Deceptive Deletion Triggers under Coercion

Lianying Zhao and Mohammad Mannan

***Abstract*—For users in possession of password-protected encrypted data in persistent storage (i.e., "data at rest"), an obvious problem is that the password may be extracted by an adversary through dictionary attacks, or by coercing the user. Traditional full disk encryption (FDE) or plausibly deniable encryption (PDE) cannot adequately address such situations. Therefore, making data verifiably inaccessible in a stealthy and quick fashion may be the preferred choice, specifically for users such as government/corporate agents, journalists, and human rights activists with highly confidential secrets, when caught and interrogated in a hostile territory. Using secure storage on a Trusted Platform Module (TPM) and modern CPU's trusted execution mode (e.g., Intel TXT), we design *Gracewipe* to enable secure and verifiable deletion of encryption keys through a special deletion password. When coerced, a user can fake compliance and enter the deletion password; and then the user can prove to the adversary that Gracewipe has been executed and the real key is no longer available (through a TPM quote), hoping for a favorable situation (e.g., end of torture). To unlock the target encryption key the adversary can only guess passwords through the valid Gracewipe environment with a high-risk of triggering deletion of the real key. Based on our two primary Gracewipe prototypes (i.e., software-based FDE with TrueCrypt and hardware-based FDE with SED), we also design and implement an extended family of unlocking schemes for triggering deletion, to achieve better plausibility, security and usability. We incur between 2–2.5 seconds delay during boot, and no performance penalty at run-time.**

***Index Terms*—Coercion, full disk encryption, panic passwords, cryptographic data deletion**

I. Introduction and Motivation

PLAUSIBLY deniable encryption (PDE) schemes for file storage were proposed more than a decade ago; see Anderson et al. [2] for the first academic proposal (1998). In terms of real-world PDE usage, TrueCrypt [3] is possibly the most-widely used tool, available since 2004. Several other systems also have been proposed and implemented. All these solutions share an inherent limitation: an attacker can detect the existence of such systems (see e.g., TCHunt [4]). A user may provide *reasonable* explanation for the existence of such tools or random-looking free space; e.g., claiming that TrueCrypt is used only as a full-disk encryption (FDE) tool, no hidden volumes exist; or, the random data is the after-effect of using tools that write random data to securely erase a file/disk. However, a coercive attacker may choose to detain and punish a suspect up until the true password for the hidden volume is revealed, or up to a time period as deemed necessary by the attacker. Such coercion is also known as *rubberhose cryptanalysis* [5], which is alleged to be used in some countries (e.g., Turkey [6], USA [7]); several incidents of forced password extractions during border crossings have also been reported in the recent past (e.g., USA [8], France [9]). The use of multiple hidden volumes or security levels (e.g., as in StegFS [10]), may also be of no use if the adversary is patient. Another avenue for the attacker is to derive candidate keys from a password dictionary, and keep trying those keys, i.e., a classic offline dictionary attack. If the attacker possesses some knowledge about the plaintext, e.g., the hidden volume contains a Windows installation, such guessing attacks may (easily) succeed against most user-chosen passwords.

Another option for the victim is to provably destroy/erase data when being coerced, unbeknownst to the adversary (i.e., triggered in an undetectable way). Note that such coercive situations mandate a very quick response time from tools used for erasure irrespective of media type (e.g., magnetic or flash); i.e., tools such as ATA secure erase, and DBAN [11] that rely on data overwriting are not acceptable solutions (cf. [12]). Otherwise, the attacker can simply terminate the tool being used by cutting off the power, or make a backup copy of the target data first. The need for rapid destruction was recognized by government agencies decades ago; see Slusarczuk et al. [13]. For a quick deletion, cryptographic approaches appear to be an appropriate solution, as introduced by Boneh and Lipton [14] (see also [15], [16]). Such techniques have also been implemented by several storage vendors in solid-state/magnetic disk drives that are commonly termed as self-encrypting drives (SEDs); see, e.g., Seagate [17], HGST/Western Digital [18] (cf. ISO/IEC WD 27040 [19]). SEDs allow overwriting of the data encryption key via an API call. Currently, as we are aware of, no solutions offer pre-OS secure erase that withstand coercive threats (i.e., with undetectable deletion trigger). Even if such a tool is designed, still several issues remain: verifiable deletion is not possible with SEDs alone (i.e., how to ensure that the secure erase API has been executed); and undetectable deletion trigger does not mean undetectable execution (e.g., calls to the deletion API can be monitored via SATA/IDE interface). We use SEDs as part of our solution without directly depending on their key deletion API.

In this paper, we discuss the design and implementation of *Gracewipe*, a solution implemented on top of TrueCrypt[1] and SEDs that can make the encrypted data permanently inaccessible without exposing the victim. When coerced to reveal her hidden volume encryption password, the victim will use a special pre-registered password that will irrecoverably erase the hidden volume key. The coercer cannot distinguish

This work is the extension of an NDSS 2015 paper [1]. The new contributions are summarized under "Differences with the NDSS version" in Section I.

L. Zhao and M. Mannan are with the Concordia Institute for Information Systems Engineering at Concordia University, Montreal, Canada.
E-mail: {z_lianyi, mmannan}@ciise.concordia.ca

[1]Projects that are based on TrueCrypt codebase or related to TrueCrypt, can also used/adapted with Gracewipe; e.g., TCnext (http://truecrypt.ch), CipherShed (https://ciphershed.org), VeraCrypt (https://veracrypt.codeplex.com).

the deletion password from a regular password used to unlock the hidden volume key. After deletion, the victim can also prove to the coercer that Gracewipe has been executed, and the key cannot be recovered anymore. A trusted hardware chip such as the Trusted Platform Module (TPM) alone cannot realize Gracewipe, as current TPMs are passive (i.e., run commands as sent by the CPU), and are unable to execute external custom logic. To implement Gracewipe, we use TPM along with Intel trusted execution technology (TXT), a special secure mode available in several consumer-grade Intel CPU versions (similar to AMD SVM).

The basic logic in Gracewipe for a PDE-enabled FDE system (e.g., TrueCrypt) can be summarized as follows. A user selects three (types of) passwords during the setup procedure: (i) Password *PH* that unlocks only the hidden volume key; (ii) Password *PN* that unlocks only the decoy volume key; and (iii) Password *PD* that unlocks the decoy volume key and overwrites the hidden volume key (schemes with multiple *PDs* are discussed in Section V). These volume keys are stored as TPM-protected secrets that cannot be retrieved without defeating TPM security. Depending on the scenario, the user will provide the appropriate password. When coerced, the user can disclose *PDs* or *PN*, but not *PH*. Attackers' success probability of accessing the hidden volume can be configured to be very low (e.g., deletion after a single invalid password), and will depend on their use of user-supplied or guessed passwords, and/or the deployed variant in Gracewipe-XD; see Section V. Deletion (overwriting with zeros) of the hidden volume key occurs within the TPM hardware chip, an event we assume to be unobservable to the attacker. Now, the attacker does not enjoy the flexibility of password guessing without risking the data being destroyed.

The relatively simple design of Gracewipe however faced several challenges when implemented with real-world systems such as TrueCrypt and SEDs. As Gracewipe works in the pre-OS stage, no ready-made TPM interfacing support is available. We have to construct TPM protocol messages on our own. Furthermore, we primarily base Gracewipe on TrueCrypt as it is open sourced. Auditability is essential to security applications, and most other FDE solutions as we found are proprietary software/firmware and thus verifying their design and implementation becomes difficult for users. For this reason, we must be able to load Windows after exiting TXT (as TrueCrypt FDE is only available in Windows), which requires invocation of real-mode BIOS interrupts. It turned out to be a major challenge for Gracewipe. For the SED-based solution, we also choose to boot a Windows installation from the SED disk. However, our Windows-based prototypes require a few heuristic changes specific to the versions of tboot and Windows. This is due to Intel TXT's incompatibility with real-mode (switching from protected to real-mode is required by Windows boot) and Windows' unawareness of TXT. Booting a Linux-based OS after Gracewipe would have been easier to implement (we also managed to do so), but that would have less utility than the Windows-targeted implementations.

Note that, in Gracewipe, the victim actively participates in destroying the hidden/confidential data, and thus may still be punished, e.g., put into jail for a significant period of time (e.g., [20]; see also cryptolaw.org for a survey on related laws in different jurisdictions). Gracewipe is expected to be used in situations where the exposure of hidden data is no way a preferable option. We assume a coercive adversary, who may release the victim when there is no chance of recovering the target data. Complexities of designing technical solutions for data hiding (including deniable encryption and verifiable destruction) are discussed in a blog post by Rescorla [21].

Authentication schemes under duress have been explored in recent proposals, e.g., [22], [23]. Such techniques may be integrated with Gracewipe, but they alone cannot achieve its goals, e.g., being able to delete keys under duress.

Contributions.

1) We propose Gracewipe, a secure data deletion mechanism to be used in coercive situations, when protecting the hidden/confidential data is of utmost importance. To the best of our knowledge, this is the first proposal to enable the following features together: triggering the hidden key deletion process in a way that is indistinguishable from unlocking the hidden data; verification of the deletion process; preventing offline guessing of passwords used for data confidentiality; restricting password guessing only to an unmodified Gracewipe environment; and tying password guessing with the risk of key deletion.

2) We implement Gracewipe with a PDE-mode TrueCrypt installation, and with an SED disk. Our implementation relies on secure storage as provided by TPM chips, and the trusted execution mode of modern Intel/AMD CPUs; such capabilities are widely available even in consumer-grade systems.

3) From our implementation experience with TrueCrypt and SED, apparently the design of Gracewipe is generic enough that it can be easily adapted for other existing software and hardware based FDE/PDE schemes. SED-based Gracewipe is discussed elsewhere [1].

4) Apart from secure deletion, our pre-OS trusted execution environment may enable other security-related checks, e.g., verifying OS integrity as in Windows secure boot, but through an auditable, open-source alternative. To the best our knowledge, Gracewipe is the first project to enable running a fully-functional Windows OS at the end of a trusted execution session (Intel TXT).

5) We also analyze and compare several schemes for triggering password-based deletion, with considerations respectively on plausibility, security, and usability; some of these schemes are adapted from Clark and Hengartner [24]. We also discuss implementation of some selected schemes. We label these schemes as Gracewipe-XD (Gracewipe Extended Deletion). Users may choose a scheme suitable to their threat model.

Differences with the NDSS version [1]. We make the following significant changes to the current article. Five new deletion triggering password schemes have been introduced to address various attack scenarios, which were not considered previously (Section V). All schemes (the basic variant in [1] and the new ones) are also analyzed from security and ease-of-use perspectives (Section VII). Due to Windows being unaware of some protections as enabled by Intel TXT, we had to disable DMA for disk access in the previous implementation; this limitation has been mitigated in the current

implementation (Section IV-D). The basic Gracewipe variant would also result in TPM deadlock as expected according to the TPM specifications (Section IV-D); we manage to bypass this deadlock with the new schemes. The overall design has also been updated to use different TPM APIs that simplify the earlier design and implementation, without losing any security guarantees (Section III, specifically, under Section III-C). We also evaluate the performance overhead of Gracewipe at boot-time (Section VI); note that there is no run-time overhead.

II. Goals and Threat Model

Gracewipe leverages several existing tools and mechanisms, such as multiboot [25], chainloading,[2] tboot [26], and TrueCrypt. We assume the reader is familiar with these techniques (for a brief introduction, see [1]).

A. Goals and terminology

Goals. (1) When under duress, the user should be able to initiate key deletion in a way indistinguishable to the adversary. The adversary is aware of Gracewipe, and knows the possibility of key deletion, but is unable to prevent such deletion, if he wants to try retrieving the suspected hidden data. (2) In the case of emergency data deletion (e.g., noticing that the adversary is close-by), the user may also want to erase her data quickly. (3) In both cases, when the deletion finishes, the adversary must be convinced that the hidden data has become inaccessible and no data/key recovery is possible, even with (forced) user cooperation. (4) The adversary must be unable to retrieve TPM-stored volume encryption keys by password guessing, without risking key deletion; i.e., the adversary can attempt password guessing only through the Gracewipe interface. Direct offline attacks on volume keys must also be computationally infeasible.

Terminology and notation. We primarily target the software-based FDE with support for plausible deniability (termed as *PDE-FDE*, e.g., TrueCrypt under Windows). A *decoy system* refers to the one appearing to be the protected system. The user should maintain certain frequency of using it for the purpose of deception. A *hidden system* is the actual protected system, the existence of which *may* be deniable and can only be accessed when the correct password is provided. The user should avoid leaking any trace of its use (as in TrueCrypt; cf. [27]). *KN* is the key needed to decrypt the decoy system, and *PN* is the password for retrieving *KN*. Similarly, *KH* is the key needed to decrypt the hidden system and *PH* is the password for retrieving *KH*. In addition, *PD* is the password to perform the secure deletion of *KH*; note that there might be multiple *PDs* (see Section V), but to simplify discussion, we consider only one here. *KN* and *KH* are stored/sealed in TPM NVRAM, which can be retrieved using the corresponding password, only within the verified Gracewipe environment. We use hidden/protected/confidential data interchangeably.

Overview of how Gracewipe goals are achieved. For goal (1), we introduce *PD* that retrieves *KN* but at the same time deletes *KH* from TPM. Thus, if either the user/adversary enters a *PD*, the hidden data will become inaccessible and unrecoverable (due to the deletion of *KH*). *PN*, *PH* and *PDs* should be indistinguishable, e.g., in terms of password composition. In a usual situation, the user can use either *PH* or *PN* to boot the corresponding system. If the user is under duress and forced to enter *PH*, she may input a *PD* instead, and Gracewipe will immediately delete *KH* (so that next time *PH* only outputs a null string). Under duress, she can reveal *PN*/*PDs*, but must refrain from exposing *PH*. The use of any *PD* at any time (emergency or otherwise), will delete *KH* the same way, and thus goal (2) can be achieved.

Goal (3) can be achieved by a chained trust model and deterministic output of Gracewipe. The trusted environment is established by running the deletion operation via DRTM, e.g., using Intel TXT through tboot [26]. We assume that Gracewipe's functionality is publicly known and its measurement (in the form of values in TPM PCRs) is available for the target environment, so that the adversary can match the content in PCRs with the known values, e.g., via a TPM quote. Gracewipe prints a hexadecimal representation of the quote value, and also stores it in TPM NVRAM for further verification. A confirmation message is also displayed after the deletion (e.g., "A deletion password has been entered and the hidden system is now permanently inaccessible!").

For goal (4), we use TPM sealing, to force the adversary to use a genuine version of Gracewipe for password guessing. Sealing also stops the adversary from modifying Gracewipe in such a way that it does not trigger key deletion, even when a *PD* is used (otherwise unsealing would fail). We use long random keys (e.g., 128/256-bit AES keys) for actual data encryption to thwart offline attacks directly on the keys. A by-product of goal (4) is that, if a Gracewipe-enabled device (e.g., a laptop) with sensitive data is lost or stolen, the attacker is still restricted to password guessing with the risk of key deletion.

B. Threat model and assumptions

Here we specify assumptions for Gracewipe, and list several unaddressed attacks.

1) We assume the adversary to be hostile and coercive, but rational otherwise (cf. [21]). He is diligent enough to verify the TPM quote when key deletion occurs, and then (*optimistically*) stop punishing the victim, as the hidden password is of no use at this point. If the victim suspects severe retaliation from the adversary, she may choose to use the deletion password only if the protected data is extremely valuable, i.e., she is willing to accept the consequences of provable deletion.

2) The adversary knows well (or otherwise can easily find out) that a TrueCrypt disk is used, and probably there exists a hidden volume on the system. He is also aware of Gracewipe, and its use of different passwords for accessing decoy/hidden systems and key deletion. However, he cannot distinguish *PDs* from other passwords that the victim is coerced to provide.

3) The adversary can have physical control of the machine and can clone the hard drive before trying any password. However, we assume that the adversary does not get the physical machine when the user is using the hidden system (i.e., *KH* is in RAM). Otherwise, he can use cold-boot attacks [28] to retrieve *KH*; such attacks are excluded, but see also TRESOR [29].

[2] https://www.gnu.org/software/grub/manual/html_node/Chain_002dloading.html

4) The adversary may reset the TPM *owner* password with the *takeownership* command, or learn the original owner password from the victim; note that NVRAM indices (where we seal the keys) encrypted with separate passwords are not affected by resetting ownership, or the exposure of the owner password. With the owner password, the adversary can forge TXT launch policies and allow executing a modified Gracewipe instance. Any such attempts will fail to unlock the hidden key (*KH*), as *KH* is sealed with the genuine copy of Gracewipe. However, with the modifications, the attacker may try to convince the user to enter valid passwords (*PH*, *PN* or *PD*), which are then exposed to the attacker. We expect the victim not to reveal *PH*, whenever the machine is suspected to have been tampered with. We do not address the so called evil-maid attacks [30], [31], but Gracewipe can be extended with existing solutions against such attacks (e.g., [32]).

5) We exclude inadvertent leakage of secrets/passwords from human memory via side-channel attacks, e.g., the EEG-based *subliminal probing* [33]; see Bonaci et al. [34] for countermeasures. We also exclude *truth-serum* [35] induced attacks; effectiveness of such drugs is also strongly doubted (cf. [36]).

6) Gracewipe facilitates secure key deletion, but relies on FDE-based schemes for data confidentiality. For our prototypes, we assume TrueCrypt adequately protects user data and is free of backdoors.

7) We assume the size of *hidden data* is significant, i.e., not memorizable by the user, e.g., a list of all US citizens with top-secret clearances (reportedly, more than a million citizens[3]). After key deletion, the victim may be forced to reveal the nature of the hidden data, but she cannot disclose much.

8) We assume Intel TXT is trustworthy and cannot be compromised and thus ensures the calculated measurements can be trusted (hence only genuine Gracewipe unseals the keys); past attacks [37], [38] on TXT include exploiting the CPU's SMM (System Management Mode) to intercept TXT execution. SMM attacks can be addressed by Intel SMI transfer monitor (STM [39]). We also assume that hardware-based debuggers cannot compromise Intel TXT. We could not locate any documentation from Intel in this regard.[4] As documented [40], AMD's SVM disables hardware debug features of a CPU.

III. Gracewipe Design

In this section, we expand the basic design as outlined in Section I. We primarily discuss Gracewipe for an FDE solution with deniable hidden volume support (i.e., PDE-FDE), and we use TrueCrypt as a concrete example. The FDE-only version (e.g., based on SED, not discussed here) is simpler than the PDE-FDE (TrueCrypt) design, e.g., no decoy volume and no chainloading are needed. These two versions mostly use the same design components, differing mainly in the key unlocked by Gracewipe and the destination system that receives the key.

A. Overview and disk layout

Gracewipe inter-connects several components, including: BIOS, GRUB, tboot, TPM, *wiper* (provides Gracewipe's core functionality—see below under "Wiper"), TrueCrypt MBR, and Windows bootloader. The hidden data is stored encrypted on a hard drive, as in a typical TrueCrypt hidden volume. We assume two physical volumes: one hosting the decoy system (regular TrueCrypt encrypted volume), and the other volume containing the hidden system (hidden TrueCrypt volume). *KN* and *KH* are technically TrueCrypt volume "passwords" for the two volumes respectively, but we generate them from a random source. Both are stored in TPM NVRAM, and are not typed/memorized explicitly by the user. In the deployment phase, they are generated with good entropy and configured as TrueCrypt "passwords". Each valid user password (including any *PD*) will unlock a corresponding key in TPM NVRAM for a specific purpose. See Fig. 1 for Gracewipe components.

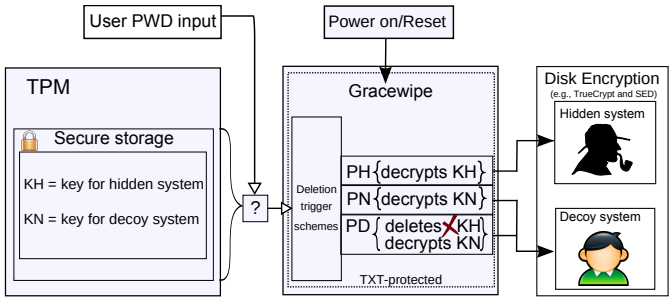

Fig. 1: A generalized representation of Gracewipe. *PN* = password for the decoy system; *PH* = password for the hidden system; *PD* = deletion password. *PH* unlocks *KH*, the key that decrypts the hidden system; *PN* unlocks *KN*, the key that decrypts the decoy system; *PD* deletes *KH* from TPM NVRAM, and may optionally unlock *KN*.

Wiper. The core part of Gracewipe's functionality includes bridging its components, unlocking appropriate TPM-stored keys, and deletion of the hidden volume key. We term this part as the *wiper*, which is implemented as a module securely loaded with tboot. It prompts for user password, and its behavior is determined by the entered password (or more precisely, by the data retrieved from TPM NVRAM with that password). Namely, if the retrieved data contains only a regular key (*KH*/*KN*), the wiper passes it on to TrueCrypt, or if it appears otherwise (as designated by a deletion indicator) to have a control block for deletion, the wiper performs the deletion and passes the decoy key *KN* to TrueCrypt. We modified TrueCrypt to directly accept input from the wiper (i.e., no password prompt), and boot the corresponding encrypted system. Note that each user password corresponds to one TPM NVRAM index. Each index contain an indicator value (byte-long; 'P' for *PH*, 'K' for *KN* and 'D' for *PD*) concatenated with a proper decryption key (*KH*/*KN*, or for the deletion index either some random data or *KN*). Both the indicator and key values are protected by TPM sealing, and an attacker cannot exploit the indicator values (see Section VIII under item (b)).

As the wiper must operate at an early stage of system boot and still provide support for relatively complex functionality, it must meet several design considerations, including: (1) It must be bootable by tboot, as we need tboot for the measured launch of the wiper. This can be achieved by conforming to required file formats (e.g., ELF) and header structures (e.g.,

[3] http://www.usatoday.com/story/news/2013/06/09/government-security-clearance/2406243/

[4] See a related tboot discussion thread at (Aug. 2012): http://sourceforge.net/p/tboot/mailman/message/29747527/

multiboot version number). (2) It must load the TrueCrypt loader for usual operations, e.g., decrypt the correct volume and load Windows. This is mainly about parameter passing (e.g., TrueCrypt assumes register DL to contain the drive number). (3) It must access the TPM chip and perform several TPM operations including sealing/unsealing, quote generation, and NVRAM read/write. Note that at this point, there is no OS or trusted computing software stack (such as TrouSerS [41]) to facilitate TPM operations. (4) It must provide an expected machine state for the component that will be loaded after the wiper (e.g., Windows). Both TrueCrypt and Windows assume a clean boot from BIOS; however, Windows supports only strict chainloading, failure of which causes several troubles including system crash (see Section IV-D).

B. Execution steps

Gracewipe's execution flow is outlined in Fig. 1. It involves the following steps: (1) The system BIOS loads GRUB, which then loads tboot and other modules including the wiper and ACM SINIT module. (2) Tboot performs necessary checks and calculates/matches the platform measurement with the values stored in TPM (halts on failure). (3) The wiper prompts the user for password, and uses the entered password to access TPM indices where we store *KH*/*KN* one by one (the extended schemes are different, see Section V). If no index is accessible, an invalid password is received (resulting reboot/halt). (4) As part of the TPM accessed data, an indicator field shows if the entered password is *PH*, *PN* or *PD*. Upon reception of *PD*, the wiper immediately erases *KH* from TPM, and performs a quote to display the attestation string on the screen. (5) In the case of *PH* or *PN*, the wiper copies the decryption key (i.e., TrueCrypt "password") to a memory location to be retrieved later by TrueCrypt. (6) The wiper switches the system back to real-mode, reinitializes it by mimicking what is done by BIOS at boot time. (7) TrueCrypt MBR is executed, which proceeds as if the wiper-copied "password" is typed by the user and loads the system as usual (the decoy or the hidden one).

C. Sealing in NVRAM

TPM specifications mandate mechanisms against guessing attacks on password-protected NVRAM data (e.g., only a few passwords may be consecutively tested within a specific period of time). However, such mechanisms are inadequate for Gracewipe as the adversary has physical control and can patiently keep testing passwords, and user-chosen passwords tend to be relatively weak. The implementation of such mechanisms is also vendor-specific (see Section IV-D). If the adversary would like to brute-force a specific index a few times until the chip is locked out and reset it with TPM_ResetLockValue, he may eventually succeed by automating the process.

To address this, we apply TPM's data sealing technique, so that if an altered software stack (i.e., anything other than the genuine copy of Gracewipe) is run, the desired data will not be unsealed, and thus will remain inaccessible. Note that sealing does not disallow guessing from within the Gracewipe environment; however, when Gracewipe is active, each guess may unlock the hidden/decoy data, or trigger key deletion.

Instead of directly sealing the keys into NVRAM, we make use of the access-control-based PCR binding to achieve the same goal. When an NVRAM index is defined, selected PCRs are specified as the access requirement in addition to the user password (authdata). The stored key can be accessed only if both the password and the PCR values (correct environment) are satisfied. This design choice prevents offline guessing of user passwords protecting the sealed keys, as opposed to the following construction: use *PN* as the authdata secret to protect *KN* stored in NVRAM in its sealed form. Without the correct environment, *KN* cannot be unsealed. However, by checking the TPM's response (success/failure) to a guessed authdata secret value, the attacker can learn *PN* and other valid passwords, without going through Gracewipe; the attacker can then use the (guessed) valid passwords in Gracewipe to unlock corresponding keys. With our current construction, TPM will output the same failure message, if either PCR values or passwords are incorrect.

D. Password management

Under the strong adversarial model in Gracewipe, the user (e.g., a security personnel) is expected to properly maintain the configured passwords, and if they are lost the recommended solution is redeployment; i.e., there must be no *password recovery*. The data or keys protected under Gracewipe must not be backed-up in any Internet-accessible storage under any circumstances; this will enable easy coercion even after a successful local deletion. However, password update can still be supported; we propose a simple mechanism below (can be extended to accommodate other schemes in Section V).

At the *same* password prompt where the user normally unlocks the system at boot-time, password update mode can be triggered by using a special key sequence (e.g., "Ctrl+Enter" instead of the regular "Enter" key). The received password is first handled by the deployed deletion triggering scheme, i.e., deletion will be initiated if *PD* is typed. After the password update mode is entered (i.e., deletion is not triggered), the user is prompted to enter an existing password to be changed (*PH*/*PN*/*PD*), and then type the new password twice. Gracewipe will try to access indices one by one until a success and replace the protecting password with the new one. Note that in order not to reveal any further information in the password update mode, no explicit feedback is provided, i.e., merely "Update process is done!" is displayed regardless of a successful update or failure. Also, a random delay can be added so that timing characteristics do not help distinguish valid passwords. We have implemented this scheme.

IV. Implementation with TrueCrypt

In this section, we summarize certain implementation considerations of Gracewipe specific to TrueCrypt; for details, see [1]. We also discuss several side effects resulting from our implementation choices and corresponding workarounds. The implementation effort mainly involves the wiper (Gracewipe core), TrueCrypt modifications, and a few configuration steps to make the components work together.

Our choice of Windows is largely in consideration of its prevalence, and TrueCrypt FDE's availability only under Windows. We have also successfully booted up Linux via Gracewipe with fewer changes compared to Windows. Gracewipe in itself is a boot-time tool, which does not run

along-side the user OS. For our prototype system, we used a primary test machine with an Intel Core i7-3770S processor (3.10 GHz) and Intel DQ77MK motherboard, 8GB RAM with 1TB Western Digital hard drive.

A. Implementing the wiper

Approximately, the wiper has 400 lines of code in assembly, 700 lines in C and 1300 lines of reused code from tboot. As discussed in Section III-A, TPM must be accessed by the wiper that runs at an early stage of system boot, i.e., right after GRUB and tboot. Due to lack of TPM access support at boot time (at least for NVRAM storage as in our case), we must handle the communications between TPM and the wiper, and implement a subset of the TCG software stack [41].

We choose to communicate with TPM through the MMIO interface for future compatibility and consistency with tboot. After being able to send commands to TPM via MMIO, we implement the authdata-protected NVRAM access functions (for secure storage of Gracewipe keys). Due to inadequate documentation, we had to reverse-engineer related functions in the TCG stack for our implementation.

Moreover, for verifying the correct execution of Gracewipe (and thus the deletion of *KH*), the wiper must be able to perform a TPM Quote. A quote operation involves generating a signature on a requested set of PCRs, and a verifier-provided nonce with TPM's attestation identity key (AIK). We allow the verifier (adversary) to enter a string of his choice as the nonce and store the quote value in an unprotected NVRAM index as well as displaying it on the screen.

At the end, we have developed the following TPM functions: *tpm_nv_read_value_auth()*, *tpm_nv_write_value_auth()*, *tpm_nv_define_release()*, *tpm_loadkey2()*, *tpm_quote2()*; we reuse most other functions from tboot.

B. Adapting TrueCrypt

To make TrueCrypt aware of Gracewipe, we require some changes in its source code. We keep such changes to a minimum for easier maintenance and deployment. They are mostly in BootLoader.com.gz (BootMain.cpp), and a few minor changes in BootSector.bin (BootSector.asm). In BootSector.bin, changes are for the modified version of BootLoader.com.gz to pass the original integrity check (CRC32). In BootLoader.com.gz (TrueCrypt modules), the modifications are mainly for receiving decrypted passwords (treated as keys in Gracewipe) from the wiper without user intervention.

C. Orchestrating components

Additional deployment-time efforts are needed to make Gracewipe components work seamlessly. Such efforts include configuring GRUB (with a Gracewipe-customized menu.lst), initializing TPM, and installing TrueCrypt. Also, as the integrity of the Gracewipe environment relies on tboot's policy enforcement, it is critical to ensure the proper setup of the MLE policy and tboot's custom policy.

Preparation in the host OS. A script that works with the TrueCrypt installer must automatically generate a strong key (i.e., random and of sufficient length) to replace the user-chosen password. This is done for both *KN* and *KH*. Then the user must copy (manually or with the help of the script) *KN* and *KH* to be used with Gracewipe. She must destroy her copies of the two keys after the setup phase.

Preparation in Gracewipe. Gracewipe comes with a single consolidated binary with two modes of operation: deployment and normal. Modes are determined by the value (zero/non-zero) in an unprotected NVRAM index; note that, reinitializing Gracewipe has no security impact (beyond DoS), but still a simple password can be set to avoid inadvertent reset. If the value is non-zero, normal mode is entered; otherwise, Gracewipe warns the user and enters the deployment mode.

In the deployment mode, the user is prompted for the three passwords (*PN*, *PH* and *PD*) of her choice and the two keys (*KN* and *KH*) generated in the OS. The wiper seals the two keys with the current environment measurements into NVRAM indices with the three passwords. The indices can be user-configured to avoid conflicts with other use of the TPM; however, the order of password to index assignment is randomized to avoid any possibility of interference with the deletion process using time differences (cf. Section VI). Note that different unlocking schemes may involve different procedures, see Section V. In the end, the wiper toggles the mode value for the next time to run in the normal mode.

D. Windows and TPM issues

Disabling TXT DMA protection for Windows. During implementation, we faced several issues related to Windows, mostly due to Windows being unaware of protections enabled by Intel TXT. Unlike Linux, Windows also cannot be adapted for TXT as it is closed-sourced. Below, we discuss a DMA problem and its solution.

TXT protects the execution environment from unauthorized code. The I/O protection (i.e., no peripheral on the bus other than the measured code can access protected memory regions) is enforced in hardware by IOMMU in collaboration with the chipset. By default, it is left enabled, when TXT is torn down with instruction GETSEC[SEXIT] (see [42]); the guest OS is supposed to be aware of the DMA protection, and perform any additional cleanup operations.

In addition to the platform-specific fixed DMA Protected Range (DPR, usually 3MB in size), custom Protected Memory Regions (PMRs [43]) can also be specified to SINIT for DMA protection. SINIT guarantees that the measured program (in our case: tboot and the wiper) is covered by either the DPR or one of the PMRs; otherwise, the program cannot be started.

The consequence of the aforementioned DMA protection depends on the OS taking control after TXT exit; e.g., if the OS is aware of IOMMU, the protected ranges can be avoided or remapped, or IOMMU should be disabled. For Gracewipe with Windows, IOMMU must be disabled due to Windows' unawareness. There is an IOMMU register *DMAR_PMEN_REG*, and by setting its *DMA_PMEN_EPM* bit to 0, the IOMMU PMR can be disabled.

If the PMRs are left enabled to Windows, as in the initial implementation of Gracewipe [1], the system will behave unpredictably, when memory access hits the protected regions. For example, right after Windows switches to the hard disk device driver from BIOS calls, the booting process fails with UNMOUNTABLE_BOOT_VOLUME (0x000000ED). The rea-

son code of 0xC000014F indicates a disk hardware problem, which is incorrect as we could boot Windows without tboot. Initially, we changed the ATA channel to use the "PIO" mode instead of "Ultra DMA Mode 5", and Windows booted successfully, but with disabled DMA for disk operations (i.e., degraded disk performance). Disabling IOMMU PMRs solved this issue without affecting disk performance.

TPM deadlock. Here, we discuss an issue originating from our somewhat unusual way of leveraging TPM. By design, TPM NVRAM is intended to provide protected access to confidential data. Such protection, especially with authdata access, is unsuitable to be used as a general purpose decryption oracle: a program accessing NVRAM is expected to supply the correct authdata secret, and a failed attempt is considered as part of a guessing attack or an anomaly.

We attempt to consecutively access one to three NVRAM indices with the same user password, i.e., until we can unlock a key, or fail at all three authdata-protected indices. Therefore, TPM actually counts each failed attempt as a violation and may enter a lockout state (released by an explicit reset or timeout); for details, see under *dictionary attack considerations* in the TPM specification [44]. We relied on TPM_ResetLockValue and time-out during our development.

Note that this limitation is mitigated by the DL-distance and pattern-based Gracewipe-XD schemes (Section V), where the secret data to compare with is unsealed from NVRAM and the user-typed password is not used as authdata (thus no failed authentication to access NVRAM).

V. Extended unlocking schemes

In the basic version of Gracewipe, only one or a few predefined *PDs* are allowed. In this section, we discuss password-based deletion triggers to avoid limitations of the basic Gracewipe design (see below). We adapt some existing schemes and explore new ones, and implement the most promising variants (called Gracewipe-XD).

Limitations of few deletion passwords. (1) The adversary's risk in guessing passwords is rather low. One or a few deletion passwords represent a very small fraction of a large set of possible passwords, e.g., millions in the case of brute-forcing, or at least hundreds, in the case of a small dictionary of most frequent passwords. (2) In terms of plausibility, the user is left with too few choices when coerced to provide a list of valid passwords; passwords other than *PN*, *PH* or *PD* will unlock no system nor trigger the deletion, and generate an error message. The adversary may choose to punish the user for any invalid password. With three valid passwords, the attacker's chance of guessing *PH* is at least $\frac{1}{3}$ (although the risk of triggering deletion is also the same).

A. Existing panic password schemes

We summarize several existing panic password proposals [24] (primarily for Internet voting), and analyze their applicability in our threat model (client-only).

2P. The user has a regular password (in our case, *PH*) and a panic password (in our case, *PD*). The 2P scheme applies to situations where authentication reactions are *unrecoverable*; e.g., if *PD* is entered in Gracewipe, further adversary actions cannot help data recovery, as the target key *KH* is now permanently inaccessible. However, if the attacker can extract both passwords from the victim, the chance of triggering deletion/panic is $\frac{1}{2}$; for 3P, the chance is: $\frac{2}{3}$, and so on. Thus 2P resembles the mechanism in the basic Gracewipe, which has both the aforementioned limitations.

2P-lock. When the reactions are *recoverable*, i.e., after *PD* is entered, knowing *PH* is still useful for the adversary (unlike Gracewipe), the adversary may continue guessing until he finds *PH*, but is bound to a time limit to end coercion (e.g., for escaping). In this case, a lockout mechanism can be applied to allow only one attempt, and make the two passwords indistinguishable. If a valid password is entered, the system always behaves the same (the panic passwords would signal coercion silently); then within a specified period, if a second valid but different password is used, the system locks out for a period longer than the adversary's time limit. However, 2P-lock is ill-suited for Gracewipe as there is no trusted clock to enforce the lockout (the BIOS clock can be easily reset).

P-Compliment. This scheme is applicable against *persistent* adversaries (i.e., reactions are recoverable and no time limit for coercion). Instead of having a limited number of panic passwords, any invalid password (i.e., other than the correct one) can be considered a panic password. This simple rule will result in user typos to trigger unwanted panic/deletion. To alleviate, passwords that are close to the correct one (i.e., easily mistyped) can be considered invalid (instead of panic), and thus the password space is divided into three sets based on edit distance: the correct password, invalid passwords and panic passwords. The user can now provide a large number of panic passwords, and typos are tolerated. Note that for a persistent adversary, it is assumed that there is no fatal consequence when a panic password is used (e.g., as in the case of online voting, the account is locked for a while). Thus, if the panic password and invalid password spaces are not well mixed, the adversary can try to approximate the boundary between them with multiple attempts. In Section V-C, we discuss a Gracewipe variant derived from P-Complement.

5-Dictionary. For better memorizability, a user can choose 5 words from a standard dictionary, using a password space division similar to P-Compliment: any 5 words in the dictionary other than the user-chosen ones are considered panic passwords; any other strings are invalid. This scheme tolerates user typos and provides a large set of panic passwords. However, the number of panic passwords ($^nP_5$, for a dictionary of n words) could still be much smaller than the invalid ones. We propose an adapted version of this scheme in Section V-D.

5-Click. For image-based schemes, any valid region in an image (excluding parts used for the correct login) can be used to communicate the panic situation. As Gracewipe relies only on text passwords, we exclude such schemes.

B. Counter-based deletion trigger

We design a counter-based mechanism by adapting 2P-lock to limit adversarial iterative attempts without increasing the risk of accidental deletion (by user). Reaching the limit of failed attempts is used to trigger deletion, instead of

locking out the system. Below, we provide the design and implementation of this adapted scheme.

Design. We keep the functionality of *PD*/*PH*/*PN* as in the basic Gracewipe design, i.e., entering *PD* will still initiate an immediate deletion. In addition, we now count the number of invalid attempts (i.e., entry of passwords other than *PD*/*PH*/*PN*), and use the counter value as a deletion trigger when a user-defined preset threshold (e.g., 10) is reached. The counter must be integrity-protected—i.e., can be updated only by the correct Gracewipe environment.

An important consideration is when to reset the counter value. Because a legitimate user may also mistype sometimes, and as such errors accumulate the deletion will be triggered eventually. We consider two options for resetting the counter value: (1) *Timeout*. It is mainly used in online authentication systems. However, without a reliable clock source, it is inapplicable to Gracewipe. (2) *Successful login*. Assuming that typos are relatively infrequent, a legitimate user will successfully login before the threshold is reached. We use such login to reset the counter. Note that only the entry of a valid *PH* is considered a successful login (but not *PN*, which can be revealed to the adversary when needed).

Implementation. This scheme is implemented by simply adding checks to where the entered password has failed to unlock any indices and where *KH* is successfully unlocked. The counter value is sealed in a separate NVRAM index with the environment measurements. We also bind the measurements to both read and write access of this index so that a modified program cannot even read it, not to mention updating. At deployment time, the counter is initialized to 0. A user-specific trigger value is secured the same way as the counter (i.e., no access outside the correct Gracewipe environment). If the adversary tries to reset either of them by re-initiating Gracewipe deployment, he will have *KH* erased first before both values are reset. The logic is as follows: Any invalid passwords will increment the counter; entry of *PN* does not affect the counter; entry of *PH* will reset it to 0. Whenever the counter value is equal to the trigger value, deletion is initiated.

C. Edit-distance-based password scheme

The counter-based deletion trigger can severely limit guessing attempts. However, if the user is forced to reveal all *valid* passwords, the attacker's guessing success rate will increase, due to the limited number of valid passwords (*PN* and few *PDs*). We design the following variant to counter both threats.

Design. Following the P-Compliment scheme, we use *edit distance* to divide the password space. Instead of predefined *PD*/*PH*/*PN*, we develop a rule to determine which category the password falls into during authentication. There will be no invalid passwords any more, and actions are taken silently (unlock the hidden system, decoy system, or trigger deletion).

We must balance between the risk of user typos and the coverage of passwords the adversary may guess. To measure the closeness between two passwords, we use *edit distance*: the number of operations (edits) required to convert one string to another. By centering to user-defined *PH*, we can treat the rest of the password space according to edit distance. The

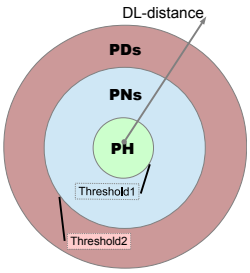

Fig. 2: Dividing password space with DL-distance

farther a password is from *PH*, the more likely it is entered by the adversary, and vice versa.

There are different variants of edit distance metrics, mostly depending on the types of allowed edit operations (e.g., insertion, deletion, substitution, and transposition). These metrics usually provide similar performance in distinguishing strings but with various computation complexity (less critical for Gracewipe). We use the Damerau-Levenshtein distance [45] (DL-distance), which considers only the following operations: insertion (one character), deletion (one character), substitution (one character) and transposition (two adjacent characters).

The choice of edit distance metrics may also involve other considerations. For example, we can take into account cognitive aspects (e.g., words with interchangeable meanings or user-specific typing habits), and device/physical aspects (e.g., common keyboard layouts). Especially, the CapsLock key must be checked, which can lead to large edit distance even when the correct password characters are typed, and convert all characters into lower case before processing. If such aspects are parametrized, a training process can also be introduced to customize Gracewipe-XD for a specific user.

If we denote the entered password as *PX*, the overall logic is as follows (see Fig. 2): if *DL-distance (PX, PH)* is less than or equal to *Threshold1*, *PH* is received; if *DL-distance (PX, PH)* is greater than *Threshold1* but less than or equal to *Threshold2*, then *PN* is received; otherwise, *PD* is received, which triggers the deletion process (including quote generation). We convert both *PX* and *PH* into lowercase for the DL-distance calculation (to avoid accidental CapsLock on status).

Note that, using DL-distance, multiple *PHs* can be allowed to access the hidden volume (e.g., by allowing *Threshold1* to be greater than 0). However, the usability benefit may be insignificant, as the range has to be centered to one *PH*, and thus forgetting *PH* may also indicate not remembering those that are only one or two characters different. On the other hand, allowing multiple *PHs* will increase the adversary's guessing probability (*PHs* cover more in the guessable space). At the end, we kept *Threshold1* to 0, i.e., a single *PH* is used.

In contrast to P-Complement [24], password spaces for *PN* and *PD* may not need to be well-mixed in our variant. However, we still re-examine any potential security consequence of our choice as follows:

1) Only a single *PD* will suffice to make the target data inaccessible. Thus approximating *PH* with multiple provided *PDs* or *PNs* is infeasible due to the high risk of deletion.

2) The adversary may extract non-*PH* words (i.e., *PNs* and *PDs*) from a victim, before launching a guessing attack

through the Gracewipe interface. Such seemingly non-secret information may help the adversary to identify boundaries between password spaces (cf. [46]). In Gracewipe-XD, edit distance is omnidirectional (unlike the simple depiction in Figure 2, where values are centralized to one *PH* on the same plane), and also parametrized by character sets, maximum length etc. We thus argue that the attacker cannot easily identify a trend/pattern pointing to *PH*.

Implementation. A significant change in the edit-distance-based scheme is that we must store *PH* in an NVRAM index for the DL-distance calculation. We seal *PH* with the Gracewipe environment measurements. At evaluation time, *PH* must be loaded to the system memory (with the correct Gracewipe environment), which may provide a chance to launch cold boot attacks [28]; note that DMA attacks are prevented by TXT. However, in our implementation, *PH* stays in memory for a short period of time—*PH* is unlocked after the candidate password is entered, and erased immediately after the DL-distance calculation (on average, 3-4 milliseconds in our test environment, for 8-character passwords). In this case, we argue that timing the cold boot attack to extract *PH* would be infeasible; for complexities of such attacks, see e.g., [47], [48]. Alternatively, *PH* can be copied directly to CPU registers to bypass memory attacks (cf. TRESOR [29]).

D. Other possible schemes

We have also explored more possibilities for the password schemes and deletion triggers. They can be further examined and implemented for specific use-cases.

Pattern-based deletion passwords. The user is allowed to define her own customized pattern for *PDs*, e.g., using regular expressions. Any string that does not match such pattern will be treated as *PNs* or invalid. This may provide better memorizability while allowing a large number of *PDs* (users must remember the pattern, but not the actual *PDs*). A foreseeable downside is that the adversary may learn the pattern (e.g., through text-mining) from passwords extracted from the victim, and then avoid passwords of such pattern when guessing. Also, this scheme does not address mistyping.

Misremember-tolerant deletion passwords. A user may accidentally enter a deletion password (e.g., due to stress, misremember) and realize the mistake instantly. In the basic Gracewipe, this would be fatal, as *KH* will be deleted immediately after receiving a *PD*. To reduce such accidental dental deletion, we adapt the counter-based scheme as follows.

For any entered *PD*, before triggering deletion, a counter value is checked; if it is already 1 (or any custom threshold), deletion is triggered as usual; otherwise, the counter value is incremented and the entered *PD* is just treated as *PN*. The counter value is initialized to 0 during deployment. A correct entry of *PH* will reset the counter. Thus, at the cost of allowing the adversary to try an additional password, accidental deletion can be avoided.

Small-dictionary scheme. The use of a built-in dictionary may serve as an alternative for tolerating user typos. The assumption here is that a mistyped word is more likely to be absent in the dictionary (but not always, e.g., *race* and *face*). Multiple user-chosen words (e.g., 5) form a *passphrase*. We adapt the 5-Dictionary scheme [24] to incorporate the following considerations: (1) We would like to follow the principle in the edit-distance-based scheme, i.e., the number of *PDs* is arbitrarily large to make sure that the probability of triggering deletion is rather high, meanwhile with *PNs* serving as a buffer zone to accommodate typos. (2) To increase the number of *PDs*, we can treat the invalid passwords in 5-Dictionary as *PDs*. However, most such invalid passwords are formed by non-dictionary words, and therefore can be easily entered by mistyping. Thus, we would like to design an adapted scheme that triggers deletion with non-dictionary words but tolerates mistyping.

Instead of using large natural dictionaries (e.g., English vocabulary), the user defines a custom dictionary that contains her memorizable strings (words or non-words). The size of the custom dictionary is relatively small that fits in TPM NVRAM, e.g., 50–100 words; such a small dictionary ensures that at a very high probability a random word falls outside the dictionary and may eventually lead to deletion. However, the custom dictionary must be both confidentiality and integrity protected, unlike the public standard dictionary in 5-Dictionary.

A *PH* consists of three (or more) segments, each picked from the custom dictionary. If *none* of the three segments of the typed password belong to the dictionary, the password is considered as *PD*, and deletion is triggered. Otherwise, the typed password is treated as *PN*. We assume that the probability of mistyping all three segments is low, reducing the chance of inadvertent deletion trigger by mistyping or even misremembering (see below). In contrast, without the knowledge of the custom dictionary, the attacker's guessable password space is as large as 5-Dictionary.

This scheme also partially addresses accidental deletion due to misremembering. If the user chooses to include all potential words/strings she may use in her passphrases for other accounts, even if she misremembers, still one or more segments of the misused passphrase fall in the custom dictionary and thus will be only treated as a *PN*.

Another benefit of the small-dictionary scheme is that the invalid password space and the deletion password space can be better mixed (for plausibility). Also, the hidden password is more difficult to approximate from extracted passwords (i.e., not reflecting constant space "away" from other passwords), because the custom dictionary diffuses candidate invalid/deletion passwords from the hidden one. Also, the custom dictionary being small allows it to be stored securely in TPM NVRAM, which removes access to the sealed dictionary without the correct Gracewipe environment (as opposed to sealed data stored on disk).

VI. Performance overhead

By design, Gracewipe merely replaces the user authentication part of an existing FDE scheme at boot-time, and does not interfere with the runtime performance of the OS or applications. In this section, the boot-time overhead of Gracewipe in normal operations is evaluated to demonstrate Gracewipe's practicality. We exclude the one-time deployment phase and quote generation after deletion as they occur only

in special cases, and involve additionally less than a second (for operations like loading keys in TPM and quoting).

Methodology of measurement. Unlike Linux kernel's *do_gettimeofday()*, we lack a reliable clock source in the pre-OS environment. We use CPU's Time Stamp Counter (TSC) via the *rdtsc* instruction. TSC stores the total number of machine cycles since the processor reset. A divisor (denoted as *N.TSC* hereafter) can be calculated so that TSC/N.TSC produces the total number of elapsed milliseconds (instead of machine cycles). This process is called TSC calibration, where the hardware 8253 Programmable Interval Timer (PIT) is programmed to produce a millisecond-long interval and the TSC value difference before and after is N.TSC. We do not try more recent alternatives (e.g., the *invariant TSC* feature in recent CPUs) as the original calibration-based approach has been tested and used in well-established projects, e.g., tboot, Linux kernel and GRUB2.

In our test machine, we get N.TSC values roughly between 3494388 and 3504892 across multiple calibration attempts (error $\pm$0.003ms). Instead of taking an average of the calibrated values, we use the actual N.TSC value right before each measurement to calculate the elapsed milliseconds, as per-measurement calibration reflects real-time characteristics. We perform each measurement 15 times and use the R project to calculate statistics.

Tboot. The choice of using tboot (as opposed to dealing with TXT with custom code) is justified by the fact that it has undergone sufficient public/expert scrutiny and thus is more reliable especially for the crucial TXT-handling logic. It also introduces an apparently acceptable level of latency.

By default, tboot enables debugging (to VGA, serial port or memory), which slows it down significantly, taking 30 seconds or more to complete. We disable debugging by passing necessary arguments. Our 15 independent measurements demonstrate coherent execution times: mean 1611.20ms, median 1611.96ms, standard deviation (sd) 6.08.

DL-distance-based Gracewipe-XD. User response time, such as password typing, is excluded from our measurement, since it is also needed for regular FDE. We hardcode the user input corresponding to each scenario for measuring only the execution time. We count from the point where control is taken over from tboot to the point where TrueCrypt is about to be loaded. Our attempts to measure the DL-distance-based scheme result in an average of 591.16ms, median of 589.71ms and standard deviation of 7.74.

The basic Gracewipe. As the basic design tries to unlock the three defined indices in sequence until a success, we separately time the three cases: (1) Success at the first index (including deletion, if it stores *PD*): mean 646.83ms, median 645.98ms, and sd 2.81. (2) Success at the first index (excluding deletion): mean 560.21ms, median 558.90ms, sd 3.78. (3) Success at the second index: mean 616.81ms, median 617.16ms, and sd 3.47. (4) Success at the third index: mean 746.97ms, median 743.40ms, and sd 10.61.

Promptness of deletion. We also measure the duration of the deletion operation (releasing and overwriting an NVRAM index). Over the 15 attempts, we found that deletion takes about 87ms (mean 86.62 and median 87.05), with a very small deviation (sd 1.39), supporting our claim for a quick deletion.

In summary, the overall latency introduced by Gracewipe is between 2 and 2.5 seconds.

VII. Generalized workflow and comparison

Different password and deletion schemes provide flexibility, and can be used in different application scenarios. However, the core Gracewipe features are always provided: plausible user compliance, undetectable deletion trigger, risky guessing and verifiability. In this section, we provide a generalized workflow for Gracewipe variants and compare them in terms of security benefits and ease of use.

Generalized workflow. At deployment time, in addition to setting up secrets (*KH*, *KN*), according to the actual variant of Gracewipe in use, the user defines corresponding parameters (thresholds, rules, or a custom dictionary). Each time the system is booted into Gracewipe in a TXT session (loaded by GRUB and tboot). The user is prompted for a password. The difference of Gracewipe variants is reflected in the evaluation of the entered password, which eventually produces an outcome (*KN*, *KH*, or deletion). Thereafter, the system is unlocked, or a quote is generated for later verification if deletion is triggered.

In normal operations, the user chooses to enter *PN* or *PH*, which unlocks *KN* for the decoy system or *KH* for the hidden system, respectively. If she mistypes or misremembers the password, the Gracewipe variant in use determines to what extent she can avoid triggering the deletion. When the user is coerced, she can be forced to provide a list of valid password, the number of which depends on the password scheme used.

Comparison. Table I summarizes several security and ease-of-use features of different deletion triggering/password schemes.

Large Deletion Space denotes the availability of many plausible deletion passwords that the user can reveal to pretend compliance. The basic Gracewipe only supports one or a few deletion passwords; the counter-based and misremember-tolerant variants are used with other schemes, and do not offer this property alone.

	Deletion triggers/password schemes	Large Deletion Space	High Guessing Risk	Typo-tolerant	Reduced Accidental Deletion	Non-RAM Secrets
Gracewipe	Single/few deletion passwords			●		●
Gracewipe-XD	Counter-based (add-on)	—	●	—		—
	DL-distance-based	●	●	●		
	Pattern-based	●	●	○		
	Misremember-tolerant (add-on)	—	—	●	●	—
	Small-dictionary	●	●	●	○	○

TABLE I: Comparison of Gracewipe password schemes. Keys: ● (offers the feature); ○ (partially offers the feature); — (scheme-dependent, "add-ons" may not be evaluated alone for certain properties); blank (lacks the feature).

High Guessing Risk represents a relatively high probability of triggering deletion with a guessed password. For the basic Gracewipe, it is just a few out of a large password space. Other schemes offer this feature by either rate-limiting (the counter-based one), or having a large deletion password space.

Typo-tolerant means the scheme tolerates user typos. This feature can be achieved either by using static passwords (assuming they are not close in terms of edit distance), or carefully managing the password distribution (e.g., greater distance between *PDs* and *PHs*). For the pattern-based scheme, typo tolerance is determined by the defined pattern.

Reduced Accidental Deletion denotes reduced risk from accidental deletion, e.g., mistakenly typing *PD* instead of *PH*. Schemes with fixed *PD* (s), e.g., the basic Gracewipe, obviously do not offer this feature. *PDs* are not predefined in the DL-distance and pattern-based schemes; however, misremembering *PH* or the pattern can still trigger accidental deletion. Small-dictionary partially tolerates misremembering, when at least one segment of a misremembered passphrase is found in the dictionary. The misremember-tolerant add-on can reduce the risk of accidental deletion in any variant.

Non-RAM Secrets indicates that plaintext e.g., *PH*, custom dictionary, are not exposed to the system memory. Although the feasibility of cold boot attacks is arguably low (recall that DMA attacks are already prevented by TXT), due to the very-short in-RAM password exposure period, avoiding plaintext secrets in memory is a better design choice. The basic Gracewipe's password evaluation is entirely TPM-bound. In contrast, the DL-distance-based scheme loads *PH* and the pattern-based scheme loads the pattern (e.g., a regex) in memory at evaluation-time. The small-dictionary scheme can partially avoid loading secrets in memory, if the custom dictionary is unlocked chunk by chunk to be matched with the typed passphrase (at the cost of performance), so that at a specific time only a small portion of the dictionary is in RAM.

VIII. Security Analysis

In this section, we analyze possible attacks that may affect the correct functionality of Gracewipe. Note that, the verifiability of Gracewipe's execution comes from a regular TPM attestation process. Since the good values (publicly available) only rely on Intel's SINIT modules, tboot binaries and Gracewipe, as long as the PCR values (via quoting) are verified to match them, it can be guaranteed that the desired software stack has been run. All Gracewipe variants assume that TPM chips are immune to known physical attacks; we briefly discuss several (historical) attacks on TPM in the NDSS version [1] (omitted here).

(a) Evil-maid attacks. In 2009, Rutkowska demonstrated the possibility of an evil-maid attack [30] (also termed as *bootkit* by Kleissner [31] in a similar attack) against software-based FDEs. The key insight is that the MBRs must remain unencrypted even for FDE disks, and thus can be tampered with. We consider two situations directly applicable to Gracewipe: 1) In normal operation (i.e., not under duress), the user may expose her password for the hidden system (*PH*). As soon as such an attack is suspected (e.g., when *PH* fails to unlock the hidden volume), users must reinitialize Gracewipe, and change *PH* (and other attempted passwords); note that, the user is still in physical control of the machine to reset it, or physically destroy the data. 2) Under duress, we assume that the user avoids revealing *PH* in any case. However, the adversary may still learn valid *PN*/*PDs* as entered by the user without the risk of losing the data (due to the lack of Gracewipe protection). The use of multiple valid *PDs* can limit this attack. Note that if an attacker copies encrypted hidden data, and then collects the hidden password through an evil-maid attack, the plaintext data will still remain inaccessible to the attacker due to the use of TPM-bound secrets (see under "Sealing in NVRAM" in Section III). The attacker must steal the user machine (at least, the motherboard and disk) and launch the evil-maid attack through a look-alike machine. Existing mechanisms against evil-maid attacks, e.g., MARK [49], can also be integrated with Gracewipe.

(b) Undetectable deletion trigger. As discussed under "Sealing in NVRAM" in Section III, sealing prevents guessing attacks without risking key deletion. Sealing also prevents an attacker from determining which user-entered passwords may trigger deletion, before the actual deletion occurs. If the adversary alters Gracewipe, any password, including the actual deletion password, will fail to unseal the hidden volume key from NVRAM. Since the deletion indicator lies only within the sealed data in NVRAM, the adversary will be unable to detect whether an entered password is for deletion or not (e.g., by checking the execution of a branch instruction triggered by the deletion indicator).

(c) Quoting for detecting spoofed environment. Currently, we generate a quote only in the case of secure deletion. However, in normal operations, the user may want to discern when a special type of evil-maid attack has happened, e.g., when the whole software stack is replaced with a similar environment (e.g., OS and applications). For this purpose, we can generate a quote each time Gracewipe is run and store it in NVRAM. By checking the last generated quote value, the user can detect any modifications to Gracewipe. In both secure deletion and normal operation, the selection of a proper nonce is required. We currently support both arbitrary user-chosen strings and timestamps as nonces. Nevertheless, the use of a timestamp is susceptible to a pre-play attack, where one party can approximately predict the time of the next use, and pre-generate a quote while actually running an altered binary. This is feasible because the malicious party has physical access, and thus, is able to use TPM to sign the well-known good PCR values for Gracewipe and the timestamp he predicts. Therefore, for spoofed environment detection, we recommend the use of user-chosen strings during quote generation, although it requires user intervention.

(d) Booting from non-Gracewipe media. The attacker may try to bypass Gracewipe by booting from other media. For an SED-based implementation, such attempts cannot proceed (i.e., the disk cannot be mounted). Even if he can mount the disk, e.g., with a copy of Gracewipe-unaware TrueCrypt, he must use the unmodified version of Gracewipe to try passwords that are guessed or extracted from the user (e.g., under coercion), as TrueCrypt volumes are now encrypted

with long random keys (e.g., 256-bit AES keys), as opposed to password-derived keys. Brute-forcing such long keys is assumed to be infeasible even for state-level adversaries.

(e) User diligence. We require users to understand how security goals are achieved in Gracewipe, and diligently choose which password to use depending on a given context. If the deletion password is entered accidentally, the protected data will be lost without any warning, or requiring any confirmation. Note that, we do not impose any special requirement on password choice; i.e., users can choose any generally-acceptable decent passwords (e.g., 20 bits of entropy may suffice). We do not mandate *strong* passwords, as the adversary is forced to guess passwords online, and always faces the risk of guessing the deletion password. Also, the user must reliably destroy her copy of the TrueCrypt keys when passing them to configure TrueCrypt. We can automate this key setup step at the cost of enlarging the trusted computing base. However, we believe that even if the whole process is without any user intervention, the adversary may still suspect the victim to have another copy of the key or the confidential data. Here we only consider destroying the copy that the adversary has captured.

IX. Related Work

Solutions related to secure deletion have been explored extensively both by the research community and the industry; see e.g., the recent survey [50]. However, we are unaware of solutions that target verifiability of the deletion procedure, and unobservability and indistingushability of the triggering mechanism—features that are particularly important in the threat model we assumed. Here we summarize proposals related to secure deletion and coercive environment.

Limited-try approach [21]. In a blog post, Rescorla [21] discusses technical and legal problems of data protection under coercion. Limitations of existing approaches including deniable encryption (such as TrueCrypt hidden volumes), verifiable destruction (Vanish [51]) have been discussed. He also proposes possible solutions, one of which is based on leveraging a hardware security module (HSM) with a *limited-try* scheme. The HSM will delete the encryption key if wrong keys are entered a limited number of times. As mentioned [21], such a system cannot be software-only as the destruction feature can be easily bypassed. Essentially, Gracewipe combines TPM and TXT to achieve HSM-like guarantees, i.e., isolated and secure execution with secure storage (albeit limited tamper-resistance), without requiring HSMs.

Secure deletion survey [50]. Reardon et al. provide a comprehensive survey of existing solutions for secure deletion of user data on physical media, including flash, and magnetic disks/tapes. Solutions are categorized and compared based on how they are interfaced with the physical media (e.g., via user-level applications, file system, physical/controller layers), and the features they offer (e.g., deletion latency, target adversary and device wear). However, SED-based solutions were not evaluated, which is of significance to secure deletion. The authors also presented a taxonomy of adversaries that a secure deletion approach is faced with. The adversary in Gracewipe can be classified as *bounded coercive* as he can detain the victim, and keep the device for as long as he needs with all hardware tools available, but cannot decrypt the Gracewipe-protected data without the proper key. Reardon et al. also discuss a few solutions involving encrypting user data and making it inaccessible by deleting the keys. The authors suggested to be more cautious about such cryptographic deletion and consider the adversary's true *computational bound* (which would be rather high for a state-level adversary).

STARK and MARK [32]. Müller et al. propose a protocol for mutual authentication between humans and computers, arguing that forged bootloader can trick the user to leak her password (cf. [30], [31]). Even with TPM sealing, attacks aiming to just obtain the user secret can still occur, as demonstrated by the tamper-and-revert attack to BitLocker [52]. STARK allows the user to set up a sealed user-chosen message, which should be unsealed by the machine before it authenticates the user. The user can then verify if it is her message. Each time a new message is set by the user to maintain the freshness, hence its name *monce*. Its improved version MARK uses a special USB device as secure storage to bootstrap the process credibly. Gracewipe may be extended with such techniques to defeat evil-maid attacks.

DriveCrypt Plus Pack [53]. DCPP can be considered the closest prior art to Gracewipe. It is a closed-source FDE counterpart of TrueCrypt, with the support for deniable storage (hidden volumes), destruction passwords and security by obfuscation. A user can define one or two destruction passwords (when two are defined, both must be used together), which, if entered, can immediately cause erasure of some regions of the hard drive, including where the encryption keys are stored. What DCPP is obviously still missing is a trusted environment for deletion trigger, and measurement for the deletion environment. The adversary may also alter DCPP (e.g., through binary analysis) to prevent the deletion from happening. More seriously, the adversary can clone the disk before allowing any password input.

X. Concluding remarks

We consider a special case of data security: making data permanently inaccessible when under coercion. We want to enable such deletion with additional guarantees: (1) verification of the deletion process; (2) indistingushability of the deletion trigger from the actual key unlocking process; and (3) no password guessing without risking key deletion. If key deletion occurs through a user supplied deletion password, the user may face serious consequences (legal or otherwise). Therefore, such a deletion mechanism should be used only for very high-value data, which must not be exposed at any cost, and where even accidental deletion is an acceptable risk (i.e., the data may be backed up at locations beyond the adversary's reach). We use TPM for secure storage and enforcing loading of an untampered Gracewipe environment. For secure and isolated execution, we rely on Intel TXT. Millions of consumer-grade machines are already equipped with a TPM chip and TXT/SVM capable CPU. Thus, Gracewipe can immediately benefit its targeted user base. The source code of our prototypes can be obtained via: https://madiba.encs.concordia.ca/software.html.

REFERENCES

[1] L. Zhao and M. Mannan, "Gracewipe: Secure and verifiable deletion under coercion," in *NDSS'15*, San Diego, CA, USA, Feb. 2015.

[2] R. Anderson, R. Needham, and A. Shamir, "The steganographic file system," in *International Workshop on Information Hiding (IH'98)*, Portland, OR, USA, 1998.

[3] TrueCrypt.org, "Free open source on-the-fly disk encryption software," version 7.1a (July 2012). http://www.truecrypt.org/.

[4] 16s.us, "TCHunt," tool for detecting encrypted hidden volumes (version: 1.6, release date: Jan. 29, 2014). https://github.com/stephenjudge/TCHunt.

[5] M. J. Ranum, "Cryptography and the law..." newsgroup post at sci.crypt (Oct. 16, 1990). https://groups.google.com/forum/#!msg/sci.crypt/W1VUQlC99LM/ANkI5zdGQIYJ.

[6] CNet.com, "Turkish police may have beaten encryption key out of TJ Maxx suspect," news article (Oct. 24, 2008).

[7] ArsTechnica.com, "Drug dealer: Cops leaned me over 18th floor balcony to get my password," news article (Apr. 22, 2015).

[8] SFGate.com, "Stockton mayor was briefly detained on return flight from China," news article (Oct. 2, 2015).

[9] TheRegister.co.uk, "Ex-Microsoft bug bounty dev forced to decrypt laptop for Paris airport official," news article (Jan. 6, 2015).

[10] A. D. McDonald and M. G. Kuhn, "StegFS: A steganographic file system for Linux," in *International Workshop on Information Hiding (IH'99)*, Dresden, Germany, 1999.

[11] Dban.org, "Darik's boot and nuke," open-source tool for hard-drive disk wipe and clearing. http://www.dban.org.

[12] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in *USENIX Security Symposium*, San Jose, CA, USA, Jul. 1996.

[13] M. M. G. Slusarczuk, W. T. Mayfield, and S. R. Welke, "Emergency destruction of information storing media," institute for Defense Analyses Report R-321 (Dec. 1987). http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA202147.

[14] D. Boneh and R. J. Lipton, "A revocable backup system," in *USENIX Security Symposium*, San Jose, CA, USA, Jul. 1996.

[15] G. D. Crescenzo, N. Ferguson, R. Impagliazzo, and M. Jakobsson, "How to forget a secret (extended abstract)," in *Symposium on Theoretical Aspects of Computer Science (STACS'99)*, Trier, Germany, Mar. 1999.

[16] J. Reardon, S. Capkun, and D. Basin, "Data node encrypted file system: Efficient secure deletion for flash memory," in *USENIX Security Symposium*, Bellevue, WA, USA, Aug. 2012.

[17] Seagate.com, "Protect your data with Seagate secure self-encrypting drives," http://www.seagate.com/ca/en/tech-insights/protect-data-with-seagate-secure-self-encrypting-drives-master-ti/.

[18] HGST.com, "Data center and enterprise storage solutions," https://www.hgst.com/sites/default/files/resources/DC-Ent-StorageSolutions-BR.pdf.

[19] ISO.org, "ISO/IEC FDIS 27040: Information technology – security techniques – storage security," target publication: Apr. 21, 2015. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=44404.

[20] TheRegister.co.uk, "Computing student jailed after failing to hand over crypto keys," news article (July 8, 2014).

[21] E. Rescorla, "Protecting your encrypted data in the face of coercion," blog post (Feb. 11, 2012). http://www.educatedguesswork.org/2012/02/protecting_your_encrypted_data.html.

[22] P. Gupta and D. Gao, "Fighting coercion attacks in key generation using skin conductance," in *USENIX Security Symposium*, Washington, DC, USA, Aug. 2010.

[23] H. Bojinov, D. Sanchez, P. Reber, D. Boneh, and P. Lincoln, "Neuroscience meets cryptography: Designing crypto primitives secure against rubber hose attacks," in *USENIX Security Symposium*, Bellevue, WA, USA, Aug. 2012.

[24] J. Clark and U. Hengartner, "Panic passwords: Authenticating under duress," in *USENIX Workshop on Hot Topics in Security (HotSec'08)*, San Jose, CA, USA, Jul. 2008.

[25] Gnu.org, "The multiboot specification," http://www.gnu.org/software/grub/manual/multiboot/multiboot.html.

[26] Intel.com, "Trusted boot (tboot)," version: 1.8.0. http://tboot.sourceforge.net/.

[27] A. Czeskis, D. J. S. Hilaire, K. Koscher, S. D. Gribble, T. Kohno, and B. Schneier, "Defeating encrypted and deniable file systems: TrueCrypt v5.1a and the case of the tattling OS and applications," in *USENIX HotSec'08*, San Jose, CA, USA, 2008.

[28] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: Cold boot attacks on encryption keys," in *USENIX Security Symposium*, San Jose, CA, USA, 2008.

[29] T. Müller, F. C. Freiling, and A. Dewald, "TRESOR runs encryption securely outside RAM," in *USENIX Security Symposium*, San Francisco, CA, USA, Aug. 2011.

[30] J. Rutkowska, "Evil maid goes after TrueCrypt!" online report (Oct. 16, 2009). http://theinvisiblethings.blogspot.ca/2009/10/evil-maid-goes-after-truecrypt.html.

[31] P. Kleissner, "Stoned bootkit," Black Hat USA (July 2009). http://www.blackhat.com/presentations/bh-usa-09/KLEISSNER/BHUSA09-Kleissner-StonedBootkit-PAPER.pdf.

[32] T. Müller, H. Spath, R. Mäckl, and F. C. Freiling, "STARK tamperproof authentication to resist keylogging," in *Financial Cryptography and Data Security (FC'13)*, Okinawa, Japan, Apr. 2013.

[33] M. Frank, T. Hwu, S. Jain, R. Knight, I. Martinovic, P. Mittal, D. Perito, and D. Song, "Subliminal probing for private information via EEG-based BCI devices," tech-report (Dec. 20, 2013). http://arxiv.org/abs/1312.6052.

[34] T. Bonaci, J. Herron, C. Matlack, and H. J. Chizeck, "Securing the exocortex: A twenty-first century cybernetics challenge," in *IEEE Conference on Norbert Wiener in the 21st Century*, Boston, MA, USA, Jun. 2014.

[35] A. Winter, "The making of "truth serum," 1920-1940," *Bulletin of the History of Medicine*, vol. 79, no. 3, pp. 500–533, 2005.

[36] Salon.com, "James Holmes and the ethics of "truth serum": Putting the Aurora shooter through a narcolanalytic interview won't provide truth or prove sanity," news article (Mar. 13, 2013).

[37] R. Wojtczuk and J. Rutkowska, "Attacking Intel trusted execution technology," Black Hat DC (Feb. 2009). http://www.blackhat.com/presentations/bh-dc-09/Wojtczuk_Rutkowska/BlackHat-DC-09-Rutkowska-Attacking-Intel-TXT-slides.pdf.

[38] R. Wojtczuk, J. Rutkowska, and A. Tereshkin, "Another way to circumvent Intel trusted execution technology," Invisible Things Lab, Tech. Rep., 2009, http://invisiblethingslab.com/resources/misc09/Another%20TXT%20Attack.pdf.

[39] Intel.com, "SMI transfer monitor (STM) user guide," revision 1.00 (Aug. 2015). https://firmware.intel.com/sites/default/files/STM_User_Guide-001.pdf.

[40] AMD.com, "AMD64 architecture programmer's manual volume 2: System programming," revision 3.25 (June 2015). http://support.amd.com/TechDocs/24593.pdf.

[41] "TrouSerS: The open-source TCG software stack," version: 0.3.8. http://trousers.sourceforge.net/.

[42] *Intel 64 and IA-32 Architectures Software Developer's Manual*, Intel.com, June 2014, volume 2C: Instruction Set Reference.

[43] *Intel TXT Software Development Guide - Measured Launched Environment Developer's Guide*, Intel.com, May 2014.

[44] *TPM Main: Part 1 Design Principles*, Trusted Computing Group, specification Version 1.2, Level 2 Revision 116 (March 1, 2011).

[45] G. V. Bard, "Spelling-error tolerant, order-independent pass-phrases via the damerau-levenshtein string-edit distance metric," in *Australasian Information Security Workshop (AISW'07)*, Ballarat, Australia, 2007.

[46] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart, "Cracking-resistant password vaults using natural language encoders," in *IEEE Symposium on Security and Privacy*, San Jose, CA, USA, May 2015.

[47] M. Gruhn and T. Müller, "On the practicability of cold boot attacks," in *Conference on Availability, Reliability and Security (ARES'13)*, Regensburg, Germany, Sep. 2013.

[48] R. Carbone, C. Bean, and M. Salois, "An in-depth analysis of the cold boot attack: Can it be used for sound forensic memory acquisition?" Jan. 2011, Technical Memorandum (TM 2010-296), Defence Research and Development Canada (DRDC), Valcartier.

[49] J. Götzfried and T. Müller, "Mutual authentication and trust bootstrapping towards secure disk encryption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 2, pp. 6:1–6:23, 2014.

[50] J. Reardon, D. Basin, and S. Capkun, "SoK: Secure data deletion," in *IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, May 2013.

[51] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data," in *USENIX Security Symposium*, Montreal, Canada, Aug. 2009.

[52] S. Türpe, A. Poller, J. Steffan, J.-P. Stotz, and J. Trukenmüller, "Attacking the BitLocker boot process," in *Technical and Socio-economic Aspects of Trusted Computing (Trust'09)*, Oxford, UK, Apr. 2009.

[53] SecurStar.com, "DriveCrypt Plus Pack," http://www.securstar.biz/drivecrypt-plus-pack.html.

Lianying Zhao is a Ph.D. candidate at the Concordia Institute for Information Systems Engineering, Concordia University, Montreal. He received his Master's degree in Tianjin University, China. He has over six years of experience in the industry prior to his Ph.D. studies, mainly on mainframe and embedded systems. He currently works on system and hardware-related security, and authentication.

Mohammad Mannan is an Associate Professor at the Concordia Institute for Information Systems Engineering, Concordia University, Montreal. His research interests lie in the area of Internet and systems security, with a focus on solving high-impact security and privacy problems of today's Internet. He is involved in several well-known conferences (e.g., program committee: ACM CCS 2016, ACSAC 2014, USENIX Security 2010; program co-chair: ACM SPSM 2016), and journals (e.g., ACM TISSEC, IEEE TDSC, IEEE TIFS).