this section we describe the architecture and design of our uni-directional trusted path (UTP) solution. Our primary design goal is to provide an integrity-protected confirmation facility even when client users possess only a single device without full trusted path capabilities.

A. High-Level Design of UTP

The main property of UTP is to enable a remote server to gain a high level of assurance that a certain action submitted by a client system was initiated at the behest of its physically-present human user, and not by malware such as a transaction generator. This means UTP realizes properties (i) and (iii) of a trusted path (c.f. §I), but not necessarily property (ii). Hence, users might not be able to verify that they have been interacting with the intended application or web site. The underlying motivation for such an architecture is the simple fact that most users do not actually pay attention to security indicators [12], [13], [14].

If anything goes wrong, e.g., malware tries to modify a requested transaction or generate a new one, the remote server in our design will notice this and discard such malicious transactions. This is the essence of the value provided by UTP. The trusted path goes (verifiably) from the user to the server, and enables the server to significantly reduce its exposure to transaction generators and other large-scale, automated malicious client-side behavior.

UTP must enable a remote server to gain assurance that the client platform is running a software configuration that can take total control of the user-centric I/O devices, including the keyboard, display, and optionally the mouse. The client system must have the capability (via hardware or software mechanisms) to construct a trustworthy *channel* between these platform components, and to generate some form of remotely-verifiable evidence that it is indeed in control of these devices. We explicitly do not require local, user-verifiable evidence of the presence of these channels, as this implies the full trusted path capability.[1]

One viable approach for providing such a capability is an isolated execution environment with remote attestation capabilities, such as Flicker [15]. Flicker enables security-sensitive code to execute in hardware-enforced isolation from all other code and devices on the system, and to generate attestations enabling remote verification that the execution environment is established as intended. The security-sensitive code can be constructed to take control of the user-centric I/O devices (i.e., keyboard and display), and the remote verifier can ascertain, given the code and assumed correct hardware, that user-centric I/O is working as intended.

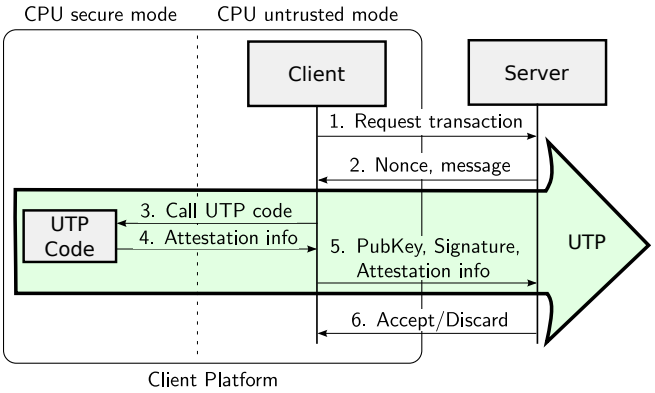

Figure 1. General architecture of UTP.

Figure 1 shows the high-level design of UTP. When the client requests an action from the server that requires a confirmation of the user's intent, the server establishes the uni-directional trusted path by sending a message to the client (and a random nonce for the purpose of preventing replay attacks). The (untrusted) client program invokes the execution of the UTP code in the CPU's secure execution mode. This mode ensures that the UTP code executes isolated from other software and successfully takes control of the user-centric I/O devices. The UTP program displays the message provided by the server (e.g., a transaction summary) to the user. Once the user has viewed the message and acted as required (e.g., confirmed her intention to submit the transaction), UTP assembles the necessary data to generate an attestation that these events transpired while in the secure execution mode. This information is cryptographically signed by a keypair that is accessible only while the isolated

[1]Note that our UTP architecture can readily support the "secret picture" form of full trusted path [7]. However, we are highly skeptical of users' motivation to diligently remember to check for their picture, and to actually stop their primary task if it is missing [12], [13], [14]. Our emphasis in this paper is on the utility of a mechanism for which users need not dedicate any additional time or attention, as compared to existing systems.

execution environment is active, in order to demonstrate its authenticity and integrity. The data and its signature are then sent to the server. The server can verify the signature with the provided and (e.g., from a trusted third party) certified public key of the client platform, and subsequently verifies the attestation information in order to get assurance about the execution of UTP. Note that UTP is more than just TCG-style attestation [16] (see also the Appendix): If the verification succeeds, the server knows that a uni-directional trusted path to the human user has been established and that the transaction can be processed.

Advantages: The realization of UTP as a small trusted program using a secure execution mode of the CPU has some advantages over solutions based on a secure OS:

- *Smaller trusted code*: The overall trusted computing base (TCB) is even smaller than a secure user interface system on top of a secure operating system kernel (e.g., [5], [6]). A lot of functionality can be "out-sourced" to the untrusted environment, including the initialization of certain hardware resources. Moreover, in contrast to a security kernel or hypervisor, our design requires no persistent code when no sensitive transactions are in progress.
- *Higher adoption rate*: The chances of adoption are much higher since our system can be readily deployed with off-the-shelf operating systems that are used today. Although modern virtualization technology offers a compelling way to run a secure hypervisor beneath commodity operating systems (e.g., [17], [18], [19], [20]), monopolization of available hardware virtualization support for security architectures has been discouraged.

B. Transaction Confirmation with UTP

We describe the interactions and user experience of transaction confirmation with UTP. The basic procedure begins after the user has prepared some kind of transaction (e.g., by populating a shopping cart at a web-based retailer) and is ready to commit (e.g., make a purchase). At this point, a well-behaved client system will invoke UTP to allow the user to confirm that transaction, though a malicious client system may attempt to fool the user.

Once invoked, the UTP code takes control of the keyboard and display and shows a summary of the transaction that is about to be committed. During this time the normal (untrusted) OS is suspended, and the UTP code executes in the secure execution mode of the CPU (c.f. step 3 in Figure 1). Displaying the transaction summary and requesting an active confirmation from the user is an essential step that reveals the human user's intentions. Otherwise, the user may be tricked into confirming anything. Note that the practical details of how to summarize the transaction are likely best implemented on the server side, either per-web server or as a standard mechanism that is adopted across several web-based service providers. It may also be an option to implement client-side logic on a per-provider basis, if the user has a finite set of service providers with which she engages in sensitive transactions. One can even imagine the content of the confirmation page being rendered somewhere in the cloud, with a static image being transmitted to the UTP environment on the client system.[2]

Next, the user is prompted to perform the actual confirmation. If no attack is taking place, then the user is indeed looking at a legitimate summary of the proposed transaction, and the UTP code will be the only code that can receive the user's forthcoming input. This confirmation is the user's opportunity to realize that an attacker has tampered with her session. The server wants to be sure that the user actually verified the summary. From a technical perspective, the actual confirmation act may be as simple as pressing "enter" to generate a physical keyboard interrupt, but we also wish to avoid habituation by users. Interesting designs may include manually entering the final invoice amount; entering a short, random confirmation code; or typing the word YES to continue. The precise design is outside the scope of this paper, but it should be selected in response to an evidence-based user study to determine which is least intrusive while providing the desired security properties. We discuss these issues again in our security analysis (§V-A).

Once the user has viewed the transaction summary and confirmed her intention to commit the transaction, UTP assembles the necessary data to generate the attestation information and terminates its execution. The CPU exits the isolated environment and resumes the normal OS. The client (e.g., the web browser) sends the attestation information via a secure channel to the remote server. The server accepts the pending transaction if it is able to verify the attestation. This indicates that the server is convinced that the legitimate and expected software ran in an isolated environment on a device associated with the user's account, and that the user successfully confirmed her desire to commit the transaction.

Our design leverages an isolated execution environment. Some implementations of such an environment necessarily halt the execution of other code running on the system. For example, the user's music will stop playing when the confirmation dialog is displayed. An important question to resolve is whether this disruption is good or bad. A "pro" is that this disruption helps to get the user's attention and focus them on the sensitive task. A "con" is that the users may find this requirement annoying, and vendors may object on the grounds that it will reduce the rate of impulse purchases.

C. CAPTCHA with UTP

CAPTCHA is defined as a Completely Automated Public Turing test to tell Computers and Humans Apart. Captchas are commonly manifested online as distorted images, and users are asked to type in the characters contained in the

[2]This type of pre-rendered design has been reviewed favorably, e.g., for electronic voting systems [21].

images. Unfortunately, captchas are often difficult even for humans to solve reliably [3]. Moreover, without a secure execution environment, captchas can readily be inlined into other types of free content. For example, it has been shown that there is no shortage of users willing to solve captchas for otherwise unfettered access to adult content online [22].

Our UTP solution is capable of supporting captchas that are simultaneously more usable and harder for attackers to inline. In essence, a UTP-based captcha amounts to asking the user to verify some transaction detail, e.g., to enter the final amount of a pending purchase.

D. Mutual Authentication Overview

As we have shown, UTP can be used to give a remote server assurance that a human user has actively entered a confirmation to a requested action. One of our motivations is to protect against malicious transactions resulting from malicious transaction generators. In scenarios where the transaction is associated with a user account (e.g., online purchases), malware could steal the user's account credentials and send them to the attacker. Given that the user cannot ascertain whether her system is truly in a trustworthy state, we cannot depend upon the user to employ appropriate discretion when deciding whether to enter sensitive data such as passwords or account credentials. Thus, we include in our architecture that the server is able to authenticate the client user without solely depending on user-memorized secrets (e.g., passwords). However, authentication is orthogonal to our work, and we therefore refer to existing approaches that can be easily incorporated into our design.

One option for mutual authentication is to use public key-based credentials. For the purposes of discussion, we consider strong device authentication based on hardware-protected keys, e.g., asymmetric non-migratable keys residing in a Trusted Platform Module (TPM) [16]. Viable alternatives may include password-based authentication, where the password is actually a high-entropy secret managed by and accessible only to trusted code. The PAKE protocol for password-assisted key exchange may also be used [23], [24], [25]. Challenge-response protocols, which verify knowledge of a shared secret between client and server, are another option. Finally, wallet-like authentication agents [26], [27], [28], [29], [30] automatically manage mutual authentication between the user's computer and a remote server. The authentication agent (wallet) is generally executed in an environment that isolates it from the rest of the software stack. Hence, an authentication agent could be executed in the secure execution mode of the CPU in the same way as the UTP code (see also §VI).

E. Enrollment and Setup

When UTP is used in the context of confirming transactions demanding (mutual) authentication, the credentials for the authentication (e.g., password, shared secret, or public key-based credentials) must be established in a setup phase. The primary obstacle is enrolling the public component of a hardware-protected keypair from the client system with the service provider for subsequent use in client device authentication and attestation. This challenge has two parts. First, the hardware-protected key generation and storage infrastructure employed must be capable of certifying that a particular keypair was generated internally and will remain protected. Second, this certified key must be somehow bound to the identity of the device's user, in order to support an authentication procedure that remains secure even if the user enters her password into a malicious application.

Certifying Hardware-Protected Keys: Although technically well-understood, certifying public key-based credentials is challenging because a global public key infrastructure that can scale to include all client devices does not exist. Thus, in practice, it is likely that some form of per-service-provider bootstrapping mechanism will be required. It may be reasonable for organizations to partner such that the bootstrapping mechanism can be out-sourced to a third-party provider (with parallels to single sign-on solutions, e.g., SAML [31] or OpenID [32]). In cases where privacy is required (e.g., unlinkability of online purchases to the real identity of hardware devices), cryptographic protocols like Direct Anonymous Attestation (DAA) [33] may be used, where a signature proves the membership to a certain group, but does not reveal the identity of that particular member.

Binding Keypairs to Users: To enable the service provider to associate a certified keypair with a user, users must provide certain information about themselves (e.g., name, e-mail address, etc.). The initial enrollment can be performed on a trust-on-first-use basis in many use cases.[3]

To bind a certified keypair to a particular user, we can use our protected input mechanism to demonstrate to the service provider that a human entered the relevant information. For instance, the UTP-based captcha approach described above can be used. This solution demonstrates that the platform from which the human's information originates is able to perform operations using the private key corresponding to the certified public key.

IV. Implementation

We have implemented an end-to-end system for transaction confirmation using a uni-directional trusted path. Our setup consists of a modified open-source online merchant software package on the server side, and a browser plugin and confirmation agent on the client side. For the client side, we have implemented the UTP confirmation agent based on the Flicker [15] framework, leveraging hardware support for dynamic root of trust using Intel TXT CPU

[3]Of course, one person could pose as another if the impostor knows sufficient information about the individual being impersonated and if the target individual has never been enrolled before. Note that this risk already exists today, and solutions that address it are outside the scope of this paper.

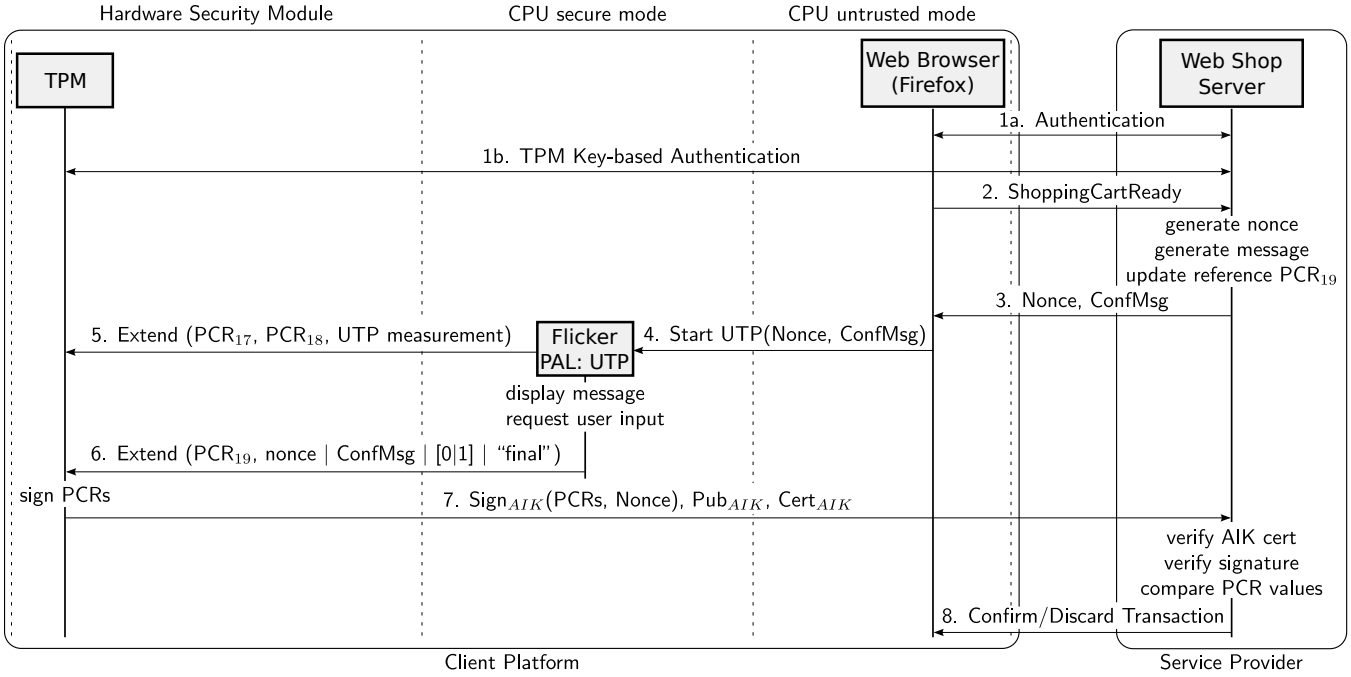

Figure 2. Prototype implementation of the UTP transaction confirmation.

extensions [10] and a v1.2 TPM [16]. Figure 2 illustrates our implementation, which we describe below. Appendix A gives additional background.

A. Authentication and Enrollment

User and Device Authentication: We implement two-factor client authentication by authenticating the client device as well as the client user. Thus, part of the user's enrolling with our web shop includes generating and registering a TPM-based Attestation Identity Keypair (AIK) to represent the user's client system to that service provider.[4] Hardware support for protecting cryptographic keys, such as the TPM chip [16], has reached a level of market penetration where it is a legitimate option on which to build real-world systems. Hence, device authentication, when implemented leveraging the TPM, provides excellent protection against a compromised OS being able to masquerade as a different physical system.

Enrollment Phase: We use an AIK to represent the identity of the client's TPM, and thus platform. In our implementation, we use a privacy CA to certify our AIK as having been generated in a real TPM. Specifically for our prototype we use `privacyca.com`, which provides an interface to obtain a valid AIK certificate that can be used by the TPM, given that the TPM manufacturer included an Endorsement Key Credential (Figure 3). Instead of including a trusted third party as privacy CA, one might achieve unlinkability, as mentioned before, by using DAA [33], [34].

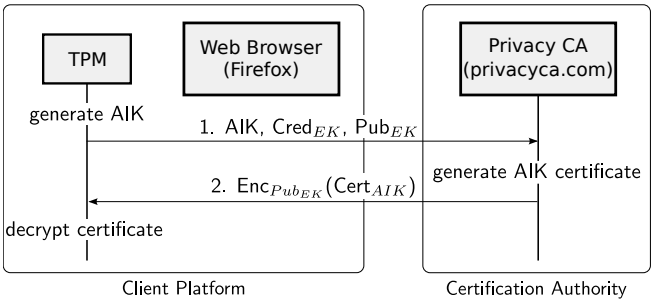

Figure 3. Device enrollment using a Privacy CA.

B. UTP Sessions

Web server: We made minor changes to the open-source FreeWebshop.org [35] to support UTP-based transaction confirmation. The server has a typical shopping cart interface where users can add items that they wish to buy, before proceeding to a check-out process that collects payment and shipping information. We modified the summary page of the ordering process in FreeWebshop.org to appear without automatically finalizing the order, and to include one additional button, which invokes UTP-based verification. When the user clicks this button, a server-side script that we have implemented to manage the UTP-based verification process is invoked. The script assembles the final verification

[4] A unique AIK can easily be used per-merchant, alleviating any risk of privacy invasion by correlating transactions between different merchants.

message to be confirmed by the user and sends it to the client. This message consists of a simplified invoice (short name for each product in the cart, quantities, itemized cost, and the total cost) and a nonce to ensure the freshness of the expected attestation covering the UTP-based verification on the client (Figure 4). The script is responsible for the exchange of the actual attestation messages and all data is tunneled through the existing https connection.

Our server-side script consists of 132 lines of code. Further, our verification program consists of 356 LoC, which includes nonce generation and verification of the client TPM's Quote operation and AIK certificate.

As soon as a client confirms a specific transaction and engages in the verification protocol, the server-side script invokes a local verification program, that we have developed, which generates a nonce for that particular user. The nonce is used to ensure the freshness of the forthcoming attestation of the UTP confirmation session on the client. The verification program also computes the expected attestation result with the nonce and the hash of the generated confirmation message. Those two hashes are later verified to be included in the hash chain that comprises the value in the TPM Quote, which is sent during attestation from the client.

After the confirmation session on the client, an attestation is sent back to the server, again through the https connection. When the server-side script receives the data it parses it and invokes the verification program to process the TPM Quote (AIK-signed PCR aggregates), and makes the decision as to whether this attestation represents a legitimate confirmation of the pending transaction. Depending on the result of the verification, the script sends a message to the client browser (that malware may suppress), stating whether or not the transaction is confirmed, and either commits or aborts the transaction in the web shop's backend.

```
Confirmation Agent

To confirm the purchase of the following 3 items:
1. Widget           50 $
2. Doodad           10 $
3. Thingamajig      50 $
--------------------------
TOTAL              110 $

Please type this in exactly: 3e

>: 3e

You typed in: 3e
Transaction will be confirmed.
```

Figure 4. Sample UTP confirmation display.

Client PC: We have developed an extension for the Firefox web browser, which runs on the untrusted OS on the client and is capable of sending and receiving data through the existing https connection. We have also developed a local utility program that is capable of invoking a UTP-specific, security-sensitive PAL (Piece of Application Logic) to run in a Flicker-based secure execution mode that is realized using Intel TXT [15]. Additionally, the program performs the attestation after the UTP Flicker session has completed.

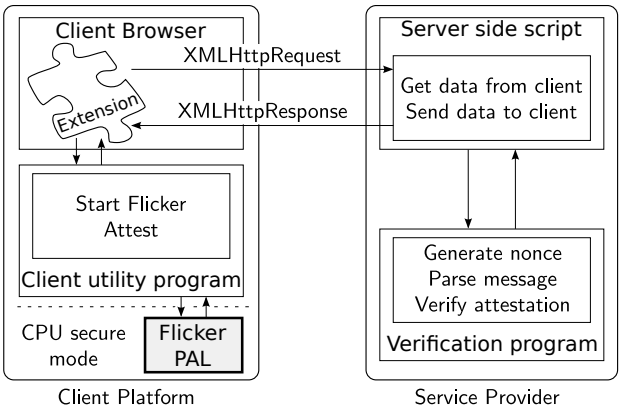

Figure 5. Implemented modules in the prototype.

In the common usage scenario, the user opens the web site of the online web shop in Firefox and selects items to buy. When the user is ready to submit her order, she presses the "Submit" button on the final summary page of the web shop. The extension then detects that this button has been clicked (because it has a special ID) and connects to the web server to request a nonce and a confirmation message. When the extension receives the requested files, it calls the client utility program, which invokes text mode in the graphics card[5] and formats the received data as inputs for a Flicker session. Note that leveraging the untrusted code to invoke text mode simplifies the work that must be done inside the Flicker-protected session. When invoked, the Flicker framework executes using TXT-based hardware protections, with the UTP module as its PAL. Figure 5 shows the additional modules we have implemented.

Table I shows the lines of code (LoC) for the implemented modules on the client side, as well as the code size of the existing Flicker framework that we build upon and that belongs to the TCB of our UTP agent. The overall size of the TCB (including drivers for user I/O) is only a few thousand LoC.

Confirmation Agent (client-side): When invoked, the UTP module displays the confirmation message. It prompts the user to enter a randomly chosen character sequence (to avoid habituation by simply hitting ENTER). Figure 4 shows our sample confirmation dialog.

When the correct character sequence has been entered, UTP extends PCR 19 with the nonce and the confirmation

[5]We have used text mode for simplicity, since a basic VGA driver is straightforward to implement inside Flicker. A fully graphical confirmation page could also be implemented, but it would require additional support inside Flicker for the graphics mode(s) of interest.

	Module	Language	LoC
untrusted	Client utility program	C	321
	Firefox extension	JavaScript	95
	AIK generation program	C	540
	Sum		**956**
trusted	Keyboard & display drivers [a]	C	254
	Flicker PAL	C	260
	Flicker TCB	C/Assembly	741
	UTP helper functions	C	1080
	Sum		**2335**

Table I
CODE SIZES ON CLIENT SIDE.

[a]The given LoC include the PS/2 and VGA drivers. Support for USB keyboards would increase the code size by 1500-2000 LoC.

text string that was provided by the server. If an incorrect sequence is entered (or if the user chooses not to confirm the transaction), then a failure message is extended into the PCR. Finally, a special value denoting that the secure session is ending is extended into PCRs 18 and 19. Though a well-behaved OS will prevent application software from accessing the TPM at localities 2 and 3, a malicious OS can access those addresses. On Intel hardware, PCR 17 can only be extended by Intel's SINIT Authenticated Code Module (we use Q35_SINIT_17.BIN for our prototype).

After the UTP module returns, the utility program requests a TPM Quote that includes PCRs 17 and 18 (containing measurements of Flicker and the UTP code), and PCR 19, which contains the hashed nonce and confirmation text string. The nonce will also be used by the TPM for signature generation during the attestation operation.

Confirmation Verification (server-side): The verification script receives the data and verifies it according to the following criteria:

- The signature is valid and from a registered TPM (i.e., one that has been registered during the enrollment phase). We currently authenticate the client device based on a simple whitelist of allowed devices, realized as a list of known public AIKs. We note that more sophisticated authentication methods are readily applicable (e.g., [23], [24], [25]).
- The PCR values for Flicker and the UTP code are correct in PCRs 17 and 18, i.e., they are on the known-good list.
- The PCR value of the nonce and confirmation text string in PCR 19 is correct, i.e., corresponding to those sent by the verifier for this transaction confirmation.

When all checks are successfully performed, the transaction is accepted by the web server, and the user will see a corresponding message in the (well-behaved) web browser.

Detailed PCR Contents: Here we describe in detail the exact data that is extended into PCRs 17–19 in our prototype. This helps to illustrate precisely what code and operations comprise the software Trusted Computing Base (TCB) for the UTP verification session.

- PCR 17: (these values represent the SINIT module for SinitMleData.Version 6 [10]):

```
Extend(SHA1(SinitMleData.SinitHash |
            SinitMleData.EdxSenterFlags))
Extend(SHA1(SinitMleData.BiosAcm.ID |
            SinitMleData.MsegValid |
            SinitMleData.StmHash |
            SinitMleData.PolicyControl |
            SinitMleData.LcpPolicyHash |
            (OsSinitData.Capabilities,0)))
```

- PCR 18: (this is precisely the hash of the Flicker-based PAL that we have written for UTP):

```
Extend(SinitMleData.MleHash)
```

- PCR 19: (these are the input and output parameters of the UTP Flicker session, where 1 means confirmed, and 0 means not confirmed):

```
Extend(1|0); Extend(nonce); Extend(SHA1(ConfMsg))
```

C. Deployment

The design of our UTP confirmation agent is generic and not service provider-specific. Since the confirmation message is provided by the server for each transaction, the service providers can adapt the content or look and feel of these messages without the need to redesign the UTP agent. The end user is not required to have any understanding of the code that runs in secure mode, nor is there any need to do per-system or per-TPM generation of expected hashes.

To use our UTP agent, the user only needs to download the agent code and the browser extension and install it on an OS that supports the Flicker framework (currently Linux). The same agent code and browser extension can be reused across many platforms. There may need to be a few variants (e.g., Windows vs Linux, Intel vs AMD), but we do not foresee a significant scalability problem.

To support a wide range of hardware for user input, our prototype includes drivers for VGA display and PS/2 keyboards, and optionally USB keyboards. On PC-class platforms there is a basic set of VGA and PS/2 I/O available (even on systems with USB keyboards). These can be used as a widely available fall-back in the event that more sophisticated drivers for a particular platform are not available. Our current prototype is built with the assumption that the OS is the lowest layer of system software. However, Flicker and UTP can coexist with virtualization software. A hypervisor such as Xen [36] could be readily modified to enable Flicker and thus our UTP agent.

D. Performance Evaluation

We have implemented and tested our prototype on a Dell Optiplex 755, with a 3.0 GHz Intel Core2 Duo E6850 and ST Microelectronics v1.2 TPM. We run Ubuntu 9.04 (i386) and Linux kernel 2.6.30-6.

Figure 6 shows the timeline of a typical transaction confirmation. While the system waits for user confirmation in secure mode, the rest of the OS is halted, and thus the user's current work is suspended. However, all of the steps before entering secure mode and after resuming the system happen in background and are transparent to the user. There is a delay time of approximately one second before the actual confirmation summary is displayed. This time includes the time to switch the graphics card to text mode, enter secure mode, and initialize the video device in the Flicker PAL. However, such a short delay is quite common across web sites when waiting for a summary of a transaction.

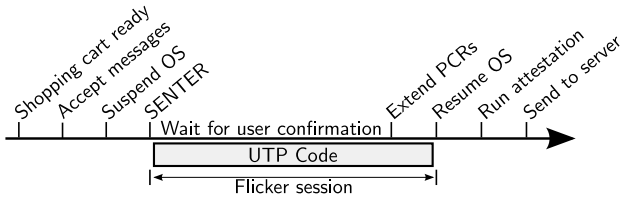

Figure 6. Timeline of a typical UTP session.

V. Security Analysis

The security of our approach relies mainly on the following three properties: (1) the server accepts only transactions that are properly confirmed; (2) the UTP agent is executed in an isolated mode where it is protected against software-based tampering, and where it solely controls the user-centric I/O devices (keyboard, screen, mouse); and (3) the remote server can get assurance as to whether the confirmation agent was actually executed in the secure mode and that the user indeed confirmed the transaction via the keyboard. Also required is that the user actually read the transaction summary. We discuss different attack scenarios in the following.

A. Transactions Must Be Confirmed

The essence of our approach is that a specially designed confirmation agent is executed in an isolated environment on the client system to enable the user to confirm a proposed transaction. Given the strength of our adversary (superuser privileges on the legacy OS), we ultimately depend on the user during a final confirmation step. Thus, the usability of the confirmation procedure is of utmost importance.

We also briefly comment on the ability of attackers to attach malicious input devices or compromise other peripheral devices and reprogram them to masquerade as input devices. While heuristics such as refusing to verify a transaction on a system with more than one USB keyboard attached may offer some level of protection, fundamentally this is a significant challenge. However, solutions where the user must transcribe a few digits from the display again raises the bar for attackers, as they would also need a malicious video interface card. This is a significantly more complex task than compromising [37] or impersonating a USB keyboard.

In our implementation, the server sends the confirmation message to be displayed by the UTP confirmation agent. The agent displays the message, and requests that the user enter a few (randomly chosen) characters as confirmation. The adversary (e.g., malware on the user's computer) may modify a user-generated, or construct a new, transaction request and send it to the server. He has this power because he can control the execution of the web browser and OS as long as the isolated execution environment is inactive.

Though the details of the malicious transaction are displayed on-screen, the user could potentially confirm the modified transaction through an oversight. For example, a study of transaction authentication via mobile phones as used in some online banking applications has shown that users do not always check the confirmation message carefully enough [38]. Thus, the particular details of the displayed message, and the actions required of the user that will be interpreted as confirmation, are of great interest.

Here, we focus on the technical realization of the one-way trusted path to realize secure transaction confirmation. The definition of confirmation messages is up to the service provider. The transaction confirmation may be improved by letting the user enter a transaction-specific detail as confirmation instead of a few random characters. Researchers have shown that users' intentions are more accurately captured when the user actively re-selects the destination of a transaction from a list of different destinations [29].

B. Isolated Execution

Our confirmation agent executes with hardware-enforced isolation, taking advantage of Intel TXT and the Flicker architecture to protect itself from all other code and devices on the system. Our confirmation agent is specifically constructed to take control of the user-centric I/O devices, i.e., the keyboard and graphics display. The adversary has two options for tampering with the confirmation agent: (1) manipulate or replace the confirmation agent code when it is going to be executed, or (2) execute an entirely different application that fakes the confirmation agent display on screen (thereby fooling the user).

In case (1) the malicious agent will execute, and the user may be tricked into confirming a different transaction than the one that the malicious code will attempt to submit to the web server. However, activation of the secure mode of Intel TXT will result in hashing the agent's code into PCR 18. We show in §V-C that this will be detected by the web server, which will invalidate the transaction.

In case (2) neither the confirmation agent nor the isolated execution mode are invoked. Instead, an application in the untrusted mode simulates the confirmation dialog. The user is not able to detect a faked dialog since we do not have a full trusted path with the ability to authenticate the currently

running application to the user.[6] However, even if the user confirms the faked confirmation dialog, the malicious code is unable to cause PCR 18 to reset and be extended with the hash of the confirmation agent expected by the server. Again, §V-C shows how the web server detects this.

C. Remote Attestation

As our adversary has superuser privileges on the untrusted OS, he may be able to invoke an isolated execution environment with code and inputs of his choosing. Or, he may fake the entire process from the perspective of the user. In either case, our defense rests upon the contents of the PCRs in the client system's TPM. Specifically, PCRs 17–19 contain (respectively) hashes of the chipset-specific signed code from the chipset vendor (e.g., Intel), the hash of the confirmation agent code itself, and hashes summarizing the nonce, confirmation string that was displayed to the user, and the user's response.

PCR 17 can only be extended by the CPU or Intel-signed SINIT Authenticated Code Module. Thus, the remote server can learn precisely what code was invoked in the isolated environment and is able to detect unintended code execution when it verifies the values during the attestation process. This gives the web server the ability to detect and deny transactions that originate on hardware with known chipset or CPU vulnerabilities. Perhaps more importantly, it also enables the web server to detect that no such operation took place, i.e., that the isolated environment was *not* invoked.

PCR 18 is extended as part of the dynamic root of trust operation that invokes the isolated execution environment (e.g., GETSEC[SENTER]). It will contain the hash of precisely the code that received control following the reset of the dynamic PCRs (including 17–19) and platform reinitialization by the Authenticated Code Module. On a well-behaved system, this will be the hash of the UTP confirmation agent code. Again, the web server can verify the presence of known-good code. The presence of other code, or the absence of any measurement, signify an attack.

PCR 19 is used to summarize the anti-replay nonce provided by the web server prior to confirmation, the confirmation message itself, and the user's input in response to the confirmation message. The user's input must arrive via the keyboard driver in the isolated execution environment. Thus, solely software-based attacks are prevented. An adversary's best course of action is to physically compromise the keyboard, which is a significantly more difficult attack to perpetrate without physical access to the system.

D. Denial of Service

Instead of manipulating a transaction confirmation, the adversary can block a pending confirmation by preventing the UTP code from running on the client. The server will not receive any confirmation and abort the transaction after a timeout. This is a denial-of-service (DoS) attack and outside the scope of this paper.

[6]This is where a "secret picture" (or similar) mechanism can be a great aid if users can be trained to exhibit sufficient diligence.

VI. Related Work

E-EMV [39] is a software-based credit card application to secure transaction confirmation, similar to our UTP agent. They also use an enrollment phase where an AIK of the user's TPM is certified by a Privacy CA or in a DAA group. However, they do not combine transaction details with an explicit user interaction. Users can still be tricked into authorizing malicious transactions. Our approach includes a transaction summary and user-entered text into the attestation. Moreover, their approach needs a persistently running security kernel and a full trusted path.

Phoolproof Phishing [40] seeks to strengthen user authentication to web sites by leveraging users' smartphones as a store for public key-based authentication credentials that serve as an additional factor for authentication. Dedicated hardware solutions for transaction confirmation have also been proposed. The IBM ZTIC [41] is a small USB device that is assumed to share cryptographic keys with the user's financial institution. It includes a screen to display transaction details to the user, and buttons for accepting or aborting the transaction. Our design is similar to both of these in that we also use public key-based authentication credentials stored in the system's TPM. However, our design has the advantage that the user is not required to carry around or manipulate multiple devices during a transaction.

Several wallet-based approaches [26], [27], [28], [29], [30] have been proposed to secure user authentication or protect login credentials. They either require a trustworthy external device as an out-of-band channel for authentication [28], [42], [43], or require a secure kernel [26], [27] that isolates the underlying operating system and its complex software stack from the wallet. Many existing authentication agent proposals are prototyped using off-the-shelf virtual machine monitors, which are not without their share of vulnerabilities, e.g., [44], [45], [46]. In contrast, TruWalletM [30] uses M-Shield [47], a hardware-provided secure execution mode on a mobile phone, to protect login credentials of a user. The wallet is only invoked during login and executed in isolation in the secure mode. Although some use cases of UTP require authentication, this is orthogonal to our work. It should be a moderate effort to port their authentication agent code to Flicker and, hence, integrate them for the authentication step in our design.

SpyBlock [27] is a browser extension that requests authentication and confirmation of transactions from the user in a separate confirmation agent. However, the confirmation agent relies on a trusted window (full trusted path) and a secure hypervisor platform. In contrast, in our design we use the secure execution mode of the CPU to enable UTP

to execute exclusively during confirmation. Hence, the confirmation agent in our design depends on less trusted coded and is less disruptive to existing software environments.

The Not-a-Bot system employs an attester that certifies network traffic as originating in a system within some threshold time period after legitimate user input [48]. While an interesting design, the presented prototype includes a hypervisor and additional operating systems and thus includes a large TCB. Further, the binding between traffic and user input is temporal only – malware can readily wait and send all of its traffic only while the user types.

Grawrock describes the concept of a *Verification Model* possible on platforms that include support for dynamic root of trust [49]. The system that we describe in this paper can be considered a concrete realization of these abstract concepts, complete with evaluation and several use cases.

Dynamic root of trust on x86 systems is not the only mechanism available today for isolated execution. For example, ARM TrustZone [50] and Texas Instruments' M-Shield [47] offer similar functionalities, especially when paired with a Mobile Trusted Monitor. Nokia's OnBoard Credentials project explores some of the capabilities of these platforms [51]. The Cell Broadband Engine [52], used in Sony's PlayStation 3, offers eight processing cores that can be operated in a secure execution mode, isolated from each other and from the main core that runs the OS. We believe that our proposed solution should be portable to those platform architectures with manageable effort.

VII. Conclusion

We show how the combination of an on-demand isolated execution environment and temporal control of user-centric I/O devices enables construction of a mechanism for a one-way trusted path, and show that it is practical and deployable on commodity systems today. The uni-directional trusted path constructed by our system extends from the client system to the remote server. Although the immediate feedback available to the client user remains susceptible to manipulation by malware, our system enables service providers to gain additional assurance that client transactions are initiated at the behest of a human user on the computer, and not via malware such as transaction generators. Over the long term, users will be less likely to become the victims of a scam. Hence, service providers have high incentives for deployment, as they can simply offer UTP as a download for their users.

VIII. Acknowledgments

We wish to thank Ahren Studer, Jim Newsome, and the anonymous reviewers for their valuable comments, which greatly improved the paper.

References

[1] CNET, "Pop-up program reads keystrokes, steals passwords," 2004, http://news.cnet.com/2100-7349_3-5251981.html.
[2] ——, "New Trojans plunder bank accounts," 2006, http://news.cnet.com/2100-7349_3-6041173.html.
[3] W3C Working Group, "Inaccessibility of CAPTCHA," Note 23, Nov. 2005.
[4] J. Epstein, J. McHugh, H. Orman, R. Pascale, A. Marmor-Squires, B. Danner, C. R. Martin, M. Branstad, G. Benson, and D. Rothnie, "A high assurance window system prototype," *Journal of Computer Security*, vol. 2, no. 2, 1993.
[5] N. Feske and C. Helmuth, "A Nitpicker's guide to a minimal-complexity secure GUI," in *Proc. Annual Computer Security Applications Conference (ACSAC)*, 2005.
[6] J. S. Shapiro, J. Vanderburgh, and E. Northup, "Design of the EROS trusted window system," in *Proc. USENIX Security*, 2004.
[7] TrustMark, "TrustMark banking and financial solutions," http://www.trustmark.com/passmark/index.html, Aug. 2010.
[8] B. Lampson, "Usable security: How to get it," *Communications of the ACM*, vol. 52, no. 11, 2009.
[9] AMD, "AMD64 virtualization codenamed "Pacifica" technology — secure virtual machine architecture reference manual," AMD, Tech. Rep. Publication Number 33047, Revision 3.01, May 2005.
[10] Intel Corporation, "Intel trusted execution technology MLE developer's guide," Intel Corporation, Tech. Rep. Document Number: 315168-006, Dec. 2009.
[11] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 2001.
[12] L. F. Cranor, "What do they indicate? evaluating security and privacy indicators," *Interactions*, vol. 13, no. 3, 2006.
[13] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer, "The emperor's new security indicators," in *Proc. IEEE Symposium on Security and Privacy*, 2007.
[14] M. Wu, R. C. Miller, and S. L. Garfinkel, "Do security toolbars actually prevent phishing attacks?" in *Proc. Conference on Human Factors in Computing Systems (CHI)*, 2006.
[15] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for TCB minimization," in *Proc. ACM SIGOPS European Conference on Computer Systems (EuroSys)*, 2008.
[16] Trusted Computing Group, "TPM main specification, version 1.2, revision 103," Jul. 2007.
[17] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: a virtual machine-based platform for trusted computing," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 2003.
[18] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. van Doorn, J. L. Griffin, and S. Berger, "sHype: Secure hypervisor approach to trusted virtualized systems," IBM Research Division, Tech. Rep. RC23511, Feb. 2005.
[19] European Multilaterally Secure Computing Base Project, "Turaya," http://www.emscb.org/content/pages/turaya.htm.
[20] OpenTC Project Consortium, "Open Trusted Computing," http://www.opentc.net.
[21] K.-P. Yee, D. Wagner, M. Hearst, and S. M. Bellovin, "Prerendered user interfaces for higher-assurance electronic voting," in *Proc. USENIX Electronic Voting Technology Workshop*, 2006.
[22] C. Doctorow, "Solving and creating captchas with free porn," Boing Boing, Jan. 2004.
[23] S. M. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks," in *Proc. IEEE Symposium on Security and Privacy*, 1992.

[24] D. P. Jablon, "Strong password-only authenticated key exchange," *Computer Communication Review*, vol. 26, 1996.
[25] T. Wu, "The secure remote password protocol," in *Proc. Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 1998.
[26] S. Gajek, A.-R. Sadeghi, C. Stüble, and M. Winandy, "Compartmented security for browsers – or how to thwart a phisher with trusted computing," in *Proc. Conference on Availability, Reliability and Security (ARES)*, 2007.
[27] C. Jackson, D. Boneh, and J. Mitchell, "Spyware resistant web authentication using virtual machines," http://crypto.stanford.edu/spyblock/, 2006.
[28] R. C. Jammalamadaka, T. W. van der Horst, S. Mehrotra, K. E. Seamons, and N. Venkasubramanian, "Delegate: A proxy based architecture for secure website access from an untrusted machine," in *Proc. Annual Computer Security Applications Conference (ACSAC)*, 2006.
[29] M. Wu, R. C. Miller, and G. Little, "Web Wallet: Preventing Phishing Attacks by Revealing User Intentions," in *Proc. Symposium on Usable Privacy and Security (SOUPS)*, 2006.
[30] S. Bugiel, A. Dmitrienko, K. Kostiainen, A.-R. Sadeghi, and M. Winandy, "TruWalletM: Secure web authentication on mobile platforms," in *Proc. Trusted Systems, Second International Conference (INTRUST)*, 2010.
[31] OASIS, "Assertions and protocols for the OASIS security assertion markup language (SAML) v2.0," OASIS Standard saml-core-2.0-os, Mar. 2005.
[32] OpenID Foundation, "OpenID authentication 2.0 - final specification," Dec. 2007.
[33] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2004.
[34] E. Cesena, H. Löhr, G. Ramunno, A.-R. Sadeghi, and D. Vernizzi, "Anonymous authentication with TLS and DAA," in *Proc. Conference on Trust and Trustworthy Computing (TRUST)*, 2010.
[35] FreeWebShop.org, http://www.freewebshop.org.
[36] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the art of virtualization," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 2003.
[37] K. Chen, "Reversing and exploiting an Apple firmware update," in *Black Hat*, 2009.
[38] M. AlZomai, B. AlFayyadh, A. Jøsang, and A. McCullagh, "An experimental investigation of the usability of transaction authorization in online bank security systems," in *Proc. Australasian Information Security Conference (ACSC)*, 2008.
[39] S. Balfe and K. G. Paterson, "e-EMV: Emulating EMV for internet payments with trusted computing technologies," in *Proc. ACM Workshop on Scalable Trusted Computing (STC)*, 2008.
[40] B. Parno, C. Kuo, and A. Perrig, "Phoolproof phishing prevention," in *Proc. Financial Cryptography and Data Security Conference*, 2006.
[41] IBM Zurich Research Lab, "Security on a stick," Press release, Oct. 2008.
[42] A. Vapen, D. Byers, and N. Shahmehri, "2-clickAuth - optical challenge-response authentication," in *Proc. Conference on Availability, Reliability and Security (ARES)*, 2010.
[43] T. Weigold, T. Kramp, R. Hermann, F. Höring, P. Buhler, and M. Baentsch, "The Zurich Trusted Information Channel – an efficient defence against man-in-the-middle and malicious software attacks," in *Proc. Conference on Trust and Trustworthy Computing (TRUST)*, 2008.
[44] "Elevated privileges," CVE-2007-4993, 2007.
[45] "The CPU hardware emulation does not properly handle the Trap flag," CVE-2008-4915 (under review), 2008.
[46] R. Wojtczuk, "Subverting the Xen hypervisor," Invisible Things Lab, 2008.
[47] J. Azema and G. Fayad, "M-Shield mobile security technology: making wireless secure," Texas Instruments, Feb. 2008, http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf.
[48] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy, "Not-a-Bot (NAB): Improving Service Availability in the Face of Botnet Attacks," in *Proc. Network Systems Design and Implementation (NSDI)*, Apr. 2009.
[49] D. Grawrock, *Dynamics of a Trusted Platform: A Building Block Approach*. Intel Press, 2008.
[50] T. Alves and D. Felton, "TrustZone: Integrated hardware and software security," *Information Quaterly*, vol. 3, no. 4, 2004.
[51] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala, "On-board credentials with open provisioning," in *Proc. Symposium on Information, Computer, and Communications Security (ASIACCS)*, 2009.
[52] K. Shimizu, "The Cell Broadband Engine processor security architecture," http://www.ibm.com/developerworks/power/library/pa-cellsecurity/, Apr 2006.

APPENDIX

The Trusted Platform Module is an inexpensive, passive device that can serve as a hardware root of trust [16]. TPMs include a bank of Platform Configuration Registers (PCRs) that can be *extended* with *measurements* of code or data. A measurement $m \leftarrow SHA1(data)$ represents a cryptographic hash over an object of interest. An extend operation is defined as $PCR_{new} \leftarrow SHA1(PCR_{old}||m)$.

After measurements are accumulated into a hash chain inside one or more PCRs, *sealing* and *attestation* become possible. In a seal operation, a cryptographic key is access-controlled based on the values of one or more PCRs. With an appropriate software architecture, this can render data available only to certain software stacks. In an attestation, a digitally signed PCR aggregate called a *Quote* is produced. The asymmetric key used to sign the Quote is called an Attestation Identity Key (AIK), which can be linked to a particular TPM instance using the Endorsement Key (EK) certificate provided by reputable TPM vendors.

Newer x86 platforms (circa 2007) equipped with a v1.2 TPM and certain CPU and chipset enhancements can also create a *dynamic* root of trust for measurement (DRTM). A DRTM operation (1) reinitializes the CPU and memory controller into a known-good state and (2) resets *dynamic* PCRs inside the TPM into a distinguished state, enabling the possibility to bootstrap isolated execution with data sealing and attestation capabilities even in the face of a compromised OS or virtual machine monitor (VMM).