Implementing Attestable Kiosks

Matthew Bernhard† Gabe Stocco⋆ J. Alex Halderman†
† University of Michigan ⋆ Microsoft
{matber, jhalderm}@umich.edu gastocco@microsoft.com

***Abstract*—In this paper we explore the notion of a secure kiosk, a trusted computing platform built using off-the-shelf components. We demonstrate how kiosks serve as convenient primitives when designing secure computing protocols, as they allow for a very prescribed set of assumptions to be made about a system. We begin by defining the necessary properties of a kiosk, and then explain how each of these properties can (or cannot) be attained using current off-the-shelf hardware and software components. We construct a proof-of-concept implementation using TPM hardware and Windows 10. We also provide ASKVote, the Attestable and Secure Voting protocol to demonstrate the flexibility gained from the use of kiosks in a larger secure system.**

I. Introduction

Falling hardware costs and ever-increasing connectedness mean that electronic devices have become ubiquitous in everyday life. The need to carry out secure computations on trusted machines is at an all-time high. Furthermore, much of computing relies on multiple devices, meaning that users who wish to perform secure computations often want to do it across a network. There is an ever-burgeoning need for trusted machines, or kiosks, that can allow users to carry out financial transactions, communicate securely, participate in elections, or take part in other secure computations across devices [12]. Complicating matters, many applications that require secure kiosks are wide-spread, meaning the kiosk platform cannot be expensive or difficult to obtain. Applications have also already been established using existing, less-secure techniques, and these systems cannot be wholly replaced. Kiosk platforms must therefore be established using relatively cheap or already in-place systems, and secure computation protocols and applications must be developed with this in mind.

The kiosk platform presented here is a trusted computing platform built from off-the-shelf, consumer grade hardware running standard software that can be trusted across a network. We explore how this platform provides a useful framework for reasoning about usable security design and serves as an elegant primitive when designing security protocols. Until recently, there has been little in the way of a cohesive set of tools and techniques to enable the development of this kind of platform. Moreover, security features, like TPM and software support for its use have not been widely available until very recently, and this has hindered the development of more open software platforms for applications which require a very high level of system security [25]. Finally, naive approaches to developing kiosks often get ensnared in problems with bootstrapping trust from the hardware up to the software, and kiosk solutions must rely on techniques like attestation with a remote party to circumvent this [10].

For our purposes, the only necessary features to build a kiosk are a root-of-trust in hardware (via TPM 2.0), and software and system mechanisms for attestation. Many spaces where kiosks are needed have tightly constrained budgets, for instance, publicly funded county clerks who are tasked with purchasing voting machines. As long as the hardware meets our requirements (and whatever additional components the kiosk needs for its intended purpose) any machine can be used. Our kiosk primitive will allow a broad base of currently available computers to be turned into attestably secure machines, allowing any software running on them to establish a root of trust all the way down to the hardware. By securing a wide range of off-the-shelf hardware, entities like county clerks can reduce cost by buying market-priced personal computers, but still get the desired security properties a customized system would deliver.

We present a way in which consumer grade computers meeting the above requirements can be turned into a secure kiosk using the Windows 10 platform. While this discussion largely focuses on the specifics of kiosk implementation in Windows, the underpinnings of the approach are generalizable to all platforms. We construct kiosks by providing verifiable data about the machine to an outside trusted entity (the administrator) as a means of attesting to the valid state of the kiosk's hardware and software. Through this attestation process, we can provide verification that the machine booted in a valid state, has not since been restarted, and is currently enforcing device and software policies as mandated by an administrator. This allows administrators and trusted entities to not only verify that a machine is in an expected state, but also to take action in the event that it is not, for example removing the machine from its network or alerting an administrator.

II. Goals for a Kiosk Platform

A. Threats

Prior to establishing the goals for our kiosk platform, we will outline our threat model. We maintain a relatively broad set of adversaries, as our goal is to mitigate any use-case outside the specified parameters of the system. It is assumed adversaries will not modify the hardware of the kiosk, e.g. removing a TPM device, disconnecting the kiosk from a network, or otherwise damaging the machine in such a way that it can no longer function. Outside of this, adversaries will have the ability to leverage whatever services the kiosk offers (potentially including Internet connectivity and external device support).

B. Goals

We will define a kiosk as a trusted computing device built from off-the-shelf components that restricts user control over and access to software and software configuration. Specifically, an arbitrary user cannot modify any part of the machine such that the machine can enter an unexpected state or exhibit behavior that violates the kiosk's intended use. For instance, if the kiosk is acting as an ATM, a user should not be able to alter the software of the ATM to reveal other users' credit card numbers and PINs. To do this, a user would have to modify the machine to collect the data in some way and then report it, either to a local storage device or across a network. While the kiosk should protect against this kind of tampering, it should also have the facilities to report its state in the event that it is tampered with. This reporting of state is *attestation*, and it allows a third party to verify that the kiosk is behaving as expected and take action based on information about the kiosk's state.

We also desire to create an environment that allows for design based in secure computation and protocols. One of the greatest difficulties in computer security applications is implementation, so ease-of-use in development is a very important goal. Not only will developers be able to design around a trusted platform, but the trust gained by our methods should not be defeated by bugs and misuse.

In order to achieve these somewhat nebulous goals, our easy-to-use and robust trusted computing platform will revolve around policy enforcement. Much work has been done regarding this at a systems level: mandatory and role-based access control, firewalls, code integrity, and other policies have been proposed and implemented [14]. We seek modular policy enforcement so that which policies are enforced is entirely up to the protocol designer to fit the needs of the system. For instance, e-commerce kiosks may wish to use external credential inputs, like a smart card, for authenticating the user, so a smart card reading device would need to be allowed by the policy. On the other hand, an electronic voting machine may not need the capability to read smart cards, and this may in fact open up a vulnerability in the voting system [5]. Networking constraints may call for a highly configured firewall to be used. Certain applications may require the use of a web browser, or Wi-Fi functionality, while others will not.

In short, our goal is to build a machine that:

1) Protects against software and configuration modification by unprivileged users
2) Prevents access to restricted resources and information by unprivileged users
3) Attests to the system's state at arbitrary times so that malicious parties cannot know when the state kiosk is being inspected
4) Performs trusted computations in a networked environment
5) Allows for easy development of kiosk software applications
6) Enables highly configurable policy enforcement

In order to obtain these properties, we will rely on the Windows 10 operating system and external features that it utilizes to provide a trusted computing environment.

III. Background and Related Work

A. TPM

The Trusted Computing Group's Trusted Platform Module (TPM) is a foundational building block of most secure computing schemes. A TPM is a coprocessor that ties cryptographic keys to hardware, so that a machine can be uniquely identified in a trusted manner. TPMs also allow for remote attestation of system state. The way TPM is utilized in Windows reflects the TCG Software Stack specification, so we do not lose generality in our discussion of the functionality exposed by the features discussed below [19]. Windows 7 was the first version of Windows to use TPM as a means of attesting to system startup state for features like BitLocker (hard disk encryption), providing secure key storage, and other cryptographic functionality. On top of system APIs that accessed TPM, a TPM Platform Crypto-Provider (TPM PCP) was built to give developers easier access to TPM functionality and to allow services to use this functionality for attestation purposes [21]. This provider implements a Key Storage Provider (KSP) that is used by Windows for key management and storage on the TPM, and consequently the KSP forms a fundamental basis for our discussion of security and attestability in Windows 10.

The KSP utilizes the Endorsement Key (EK), which is built in to the TPM by the manufacturer along with a certificate. The EK is used to establish pseudo-anonymous Attestation Identity Keys (AIKs), which can be used by a third-party certificate authority to issue an AIK certificate. Given that a verifier trusts the issuer of the AIK certificate, it can challenge the TPM to prove possession of the AIK backing the certificate. This process allows the device to prove consistent identity over time, without exposing the same identifier to every service provider. In addition to the EK, the TPM's secure Platform Configuration Registers (PCRs) contain pertinent data about the system's state, for example which binaries have been loaded. These registers cannot be reset without a boot cycle, forming an append-only log of the system state.[1]

B. Related Kiosk Approaches

Secure kiosks have long been sought after. Many of the problems standing in the way of attaining this goal focused on bootstrapping trust from the hardware (the TPM) up through the operating system and into the user space. Without establishing a root of trust in hardware, systems are vulnerable to the cuckoo attack, in which adversaries impersonate the local TPM remotely and fake out any verification [4], [10]. Because there is no trust between the TPM, the hardware, and the software, a malicious attacker can intercept the software's requests to the TPM to validate system state and replace the on-board TPM's responses with the responses of a different TPM. Even if the attacker breaks the machine in a way that the on-board TPM would detect, the replacement TPM will still assert that the system is in a good state, fooling the software

[1] Specifically, the registers are extended by hashing new data together with the previous value in the register, forming a trust chain.

and thus totally defeating the purpose. This attack is a central problem to kiosks, as it prevents the establishment of trust even within one device, let alone trust in a remote device.

Solutions to this problem include a Seeing-is-Believing approach in which users enter data physically located on a computer (e.g., a barcode) about the kiosk into a separate channel (like a smartphone) and verify the state of the machine [8], [10]. A related approach involves a process similar to two-factor authentication, in which users pass a token with a smartphone via near-field communication (NFC) to establish an authenticated channel for verification [22].These approaches present difficulties in that they are not easily applicable to currently deployed systems, introduce a high risk of user error, and require the use of a smartphone, which for some kiosk use-cases is not tolerable (for example, voting machines). Alternative solutions such as a trusted BIOS, boot process, or trusted third-party verification, do not suffer these deficiencies [10]. Windows uses the AIK discussed above as proof of identity, and a TPM can prove the state of the machine by reporting the values in the PCRs which can then be verified remotely to check that they comply with expected values. This process is generally referred to as remote attestation. Further details regarding this process can be found in the TCG's standards [20].

Remote attestation is a key insight into the problem of bootstrapping trust, allowing a remote verification service to provide certification of the validity of the attesting platform's configuration [23]. Integrating remote attestation into the boot process through the Integrity Measurement Architecture (IMA) allowed for the establishment of a root of trust in hardware without vulnerability to the cuckoo attack [15]. Windows Secure Boot is a feature similar to IMA, and allows the operating system to establish a root of trust down to the hardware. However, problems persist with establishing an authenticated channel between the verifier and the user, and more generally gaining user trust to convince a user that a cuckoo attack has not occurred [9]. Through the use of a trusted certificate authority, as proposed in [10], we can establish an authenticated channel between the verifier and the user, and in fact this is precisely the architecture of Windows 10 Device Health Attestation.

IV. Implementation

A. Device Health Attestation

Device Health Attestation (DHA) in Windows 10 provides a framework for acquiring attestation information and using it to establish trust with a kiosk. Moreover, the Windows Health Attestation Service (HAS), a specific implementation of DHA, provides a PCR log parser which reports the status of machine parameters throughout the boot process. The overall topology of the DHA flow in Windows involves (shown in figure 1):

- the kiosk machine, whose state is being queried
- the administrative interface, which requires validation of the state of the kiosk before trusting it
- a certificate authority to issue AIK certificates
- an external service that provides validation that the attestation log of system state is genuine and also provides a parse of values taken from the log in XML format to the administrative interface.

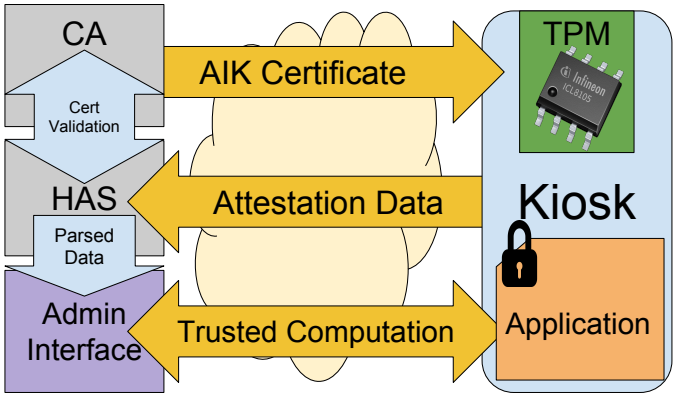

Fig. 1: The DHA topology

The external service is currently implemented as Windows HAS, a Microsoft-operated service. Microsoft additionally supplies a certificate authority for the issuance and validation of AIK certificates. In the event that a protocol has needs outside the scope of Windows HAS (for instance, it needs to be air-gapped), Windows Server 2016 provides developers with the ability to build a separate DHA service that replaces Windows HAS, referred to as an on-premises Health Service. On-premises Health Services can serve in place of Windows HAS for systems that cannot connect to the Microsoft operated service, and they can also fulfill other requirements that are not met by Windows HAS. For the sake of simplicity we will use the term HAS, with the understanding that an on-premises system may be substituted to fill the same role.

To validate the state of the kiosk remotely, the kiosk provides a signed token from the HAS to the administrative interface. This token is acquired in the first part of the health attestation protocol, wherein the kiosk sends its boot log and a quote of its PCRs to the HAS. The administrative interface presents the kiosk's token to the HAS, which, after validating the token's authenticity, reports relevant configuration information (PCR values, whether BitLocker is enabled, etc.) to the administrative interface. The administrative interface maintains a log of expected and valid configurations for the kiosk, and then checks the reported values against its expected values for the kiosk's state. This protocol relies on the AIK and AIK certificate that were established in the previous section to identify the kiosk, as well as standard secure communication and logging techniques to interface with the kiosk and the HAS and keep track of the kiosk's state. As the data is logged in append-only fashion, if the most recent attestation data is not fresh enough, the kiosk has not provided the proof that it is in a valid state, and should have its privileges revoked. Discussion about how acceptable freshness is established and remediation procedures will be described later in sections IV-D and V.

From this point forward, when we say a kiosk "attests" to itself in the context of HAS, we mean that the kiosk is providing a quote of its state to the HAS that proves it is in the same state as it was when it acquired the token. The

service is entirely abstracted from the user, or in our case, the developer who wishes to develop kiosk software. The administrative interface runs as a server communicating with the HAS, and kiosks enroll with the administrative interface as clients of the service over the network. Enrollment relies on credentials that have previously been established by the administrator and distributed to the enrolling device.

As part of enrollment, an attestation interval is established, meaning that the kiosk must attest to its state at regular intervals of time. If, during the attestation process, the state of the machine has changed so as to indicate it is operating in an undesired or undefined way, action can be taken at this point. Regular attestation, however, presents a problem to our attestable kiosk model on two fronts. First, if one were to design a protocol that relied on attestation, not having control over when a machine attests would cause significant difficulty, particularly if the protocol had critical sections that did not align with the regular intervals in which a machine must be in a valid state. Second, if a machine only ever attests to itself over regular intervals of time, an attacker can predict attestation events and clean up the state of the machine to appear normal. HAS was not developed with secure kiosks in mind, and regular heartbeats make sense in a lot of contexts like low-reliability networks or mobile device platforms. In order to develop secure kiosks with HAS, we cannot restrict ourselves to regular intervals. Fortunately, the HAS also exposes methods that will prepare a system for attestation and send an attestation log, and leveraging this API as described in V, we can get the granularity and control over attestation events that we need.

B. The WMI Bridge

As part of Windows' built-in Device Management services, the Windows Management Instrumentation (WMI) enables system information retrieval and function call invocation, both locally and remotely (though not all exposed APIs are available remotely). For instance, WMI calls can present information about processor architecture or operating system version, data which can assist in making informed decisions about remote, untrusted machines. WMI also has extensive exposure in the .NET Framework, meaning that a Windows application written in the framework can access and invoke WMI methods and information. Services and system calls are exposed via WMI Providers, and Windows DHA has a WMI provider that mediates between SyncML messages and WMI calls called the WMI Bridge.[2] Also provided by the Bridge is access to policy information about enabled devices and running software on the target machine. A sampling of the configuration information available includes the status of devices like Bluetooth, cameras, and Wi-Fi, and information about running programs like Windows Defender, browser configuration, and whether the OS is configured to allow untrusted applications to run.[3] Essentially, via the WMI Bridge, we gain increased ability to enforce application and configuration policy in addition to gaining the ability to programmatically send attestation data from the kiosk to HAS by calling `VerifyHealthMethod` in the WMI Bridge. While there are not explicit system controls to take action based on policy enforcement, if we have the information about what a kiosk is doing, we can know if it is violating a policy, and programmatically restrict that machine's privileges on a network. For instance, if our kiosk is an ATM, and we see that Bluetooth has been enabled on that machine and a hypothetical attacker could manipulate the machine with Bluetooth access, we could force the ATM software (that interfaces alongside the HAS with an administrative interface) into a fail-safe mode until a technician inspects the machine and restores it to a secure state.

C. On-Demand Attestation

Now that we have a way of programmatically sending attestation data, it is necessary to turn our attention to our goal of randomly attesting to a systems's state. Ideally, we would have an operating system-level control that would allow an administrative interface to trigger an attestation event on a remote machine. Networked computers with the correct permissions and settings can complete WMI calls across the network. As in our previous example, if an administrative machine was connected to the ATM, a remote WMI call could shut down the kiosk altogether. However, not all WMI calls can be performed across a network, as an invoked WMI call that targets a remote machine cannot cross the boundary between Windows' User and System processes. Specifically, if a process on our trusted machine attempts to invoke an attestation event on our kiosk, the call is formed in the User process on the trusted machine, but the call must be carried out as a System process on the kiosk, a boundary that cannot be crossed. Because of this, it is not currently possible to remotely invoke a health attestation event through a Windows system call. Despite this, protocols can be designed such that they preserve unpredictable attestation invocation.

If we restrict our WMI calls to the local kiosk machine, we can programmatically invoke an attestation event, which will look exactly like the periodic attestation events to the service, effectively rendering health attestations on-demand. All attestation-based security protocols implemented in Windows 10 must then have an API call that allows for an administrative interface to cause a kiosk to submit its attestation data. The software on the kiosk must include the `VerifyHealthMethod` call to trigger an attestation event. When the administrative interface wishes to check the state of the kiosk, it can send a message over the network. If the kiosk does not respond with a valid attestation, then the administrative interface can assume that the kiosk is in a bad state. The administrative interface must randomly ask all kiosks to attest, but the randomness can be parameterized by the level of trust in a system. For instance, in a highly untrusted system, with a kiosk with a misconfigured firewall, it could be the case that the administrative interface requests that a kiosk to attest to itself at a random time such that no more than

[2]For more information, see https://msdn.microsoft.com/en-us/library/dn934876(v=vs.85).aspx

[3]For a more extensive list, see https://msdn.microsoft.com/en-us/library/windows/desktop/dn905224(v=vs.85).aspx

two minutes pass between attestations. This interval could be larger or smaller depending on the trust in the system, or it could be truly random, with the assumption that the regular-interval attestation events that are part of DHA will guarantee that *some* attestation occurs during the protocol. This satisfies our goal of arbitrarily attesting to the kiosk's state.

D. Policy Enforcement

At this point, we now have all the requisite pieces for building a kiosk system through remote attestation. However, all that has been provided by the various functions of the operating system is information about the kiosk. A kiosk is the result of enforcement of policies based on that information, not just monitoring. Applying the Baconian "knowledge is power," we can now design protocols that take action based on this information, and therefore enforce security policies on the kiosk, locking down whatever features we do not want and allowing those we do. This accomplishes our goal of finding ways to enforce policy in a modular fashion, and by focusing on properties rather than disentangling each mechanism as it is implemented we also help satisfy our goal of ease-of-use [13].

The following is a list of some of the policies that are configurable via the WMI Bridge,[4] along with discussion of how each policy functions in creating a kiosk. Discussion of how to remediate policy violations follows afterwards.

- Account Policy — A kiosk should only ever have one non-administrator account through which users will interface. The user account should be restricted to the local system. Ideally, it should be entirely anonymous, or even invisible, as the kiosk will never have a user log in to a local system account. Other user authentication procedures may be present in the function of the kiosk, but not at the operating system level.
- Application Management Policy — Kiosks should have a restricted set of applications that can run, with minimal outside communication necessary for the applications to run. Additionally, application data should be restricted to the kiosk, and should not be stored or shared anywhere else.
- User Data Protection Policy — Any data tracking that would be otherwise sent out to a third-party service, like that automatically configured in Windows 10, should be turned off.
- Anti-Malware Policy — Users should not be able to interface with the anti-malware program at all, but it should still be running.
- Screen Lock Policy — The screen of the device should never turn off, or it should never lock the device such that a login is required.
- System Information Policy — Various system information reporting, such as device location, should be disabled
- External Device Policy — No unapproved external devices should be allowed to connect to the machine, either physically or wirelessly, for instance USB devices, SD cards, or Bluetooth devices.
- System Configuration Policy — All non-essential system modes, like kernel debugging, safe mode, etc., should be disabled.

Many of these features are natively handled by the HAS, and will be flagged if their values are unexpected. In addition, other system configuration data is reported, like BitLocker, the presence of an AIK certificate, whether the machine is running safe mode, if the early-launch anti-malware (ELAM) driver is loaded properly, boot and kernel debugging settings, as well as hashes of the Code Integrity policy, Data Execution Prevention policy, and the value in PCR[0] of the TPM. All of these data points are critical to ensuring a secure state for the Windows operating system and therefore the kiosk itself, and any deviation from the expected values break the root of trust for operation of the kiosk.

As for enforcement across the network, the first and most obvious way of a protocol enforcing security policy is blacklisting. If a machine is misbehaving, simply cut off its interaction with the rest of the machines on the network, either by shutting it down (if possible), black holing its network traffic, or otherwise notifying the network to ignore what it says. However, if a machine is malfunctioning for non-malicious reasons, there may be shades of grey between the blacklist of bad machines and the whitelist of appropriate behavior. For instance, if a machine's anti-malware driver is out of date, but the differences between the most up-to-date version and the version on the kiosk are minimal, it may be desirable to restrict the kiosk's interactions with the rest of the system, or monitor it more closely (by performing more frequent attestations) than to blacklist it. In scenarios like a voting precinct on election day, having the extra voting machine up, even if in a restricted state, could have a significant impact on things like queue length and throughput. Having a flexible policy framework gives the power to ratchet up security as high as it will go, but it also allows us to compensate for application-specific constraints.

A critical takeaway here is that we can integrate policy enforcement into a protocol, ensuring that actions taken for malfunctioning or misbehaving kiosks are well defined and rigorous enough to ensure the security of our software system as a whole. Again, this will be entirely dependent of the scenario, but as we have already seen, secure kiosk protocol design is particularly well-suited to sensitive transactions like authentication, verification, and secure data transfer. To provide a more concrete example of the power gained by integrating attestable kiosks into security protocols, we present the following use-case.

V. Voting Systems: A Kiosk Use-Case

Voting systems present an almost ideal use-case for secure kiosks. There is a high standard for security, highly constrained cost considerations, and the specific procedure of voting involves a computer executing a very small, well defined set of tasks. As mentioned before, voting machines are not the only case for kiosk use, and any scenario involving a very limited set of functionality and strict security constraints presents a good candidate for the use of a secure kiosk machine. In order

[4]For a more extensive list, see https://msdn.microsoft.com/en-us/library/windows/desktop/dn905224(v=vs.85).aspx

to better illustrate how one practically designs a secure kiosk, we present ASKVote, the Attestable and Secure Kiosk voting protocol. ASKVote is merely a protocol, designed to fit in with a larger secure voting scheme, and to work in conjunction with other verifiable voting schemes and systems like STAR-Vote [3], Helios [1], and others. As mentioned above, the flexibility of off-the shelf hardware and software means that a kiosk protocol can be designed independently of the specifics of electronic voting schemes, and it is not dependent on the cryptographic or procedural mechanisms of those schemes.[5]

The ASKVote primitives, as described above in our discussion of DHA, are a trusted machine running an administrative interface, one or more kiosk machines running the voting interface, and an on-premises Health Service running on another machine on the network.[6] The machines are connected via a network, either through standard mechanisms like TCP, or through more exotic protocols like Auditorium [16]. Other necessary machines may be present on this network, such as other administrative interfaces, auditing devices, ballot reading devices, or any other machines required by an election protocol.[7] As the ASKVote kiosk attestation protocol is entirely encapsulated in the connection between the administrative interface, kiosk, and on-premises Health Service, other machines are extraneous and will not be considered here.

The ASKVote protocol here presented requires that ballot information not be located on the kiosk machine; instead, it will be distributed by the administrative interface when the kiosk is ready to vote. By allowing this constraint, we can show how attestation can be integrated into the secure functions of protocols. It should be noted that this is not required for an attestable voting protocol to be secure. The attestation of the kiosk can still provide a means of further hardening a protocol that does not integrate attestation events by providing extra data to be validated by an audit process, or by simply providing control of the device based on its state. As other systems already exhibit the behavior of keeping ballot data separate from the kiosk (except when in use), we feel that it is a reasonable constraint on the scenario and a useful example of integrating secure kiosk attestation with another security system.

A. The Protocol

The ASKVote protocol can be broken into three major phases, the pre-election phase, the election phase, and the post-election phase.

1) Pre-election: In the pre-election phase, the kiosks and administrative interface are set up, configured, and networked. Once connected, the administrative interface starts an instance of the on-premises Health Service server, and the kiosk machine enrolls in the service. Now enrolled, the administrative interface will be able to collect attestation data from the kiosk throughout the duration of the election, thus enabling policy decisions to be made on the fly. Any other pre-election setup can occur as specified by the voting protocol.

2) Election: Once the election has started, a voting session takes the following shape:

1) A voter approaches the kiosk and initiates the voting process. This may or may not include the entering of an authentication code to specify what kind of ballot the voter will vote on. This is left up to the voting protocol, and discussion here is merely to paint a clearer picture of what goes on.
2) The kiosk sends the administrative interface the voter's identifier (if there is one), and also sends an attestation record to prove that the kiosk is in a good state and can be permitted to submit a ballot.
3) The administrative interface examines the attestation data parsed by the Health Service, and determines if the kiosk is indeed in a good state. The attestation data, as well as the administrative interface's determination, are both logged via a cryptographically secure logging mechanism.
4) If the kiosk's state is good, the administrative interface sends the ballot data to the kiosk.
5) The voter will begin making selections and marking the ballot. Meanwhile, the kiosk will send attestation quotes to the administrative interface at regular intervals, as well as responses to the random requests of the administrative interface as discussed above. The administrative interface will be expecting a quote at each interval, and if it does not receive a quote as expected, or if the kiosk does not respond to a requested attestation event, the administrative interface can assume that the kiosk is acting maliciously. Note that the expected interval may be a range to account for things like network latency. If the data is not received within the interval, the administrative interface will alert an election official and necessary steps will be taken, as specified by election protocol and the specifics of the voting scheme. For instance, the ballot marked in this session may not be counted, or the machine may be taken offline.
6) When finished marking the ballot, the voter will commit the ballot. The ballot will be encrypted (using a key distributed to the kiosk during the setup phase of the election). The kiosk will send the encrypted ballot data alongside an attestation quote to allow the administrative interface to determine if the completed ballot data can be accepted.[8] If the machine is still in a good state, the encrypted completed ballot data will be stored by the administrative interface. Finally, another attestation record to ensure nothing changed while committing the ballot immediately afterwards, and if the kiosk is still

[5]In the particular case of verifiable voting schemes, user trust is already strengthened by the verifiable nature of the scheme. However, ASKVote actually complements this strengthening by providing a secure platform for voting and by giving the verifiers more information to verify, since they can view the history of a machine's state throughout an election.

[6]Due to security concerns, voting systems are typically air-gapped. This means that we cannot rely on Windows HAS and will instead assume the use of an on-premises Health Service.

[7]It is incumbent on the protocol to restrict the machines on the network to exclusively the ones it needs, however.

[8]Accepted is a rather nebulous term, but generally we mean that the ballot should be considered committed as intended. If the election protocol uses a commit-challenge model, then the ballot is committed. Otherwise, the ballot may be considered cast at this time.

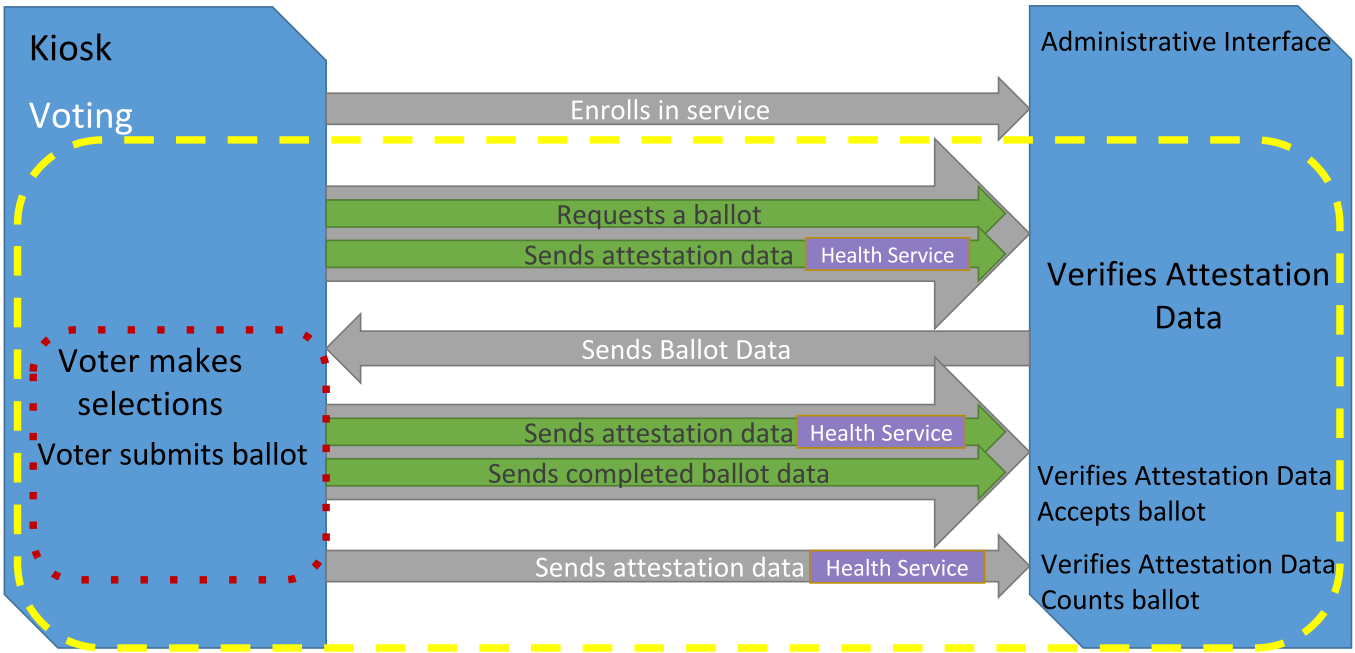

Fig. 2: The ASKVote Protocol

determined to be in a good state, the ballot will be accepted. All data in this transaction will be logged.

3) Post-Election: During the post-election phase, the election logs from the administrative interfaces may be examined to ensure that no kiosk submitted ballot data while in a state not approved by the protocol. If there are unexpected values in the logs, the individual kiosk can be checked for tampering and decisions can be made about what to do with its completed ballot data. These logs can be published, provided the ballot data is anonymous, so that third parties can verify the state of the kiosks was in fact normal during the election, in addition to other verification mechanisms that are out of scope here. In this way ASKVote serves to complement and strengthen auditability mechanisms that are currently proposed by working in conjunction with them and providing more evidence to the validity of an election.

VI. Evaluation

While we have made progress towards creating the first kiosk, there are several extensions and improvements that can be made, and we examine them here. As aforementioned, an attest-on-demand mechanism that is triggered remotely at the system level rather than at the application level would be a significant improvement, as it would reduce the attack surface of the kiosk. For instance, if somehow an attacker could insert itself into the ASKVote protocol between the two final attestations without detection, this would more or less defeat the purpose of the protocol (granted, other mechanisms like a commit-challenge model serve to lessen this threat).

One particularly dubious portion of our proof-of-concept is that it is done in a relatively closed-source environment. The Windows system calls we rely on themselves should be examined in order assure against malicious or erroneous code in the security stack on which the kiosk relies. There is promising work towards this goal in system-level provenance in an open-source environment [2].

TPM-based attestation is itself somewhat limited in its current form in Windows. The HAS can attest to everything up to the application layer, and can even attest to the signer of the application (through Device Guard Code Integrity)[9], but not beyond. For a true kiosk, the executable of the application that is running (say, the voting software) should be verified in the same manner as the driver and OS checks at present, and the results should be stored. This would of course incur performance costs, but on a kiosk machine only running one application, it likely will not have a significant impact (however there is research to be done there, too).

VII. Future Work

Looking towards forwards, there are a few areas which inform and are informed by trusted computing. Additional attestation and data provenance techniques, like dynamic attestation [6] and finer-grained attestation controls like program annotation [17] show promise in addressing some of the shortcomings we have noted above. Integrating system provenance techniques that already rely on TPM into attestation protocols may also prove fruitful for hardening kiosks [2].

In regard to the overall picture of Device Management (and specifically Mobile Device Management, MDM), we feel that there is promising work in examining the security properties of large scale systems that are controlled by MDM platforms. Again, to fall back on the same example, voting systems in

[9]Device Guard is another feature that further solidifies the trusted computing platform provided by Windows 10, and can extend the root of trust from the hardware into the application layer. However, it is beyond the scope of this paper.

large precincts involve hundreds if not thousands of voting kiosks, and having the ability to monitor and push fixes to software issues in a centralized fashion may provide an invaluable tool for election officials, likewise with ATMs, e-commerce kiosks, or others. If MDM software distribution can be used to apply our secure kiosk framework on any machine, perhaps an ephemeral "kiosk-ification" can be deployed to any user device so long as it has a TPM, allowing users to interact with secure systems from their own devices. Some work on trusted computing and online voting has already been done, but much of the infrastructure presented here was not yet widely available enough on end-user computers to make it a viable option [25]. Other work on locking down a system for security-sensitive transactions has been done in [24], in addition to work regarding the creation a red/green environment through virtualization [7], [11]. The obvious challenges to online voting (and generally to ephemeral "kiosk-ification") include a lack of coherence in application policy, which can prevent successful performance of secure operations on a wholly untrusted machine [18]. Privacy is also a concern, but it may be possible through sophisticated sandbox techniques and permissions to avoid putting users' personal data at risk when interacting with secure systems.

VIII. Conclusions

We have shown that an attestably secure and auditable kiosk can be created using existing Windows 10 features. In addition, we have developed the set of functionality appropriate to secure kiosk devices based on these features. Kiosks are powerful tools in spaces like financial transactions, elections, e-commerce, and secure communication, and the trusted computing platform provided by kiosks will only become more prevalent. We have explored how attestable kiosks provide a robust and usable environment for developers to create protocols based on hardened trust assumptions, and also assessed the greater flexibility in policy enforcement gained through kiosks. We have also provided an example of the use of a kiosk in our ASKVote protocol for increased voting security and auditability.

Acknowledgments

We would like to thank Dan Wallach and Josh Benaloh, whose advice and guidance greatly shaped and supported this project. We thank Ron Aigner, Eric Suen, Chris Fenner, and Kam Kouladjie for their help in piecing together various parts of this project.

References

[1] B. Adida. Helios: Web-based Open-audit Voting. In *Proceedings of the 17th USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[2] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer. Trustworthy Whole-System Provenance for the Linux Kernel. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 319–334, Washington, D.C., Aug. 2015. USENIX Association.

[3] J. Benaloh, M. Byrne, B. Eakin, P. Kortum, N. McBurnett, O. Pereira, P. Stark, D. Wallach, G. Fisher, J. Montoya, et al. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In *The USENIX Journal of Election Technology and Systems*, volume 1. USENIX, 2013.

[4] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Trustworthy and Personalized Computing on Public Kiosks. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, pages 199–210. ACM, 2008.

[5] H. Hrusti. Critical Security Issues with Diebold Optical Scan Design. http://www.blackboxvoting.org/BBVreport.pdf, 2005.

[6] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang. Remote Attestation to Dynamic System Properties: Towards Providing Complete System Integrity Evidence. In *IEEE/IFIP International Conference on Dependable Systems & Networks, 2009. DSN'09.*, pages 115–124. IEEE, 2009.

[7] B. Lampson. Usable Security: How To Get It. *Communications of the ACM*, 52(11):25–27, 2009.

[8] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-is-believing: Using Camera Phones for Human-verifiable authentication. In *Proceedings of the 26th IEEE Symposium on Security and Privacy*, pages 110–124. IEEE, 2005.

[9] J. M. McCune, A. Perrig, A. Seshadri, and L. van Doorn. Turtles All the Way Down: Research Challenges in User-based Attestation. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Security*, page 6. USENIX Association, 2007.

[10] B. Parno. Bootstrapping Trust in a Trusted Platform. In *Proceedings of the 3rd conference on Hot topics in security*. USENIX Association, 2008.

[11] M. Peinado, Y. Chen, P. Engl, and J. Manferdelli. NGSCB: A Trusted Open System. In *In Proceedings of 9th Australasian Conference on Information Security and Privacy ACISP*, 2004.

[12] A.-R. Sadeghi. Trusted Computing–Special Aspects and Challenges. In *Proceedings of the 34th International Conference on Current Trends in Theory and Practice of Computer Science*, pages 98–117. Springer, 2008.

[13] A.-R. Sadeghi and C. Stüble. Property-based Attestation for Computing Platforms: Caring About Properties, Not Mechanisms. In *13th Proceedings of the Workshop on New Security Paradigms*, pages 67–77. ACM, 2004.

[14] R. Sailer, T. Jaeger, X. Zhang, and L. Van Doorn. Attestation-based Policy Enforcement for Remote Access. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 308–317. ACM, 2004.

[15] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[16] D. Sandler and D. Wallach. Casting Votes in the Auditorium. In *Proceedings of the 2nd USENIX/ACCURATE Electronic Voting Technology Workshop (EVT07), Boston, MA*, 2007.

[17] E. Shi, A. Perrig, and L. Van Doorn. BIND: A Fine-grained Attestation Service for Secure Distributed Systems. In *Proceedings of the 26th IEEE Symposium on Security and Privacy*, pages 154–168. IEEE, 2005.

[18] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman. Security Analysis of the Estonian Internet Voting System. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, pages 703–715. ACM, 2014.

[19] TCG Software Stack (TSS) Specification. http://www.trustedcomputinggroup.org/files/resource_files/6479CD77-1D09-3519-AD89EAD1BC8C97F0/TSS_1_2_Errata_A-final.pdf. Version 1.2, Level 1. 7 March 2007.

[20] The Trusted Computing Group. http://www.trustedcomputinggroup.org/.

[21] S. Thom, P. England, J. Loeser, R. Spiger, R. Aigner, and M. Jim. Using the Windows 8 Platform Crypto Provider and Associated TPM Functionality. Technical report, Microsoft, 2012.

[22] R. Toegl. Tagging the Turtle: Local Attestation for Kiosk Computing. In *Proceedings of the 3rd International Conference and Workshops on Advances in Information Security and Assurance*, pages 60–69. Springer-Verlag, 2009.

[23] Trusted Platform Module Main Specification Parts 1, 2, and 3. http://www.trustedcomputinggroup.org/resources/tpm_main_specification. Version 1.2, revision 116. March 2011.

[24] A. Vasudevan, B. Parno, N. Qu, V. D. Gligor, and A. Perrig. Lockdown: A Safe and Practical Environment for Security Applications. Technical report, Carnegie Mellon University, 2009.

[25] M. Volkamer, A. Alkassar, A.-R. Sadeghi, and S. Schulz. Enabling the Application of Open Systems like PCs for Online Voting. In *Proc. of Workshop on Frontiers in Electronic Elections*, 2006.