DOWN With Trusted Devices

Mahalingam Ramkumar
Department of Computer Science and Engineering
Mississippi State University, Mississippi State, MS 39762.

Abstract

There is an ever-growing need for "trusted" devices, and consequently, assurances from technology for tamper-resistance, and read-proofing of secrets stored in such devices. We propose and investigate a simple security policy - **d**ecrypt **o**nly **w**hen **n**ecessary (DOWN) - and evaluate its implication on the security of secrets stored in trusted devices. We argue that the DOWN policy used in conjunction with *physical unclonable functions* (PUF) [1] may be a promising approach for realization of trusted devices.

While any key distribution scheme (used for securing interactions between trusted devices) would benefit from the use of the DOWN policy, key pre-distribution (KPD) schemes, particularly random key pre-distribution schemes, benefit substantially. We argue that the significant gains in security, coupled with the many inherent advantages of KPDs over PKI, render KPDs more suitable for trusted devices.

1 Introduction

One of the debilitating side effects of the rapid rate of technological advancements is the weakening of "natural bonds of trust" that develop over time, among human beings. The need for *artificial reinforcements* for trust have been growing steadily over the past few decades. While there are numerous indications of this trend, a very obvious example is the fact that a world without photo IDs would be unthinkable at present, while the need for it was not felt so strongly just a few decades ago!

However, technology also provides techniques for artificial reinforcements for the weakening bonds of trust. Cryptography, perhaps the most commonly used *artificial trust building mechanism*[1], is poised to become more and more an integral part of our day to day lives.

In spite of wide-spread use of cryptography in most conventional client-server based applications in use today, in the final analysis, it is still the individual human beings on whom the onus of "trust" is placed. For instance, end-users (or clients) authenticate themselves to a server using a shared secret (a passphrase). As anybody with access to the private key of the user can impersonate the user, the end-user is *trusted* not to disclose her private information. This trust is however not generally misplaced. For every user has the motivation (and the responsibility) to keep their private information, private. More importantly, disclosure of private information of a user, typically does not result in a great inconvenience for *other* users[2].

1.1 Trusted Devices

With recent advances in technology (mostly around wireless communications and miniaturization), and the foreseen applications of the future that are expected as a result, trusting fellow human beings is not going to be sufficient. Many evolving applications will rely on extensive mutual co-operation of a network of highly interconnected computers. Such co-operative networks of computers are expected to take over many of the mundane tasks that humans perform today.

For instance a group of computers working together may "decide" things as trivial as the setting of a thermostat (they may receive weather forecast directly from computers in the local weather station), or less trivial things like the appropriate course of action to avoid a collision (say computers working together in a car, interacting with computers in cars nearby).

Less futuristic applications may involve large scale deployments of ad hoc networks of wireless personal communication devices, where each device may be trusted to perform some tasks, like forwarding packets and co-operatively building routing tables, for overall good of the network. A crucial requirement in scenarios such as the above is the ability to "trust the devices". Trusted devices [2] are expected to possess *unflinching morals*[3] [3]. Trusted devices will, in general, not trust the owners or controllers of devices (the human operators) [2]. In other words, they cannot be directed to do something that violates the rules they are trusted to obey. Trusted devices however, *need to trust each other*.

1.1.1 Trust Between Trusted Devices

To facilitate trust between any two devices, there must exist some means of convincing each other that they "play by the rules," or are "compliant" to some pre-imposed rules. This can

[1] If we ignore mechanical locks.

[2] Even though in theory compromise of an end-user's password should not affect anybody else, in practice, this is the starting point of most attacks on computer systems - which affects many others too!

[3] For example, thou shalt always route packets; thou shalt not advertise false routing tables or weather data.

be facilitated using a suitable key distribution scheme (KDS). Only trusted devices would be provided with secrets which would enable them to authenticate each other. The secrets, however, have to be provided to a device *after* ascertaining that they are indeed compliant. In other words, the secrets serve as a "hook" for compliance. Once secrets are provided to devices, verification of the existence of secrets (without which mutual authentication would not be possible) indirectly provides a means for *verification of compliance*.

1.1.2 Read-Proofing and Tamper-Resistance

Just as human beings are trusted not to reveal their secrets in client-server applications, trusted devices are trusted not to reveal *their* secrets. To accomplish this, two fundamental assurances are required from technology - tamper-resistance and read-proofing [4].

Tamper-resistance is necessary to ensure that the software controlling the devices (which govern the *morality* of a device) cannot be changed. Read-proofing is necessary to ensure that secrets from a trusted device cannot be transferred to untrusted devices. Gennaro et al [4] provide some interesting insights into the inter-play between the two very distinct requirements. While the property of tamper-resistance may be extremely difficult to achieve, in most scenarios the ability obtain *tamper-evidence*, followed by *self-destruction* would be sufficient.

Self-destruction of trusted devices does *not* necessarily mean *complete* physical destruction of a device - it just implies that the secrets stored in the device are rendered *unusable* in the future. This would prohibit the device from authenticating itself to other trusted devices, and therefore loose its ability to interact with other devices.

Trust modules (or more commonly HSM - hardware security modules) are therefore increasingly relevant in evolving application scenarios. In spite of the fact that assumptions of tamper-resistance and / or read-proofing has been, and perhaps will continue to be a controversial issue among cryptographers [5]-[7], there is no denying the fact that they are going to be *mandatory* in the future with ubiquitous cryptography. After all, necessity has always been the mother of invention. Motivated by the pressing need we could very soon expect technology to provide suitable solutions.

1.2 Theses of the Paper

The theses of this paper is two-fold.

1. The Decrypt Only When Necessary (DOWN) security policy, when used in conjuction with Physical Unclonable Functions (PUF) could significantly improve the ability of trusted devices to protect their secrets.

2. Key pre-distribution (KPD) schemes are more suitable (than PKI) for practical large-scale deployments of trusted devices, especially when used in conjunction with the DOWN policy.

In Section 2 of this paper we provide a brief overview of key distribution schemes (KDS) with more emphasis on key pre-distribution (KPD) schemes. In Section 3 we discuss typical attacks employed for exposing secrets stored in trusted devices, and possible countermeasures. In Section 4 we introduce the DOWN policy, and discuss methods for practical realization of the DOWN policy. The primary advantages offered by the adoption of the DOWN policy are three-fold:
1. It minimizes (or even obviates) the steps needed for a trusted device to go from "in-use" state to "rest-state."
2. It provides a guarantee that *not more than a small fraction* of the secrets stored in trusted devices can be compromised by the attacker (in which process the device is rendered unusable in the future).
3. Non-volatile storage does *not* need *any* protection.

While the DOWN policy can be used to protect the secrets of any KDS, their positive effect on KPD schemes is more dramatic. We argue in Section 5, that KPDs, especially random KPDs [8], used in conjunction with the DOWN policy may be a better approach for trusted devices (as opposed to PKI). Conclusions are offered in Section 6.

2 Key Distribution Schemes

A KDS is a mechanism of distributing secrets to each participant in a system such that any two (or more generally any group of) participants can authenticate each other. The process of authentication typically involves discovery of a shared secret between the participants, which is facilitated by the KDS secrets provided to each participant (or node). The process of discovery of a shared secret also serves as a mechanism for mutual confirmation of the claimed identities of the participants.

In general KDSes could be divided into two categories. For the first category, the secrets provided to each node are *independent*. In other words, secret(s) of a node do(es) not provide *any* information regarding the secret(s) of *other* nodes. This category includes the most commonly used KDSes like Kerberos (or any KDS based on the Needham-Schroeder symmetric key protocol [9]), and PKI.

For the second category however, secrets distributed to each node are *not* independent. The secrets are all derived from a set of secrets chosen by a *trusted authority* (TA) (who deploys the network). This category of schemes are referred to as key pre-distribution (KPD) schemes[4]. With KPD schemes, a group of "colluding" nodes could pool their secrets together to compromise the entire system (or obtain secrets of *all* nodes). There is thus a concept of n-secure KPDs. A n-secure KPD can resist collusions of up to n nodes.

[4]It could be argued that the "basic" key pre-distribution scheme (where for a network of N nodes $\binom{N}{k}$ secrets are distributed such that each node gets $N-1$ secrets) belongs to the former category. However, as it does not scale well to large network sizes, it is irrelevant for our discussion.

2.1 Key Predistribution Schemes

A KPD scheme consists of a trusted authority (TA), and N nodes with unique IDs (say $ID_1 \cdots ID_N$). The TA chooses P secrets $\mathcal{R}$ and two public operators $f()$ and $g()$. The operator $f()$, is used to determine the secrets $\mathcal{S}_i$ that are preloaded in node i. Any two nodes i and j, with preloaded secrets $\mathcal{S}_i$ and $\mathcal{S}_j$ can discover a unique shared secret K_{ij} using a *public* operator $g()$ without further involvement of the TA.

The restrictions on $f()$ and $g()$ in order to satisfy these requirements can be mathematically stated as follows:

$$\begin{aligned} \mathcal{S}_i &= f(\mathcal{R}, ID_i) \\ K_{ij} &= g(\mathcal{S}_i, ID_j) = g(\mathcal{S}_j, ID_i) \\ &= f(\mathcal{R}, ID_j, ID_i) = f(\mathcal{R}, ID_i, ID_j). \end{aligned} \quad (1)$$

As $g()$ is public, it is possible for two nodes, just by exchanging their IDs, to execute $g()$ and discover a unique shared secret. As the shared secret is a function of their IDs, their ability to arrive at the shared secret provides mutual assurances to i and j that the other node possesses the necessary secrets $\mathcal{S}_j$ and $\mathcal{S}_i$ respectively. The secrets preloaded in each node is referred to as the node's *key-ring*. We shall represent by k, the size of the key ring.

KPDs themselves may be divided into two broad categories - deterministic and random KPDs. Most KPDs [10] - [13] based on finite field arithmetic belong to the former category. A n-secure deterministic KPD is unconditionally secure as long as n or less nodes have been compromised. If more than n nodes are compromised however, the *entire* KPD is compromised - or the attacker could discover all the P secrets of the TA and therefore all secrets of all nodes. However, the concept of n-secureness is not an adequate description of random KPDs. For random KPDs, an attacker, by exposing secrets from n_e nodes could discover shared secrets between arbitrary nodes with a probability p_e.

In general deterministic KPDs based on finite-field arithmetic are very efficient in terms of number of keys that each node needs to store (k) in order to achieve n-secureness. For example for Blom's KPD, $k = n+1$. However, they are computationally more expensive - for example, a n-secure Blom's KPD involves evaluation of a n-degree polynomial in a prime field in order to calculate shared secrets.

Random KPDs [8], [14] - [17] on the other hand generally need more storage (even though it is still $k = O(n)$, except for the scheme in Ref [17] where $k \approx O(n^3)$). For instance HARPS needs $k = n_e\sqrt{\mathrm{e}}\log\left(\frac{1}{p_e}\right)$ keys to be (n_e, p_e)-secure. However, random KPDs are computationally inexpensive - they need only pure symmetric cryptography primitives like hash functions and / or block ciphers (or multiplication is not needed).

3 Protecting Trusted Computers

A trusted computer (usually a computer in a single chip) would include
1. A CPU,
2. secure volatile RAM (or work bench) where intermediate results of cryptographic calculations are stored
3. ROM / EEPROM where the software (the operating system and software that govern device specific functions) may be stored, and
4. non-volatile storage for storing secrets.

The ROM / EEPROM could include a cryptographic hash of its contents to prevent attackers from modifying its content. The hash could be a HMAC - based on a secret that is stored securely in the processor or just a public hash function[5].

While most computers would have external (outside the chip) RAM, and various levels of caches on board the chip, for single chip computers the entire RAM may be inside the chip. Yet, the onboard RAM could consist of various levels of caches, with different levels of protection. For instance seperate caches may be used for instructions and data[18]. Even the data cache could be split into two categories - one for storing sensitive cryptographic keys and intermediate results of cryptographic calculations, and another for storing less sensitive data.

The data stored in the non-volatile storage would include KDS secrets, personal secrets (or a personal key ring) of the user (say when the trusted computer is a smart card), and other application dependent data. All stored data apart form the KDS secrets would possibly be encrypted with the KDS secret assigned[6] to the computer.

The computer itself may be in-built into a "trusted device." For example a trusted device may be a set-top box (STB) for encrypted video playback. The trusted computer in this case would perform decryption of encrypted compressed video, and other circuitry in the device may perform video decompression (say MPEG-2 decoder), D/A conversion etc.. Alternately, the computer may just be enclosed in a "tamper resistant" casing with very little additional circuitry. For example, in devices like smart-cards, the additional circuitry may be restricted to sensors and a battery back-up, to detect attempts to open-up the casing, and perhaps reset all secrets stored in non-volatile memory.

3.1 Attacks and Countermeasures

A series of attacks involving compromising security of smart-cards, has resulted in a large body of literature investigating the feasibility of building secure hardware, [6] - [7], [19] - [30], that could be expected to protect its secrets. Kommerling et al [21] provide the following classification of attacks [21] on chips

[5] Using a public hash function would still help in thwarting attacks where the attacker cannot have precise control over the process that is used for modifying the ROM / EEPROM.

[6] The secrets "assigned" to the computer would depend on the KDS in use. For KDSes based on public-key cryptography, the secrets may be *generated* by the device itself.

to extract buried secrets - non-invasive attacks, invasive attacks, and semi-invasive attacks.

3.1.1 Non-invasive Attacks

Non invasive attacks are performed without damaging the device. Typically such attacks involve "eavesdropping" on electromagnetic radiation emanating from the devices with the intention of gaining clues about the secrets stored inside. Some non-invasive attacks involve exploitation of security vulnerabilities in software running on devices.

Another category of non-invasive attacks is the *glitch* attacks, where the device is made to malfunction (the process to achieve this could be as simple as shining a powerful light beam on the surface). The aim of a glitch attack is to effect bypass of security protections - say to gain access to some unauthorized resource. Typically this would involve making the program counter of the CPU jump over a set of instructions which perform authentication checks. However, glitch attacks could indirectly lead to compromise of stored secrets by bypassing some protection mechanisms (say mechanisms that sense intrusions and erase all secrets stored in non-volatile memory). In other words a non-invasive attack may be used as a precursor for an invasive attack that follows.

Non-invasive attacks could be very dangerous - if they could be successful. In general however, by proper shielding, it may be easier to guard against non-invasive attacks.

3.1.2 Invasive Attacks

Invasive attacks involve opening up (and usually destroying the chip in the process) to decipher the secrets. With sophisticated sensors for tamper-evidence, this could be tackled. Many invasive attacks are also based on inducing faults in memory [22], [23] and employing differential power analysis [24]. Protection mechanisms against such attacks have also been investigated [25].

Another category of invasive attacks involve "scavenging" bits from "footprints" left behind in magnetic and solid state memories [26], [27]. Secrets that may have been stored temporarily even in volatile memory could be scavenged if the secrets had been stored in a memory location for long periods. "Safe" erasure [28] of contents in magnetic and solid state memory (or removing all traces of their footprints) may require many repeated overwriting operations. The ability of the attacker to scavenge bits is also significantly enhanced by cooling the device (say by immmersing it in liquid nitrogen). Another useful technique for the attacker is to induce a glitch attack that causes the CPU to "hang" (while sensitive secrets are stored in volatile memory). The attacker could wait for a long time to ensure that the secrets currently in volatile memory leave a deep footprint before plunging the device into liquid nitrogen.

3.1.3 Semi-Invasive Attacks

Perhaps the most worrisome of current attacker capabilities is the gamut of available of **semi-invasive** attacks. Unlike invasive attacks which may provide devices with an opportunity to detect tampering attempts and reset their secrets, semi-invasive attacks, especially attacks employing focused ion beams (FIBs) or particle beam techniques in general - may go undetected [21].

In Ref. [29] the manufacturers claim protection against FIB attacks by employing an "active shield" consisting of a thin parallel grid of signal lines, covering the entire surface of the chip. Data is constantly transmitted at one end of the lines and evaluated at the other end. Any mismatch (which may occur even if one line is damaged by the FIB) would trigger appropriate counter measures.

However with improved technology, it may be possible for attackers to have better and better control over the FIBs. Nevertheless, it may still be reasonable to assume that it would be possible to protect *some* parts of the silicon substrate (or some lines on the substrate) by not providing a clear line of sight access[7]. Perhaps even multiple layers of "active shields" could be used. If we can ensure that *only a fraction* of the lines can be tapped, it may be possible to build *private* circuits in such a way [30] the attacker gains no knowledge (by tapping a few lines).

3.2 Protecting Secrets at Rest

Protecting secrets stored in a device when the device is at rest (or powered off), is in general significantly more difficult to accomplish. It may be possible to employ a much larger number of sensors that are actively monitoring for intrusions while the device is *powered-on* compared to the case when the device is powered-off. The limited battery back-up inside a chip may not be able to power as many intrusion sensing mechanisms. Even when intrusions are sensed the limited reaction time (and limited battery power) may not be enough to perform the many repeated overwriting operations to erase the secret without a discernible footprint.

While *rest encryption* [31], [32], which could help protect secrets when the device is at rest, has received substantial attention for purposes of encrypting databases in order to prevent unauthorized access of the contents[8], the problem of rest-encryption for trusted devices is very different. The fundamental difference is in terms of concerns regarding *storage of the key used for encrypting* data at rest. For rest-encryption of databases, typically the encryption key is stored *outside* the database. Obviously, for rest encryption of the secrets of a trusted device, the keys *cannot* be stored outside the device[9], as their security is not guaranteed. So the secret used for encrypting the secrets should also be stored within the trusted device (in non-volatile memory).

When the number of bits to be protected at rest is large, and if the battery back up in chips can ensure that the attacker may

[7] It may not be possible to bend beams!

[8] Say by someone who manages to remove a hardisk from a database server.

[9] Trusted devices will not trust anybody else to protect their secrets!

not be able to extract *all* bits by tampering with a device the following approach could be used[10].

Before the device is powered off, the stored secrets (say $\mathbb{S}$) may be divided into two sets L_0 and R_0, or $\mathbb{S} = [L_0 \parallel R_0]$, followed by two Fiestel-like encryption rounds,

$$L_1 = R_0, R_1 = E_{R_0}(L_0) \quad L_2 = R_1, R_2 = E_{R_1}(L_1) \tag{2}$$

As the sizes of R_0 and L_0 may be substantially larger than block sizes used for encryption, in the equation above E_{R0} stands for a chained mode of some symmetric block cipher, using a key derived from R_0 - say a cryptographic hash of R_0. Similarly, $E_{R_1}(.)$ represents chained block encryption using a key derived from R_1. The encryption can be easily reversed when the node is powered back on again as

$$R_0 = D_{L_2}(R_2) \quad L_0 = D_{R_0}(L_2) \tag{3}$$

However, even if *any* of the T bits of $\mathbb{S}' = [L_2 \parallel R_2]$ are *not* recoverable after tampering (due to partial erasure of secrets using the battery back-up as source), the complexity involved in recovery of $\mathbb{S}$ by an attacker is equivalent to that of breaking a T-bit cipher! For example, if the size of $\mathbb{S}'$ is 128,000 bits, and even if the attacker is able to retreive 127,900 of those bits from the non-volatile memory, the complexity involved for the attacker for determining $\mathbb{S}$ is equivalent to that of breaking a 100-bit cipher.

3.3 Physical Unclonable Functions

Silicon PUFs exploit uncontrollable statistical delay variations of connections and transistors etched on substrates, in each manufactured chip. Even though a manufacturer may fabricate many chips with identical masks, each chip would have a unique set of delays in its components, which even the manufacturer will not be able to measure with a high enough precision. In other words, each chip (or some components taken together in a chip) implements an "un-characterizable" (and therefore unclonable), unique, physical one-way function (POWF). The concept of physical one way functions was introduced by Ravikanth [35], analyzed further in [36], and served as the motivation for silicon PUFs in electronic chips, proposed by Gassend et al [1]. PUFs were more extensively investigated by Lim [37].

The *unique* physical unclonable function in each chip could be used along with challenge-response protocols for identification of devices. For instance, a chip with PUF $\mathcal{H}()$ generates *responses*

$$R_i = \mathcal{H}(C_i) \tag{4}$$

for different *challenges* C_i. While the manufacturer has no way no way to determine $\mathcal{H}$ for any chip, he could collect many such challenge-response pairs (CRPs) (C_i, R_i) before the chip (or device containing the chip) is distributed[11]. Later, by randomly choosing challenges from the set $\{C_i\}$ the manufacturer could verify authenticity of the devices, by checking their responses against known responses $\{R_i\}$.

Suh et al [19] discuss a comprehensive architecture for a single-chip secure processor that rely on PUFs. The key idea behind the architecture is as follows. If a user of a device can share a secret with the PUF in the chip (in the device), then it permits the user to share a secret with other programs that may run on the secure processor. The "user" may be the owner of the chip (say owner A of chip C_A) or even *other* users who share a secret with the PUF in C_A (for example users B and C may share a secret with the PUF in C_A).

"Sharing a secret with the PUF" implies knowledge of one more CRPs of the PUF. This is achieved by not permitting the users to *explicitly* choose the challenge to the PUFs (the challenges are modified in a unpredictable fashion by employing some secure one-way function). This makes it possible to "assign" *other* users (users other than the owner) some challenges (for which they are provided with the responses apriori). Thus it is now possible for B and C to authenticate C_A even over an untrusted channel. In addition, the secret shared between the "remote" users and the PUF is bootstrapped to establish secrets between the user and the programs running on the secure processor.

3.3.1 Rest Encryption with PUF

The possibility of PUFs provides a quantum leap in the ability to protect secrets stored in devices at rest by *eliminating the need for a reliable battery back-up*. The secret used for encrypting the secrets stored in non-volatile memory could be provided by the PUF!

For example, the key used for encrypting the secrets could be $R_x = \mathcal{H}(X)$, where X is a known quantity (may be stored in the clear in the device or may be a fixed value). When the device is at rest, there is no way for the attacker to determine R_x! When the device is powered on, the device (or the software running in the device) could "challenge" the PUF with X and obtain the response R_x, and then proceed to decrypt the stored secrets!

3.4 "Reasonable" Assurances

In general, it may be possible to better protect secrets if the number of secrets that have to be protected very well is small. If multiple secrets have to be protected, the secrets can be encrypted[12] with the single highly protected secret. For instance, a *highly protected* secret could be stored in special CPU registers - hidden even from the OS kernel [18], for performing cryptographic computations.

Various techniques could be used to ensure that such secrets do not leave a deep footprint in memory. One such technique is to make sure that the bits are periodically ones complemented - say every few milliseconds. However, this should be performed in

[10] This is in principle very similar to *all-or-nothing* [33], [34] encryption transforms.

[11] More specifically, until a certain stage in the chip manufacturing process.

[12] As we shall see soon, this may not be very easy to accomplish.

hardware to make sure that this process continues to occur even if the CPU hangs. Another option may be use memories that have to be refreshed periodically in response to software signals generated periodically by the CPU. So if the CPU hangs, the contents of the memory would be "discharged."

Similarly, when the device is powered-on it may be possible to protect some selected areas of volatile memory (in addition to the highly protected hidden register) by not providing clear line of sight access, or by employing private circuits [30].

Thus some reasonable guarantees that could be provided by technology for read-proofing and tamper-resistance are

G0 - Cryptanalytic attacks that employ weaknesses of ciphers are infeasible [4].

G1 - It is possible to protect some limited areas (a special cache memory and a volatile CPU register) of volatile storage from compromise of secrets *while the device is functioning* (or it is not possible for the attackers to tap lines to read the contents of these parts of the volatile memory, without triggering protection circuits).

G2 - It is possible to read-proof the contents of one or a few *volatile* registers when the device is *powered off* (by ensuring that the secret does not leave a footprint in the memory).

G3 - It is possible (using PUFs) to protect the secrets of a device stored in non-volatile memory, when the device is powered-off.

4 The DOWN Policy

The DOWN policy is motivated by the fact that most cryptographic operations have some inherent *atomicity*. At any point in time only one or may be even part of a secret may be necessary for cryptographic computations. Thus the secrets could be (stored) encrypted at *all* times. The secrets, or parts of it are *decrypted only when necessary*.

For instance, let us assume that the KDS secret to be protected is a RSA private exponent r (say of size 1024 bits). Let p represent the public exponent and n the RSA modulus. Decryption of a some cipher text C involves modular exponentiation of C with r - or $P = C^r \bmod n$. However, to perform the exponentiation, only one bit of r is needed at any point in time (say exponentiation using the square and multiply algorithm). We could thus keep r encrypted at all times, and decrypt each bit as and when necessary.

For KPD schemes, the use of DOWN is even more straightforward, and natural. For example with a KPD scheme where each device has k secrets, for every cryptographic operation, only one of the k keys is needed at any point in time.

The primary motivation for the DOWN policy is the implication of guarantee G1 - that it may be possible to prevent attackers from continuously monitoring (reading) contents of some sensitive memory locations. However, it *may be possible* to extract some information from those portions of the memory using invasive attacks involving scavenging of bits from footprints. Further, just like every operation has some atomicity, so does the operation of going from "in-use" state to "rest" state (or power-on to power-down). This transition period perhaps provides the most advantageous exploit for an attacker. An attackers best strategy would be to disable parts or even the entire sequence of events involved in the state-transition. Methods to accomplish them would be perhaps by using FIB techniques to cut-off the source of power to the processor, sudden freezing of the device, or glitch attacks that may cause the CPU to hang (followed by freezing of the device for volatile memory scavenging).

Without the DOWN policy, such an attack may imply compromise of the entire private secrets of the device (say all bits of the RSA private key). that is *currently* stored in volatile memory. With DOWN, if proper care is taken - for instance the decrypted secrets are always stored in the *same* memory location in the the special cache memory (and therefore "overwrite" each other), the attacker can discover at most 1 of the k secrets (for example, the last contents before the cache was flushed), based on residual information. The attacker can *never* expose more than one secret from each device (in case of KPDs) or one part of the secret (say one bit of the private RSA key).

Further, the strength of the "footprint" left behind would depend on the duration [28] for which a secret is stored in a particular location. With DOWN, secrets are stored in volatile memory only for very short durations - and therefore should not leave a clear footprint that could be detected by the attacker. Nevertheless, as mentioned earlier, it may be possible for the attacker to cause the CPU to "hang" resulting in a deep footprint of the *current* contents (which, if the attacker is lucky, may have one of the k secrets) of the volatile memory.

4.1 Realization of DOWN

Let us assume that we could protect a single secret K_V unconditionally (say stored in a special hidden CPU register - guarantee G2), we could use K_V to encrypt all KDS secrets. Whenever a secret is necessary, it could be decrypted, used, and deleted from RAM as soon as it is not needed anymore. However,the process of encryption using K_V *cannot* involve transfering of K_V to less protected volatile memory. Thus we are restricted to simple operations like XOR for encryptions (which could be done in the hidden CPU registers). Obviously, it is not safe to XOR *multiple* secrets with the same key K_V, or even use elementary transformations of K_V (like shifts, which could be performed without moving K_V out of the hidden CPU registers) for this purpose. For if the attacker determines even one of the k secrets (and he knows the encrypted secret stored in non-volatile memory) he can determine K_V.

If we could also implement a block cipher E (say 128-bit block size and 128-bit keys) or a secure one-way function H in hardware (say 128-bit output), accessible only to the CPU (hidden from the OS kernel), then we could generate multiple secrets from the key K_V. For instance we could generate secrets $K_i = E_{K_V}(i)$ or $K_i = H(K_V \parallel i)$ for $1 \leq i \leq k$ and encrypt the i^{th}KDS secret with key K_i. If both the i^{th}KDS se-

cret and K_i are of the same length (say 128 bits), we could simply use an XOR operation for encryption. On the other hand, if we desire to encrypt each bit (say of a RSA private key) seperately, we could encrypt the i^{th}bit b_i by using another secure hash function (implemented in software) $h(K_i \parallel b_i)$ that produces a one-bit output. Of course, it is always possible to encrypt a 1024 bit RSA private key as 8 different 128-bit chunks. In this case, upto 128 bits of the private key may be "exposed" (in the "secure" cache memory) at any point in time - which is perhaps still acceptable.

When DOWN is used, rest-encryption is *always* in effect - even when the device is functioning! Thus the most important consequence of using DOWN, is that *non-volatile memory in the chip does not need any protection*.

However, we need store K_V in some "special" non-volatile memory when the device is powered off, which needs to be protected. For this purpose, PUF is very handy!

4.2 Realization of DOWN with PUFs

If PUFs are able to provide a secure and unclonable hash function which could be used efficiently in practice, even the hash function H (or the block cipher E) used for realization of DOWN could be replaced with the PUF $\mathcal{H}$. This also simultaneously *eliminates the need for the additional key* K_V (or guarantee G2 is not required). Each KDS secret can be encrypted using responses to different challenges by the PUF. The i^{th}KDS secret can be encrypted with $K_i = \mathcal{H}(i)$.

In order to ensure that CRPs collected by manufacturers [19] will not help them discover K_is, the challenges employed by the device could be *randomized*. To randomize the challenges the key K_i could be evaluated as

$$K_i = \mathcal{H}(h(N_u \parallel i)) \tag{5}$$

where $h()$ is a secure one-way function (implemented in software). The random value N_u may be stored in the clear in non-volatile memory. The seed N_u - the choice of which the manufacturer has no way of knowing apriori - ensures that the probability that the manufacturer could have accumulated any "useful" CRPs is extremely low.

Similar to [19] PUF enabled CPUs in trusted computers would support an additional "instruction" GETRESP(j) where j is and integer. GETRESP(j) returns $\mathcal{H}(j)$. Usually the integer $j = h(N_u \parallel i), 1 \leq i \leq k$.

At all times, the k secrets are stored encrypted in non-volatile memory (by XORing with respective K_is). So there is no explicit *transition* from "in-use" to "rest-state"!

4.3 Practical Relization of DOWN

However, it may not be *practical* to employ the PUF $\mathcal{H}$ for encrypting *all* keys. One of the main problems with practical PUF realizations is that the delays are very sensitive to environmental changes. In other words the function $\mathcal{H}$ in each chip also changes with environmental variations. Thus unless some corrective measures are taken - like employing error correction codes (ECC) [38] - it may not be possible to *reliably authenticate* chips based on the PUF. PUFs could also suffer aging - drifting of $\mathcal{H}$ with use, and over long periods of time.

Such compensatory mechanisms however will reduce the entropy (or the non-determinism) of $\mathcal{H}$. This would imply use of PUFs with high complexity (involving a large number of gates) to realize cryptographically strong one-way functions which at the same time do not drift with environmental variations. Further, the error-correction codes required are typically computationally very intensive (at least compared to block ciphers or hash functions).

A practical realization of the DOWN policy could still use the **clonable** hash function H (or block-cipher E) along with a highly protected secret K_V (or guarantee G2 is required). PUFs could be used exclusively for protecting the key K_V at rest. For instance, a random challenge X may be chosen and stored in the clear in non-volatile memory. The secret K_V could be encrypted by XORing it with $K_x = \mathcal{H}(X)$ or if desired with $K_x = \mathcal{H}(H(X))$. Thus, at rest, K_V could be stored as $K_{VX} = K_V \oplus K_x$ in a hidden non-volatile register. Note that the non-volatile register need not be read-proof - as for an attacker who can gain access to X, there is no way to obtain K_X while the device is at rest. Even though the non-volatile storage register does not need to be protected, the non-volatile register could still be hidden as there is no *need* for the kernel to have access to the register.

As the PUF is used only once during the boot-up process and once during the shutdown process, the *complexity* involved (mainly due to ECC) in generating reliable CRPs is not a very crucial issue. Further, drift of the PUF due to aging should be less of an issue - as long as aging does not affect the CRPs between one shut-down and the *next* boot-up process! Note that unlike other applications [1], [19], [39] - [40], of PUF in literature we do not rely on PUFs to *authenticate a device* (in which case aging *would* be a serious issue).

For highly resource constrained devices (say wireless sensors) use of PUFs may not be feasible. However, typically the active life of such devices may depend on their battery life. Such devices may *always* be powered-on - they may never need to go into rest-state. When they do go down when their battery runs out, the secret K_V is lost forever, and so are the secrets stored in the non-volatile memory as they are encrypted with K_V. As PUFs are used only for rest-encryption, such devices do *not* need PUFs.

4.4 Self-Destruction

A trusted computer would be expected to self-destruct whenever any tampering attempt is sensed. Self-destruction is typically achieved by erasing all secrets stored in non-volatile memory.

However, as we assume that the non-volatile memory *need not be protected* - as the secrets stored are encrypted - erasing non-

volatile storage does not help. An attacker could "replay" the contents of the non-volatile storage at any time. In other words, the attacker has the ability to create a "back-up" of the entire contents of the non-volatile memory at any point in time and restore its content later.

The hardware, firmware, the system software in ROM (or EEPROM), are read-only - or things that cannot be changed, and the secure RAM is volatile, and therefore would "forget" any changes after the device is powered off. This implies that as long as the attack does not result in any permanent damage to the hardware / firmware, the device *cannot* self-destruct.

To see this, assume that an attacker attempts an attack, which is detected by some sensors in the chip. Appropriate actions are taken to say flush the RAM (which may have one exposed secret), and then shut-down the device. Ideally, under such circumstances, the device should be rendered unusable in the future, even though it is fully functional. The computer has to set a "TAMPERING_DETECTED" flag somewhere to indicate that a possible intrusion attempt was sensed, before it powers itself down. If this is possible, the computer would check the flag during the next boot-up sequence, and "refuse" to boot. However, the flag cannot be stored in the non-volatile memory - as the attacker has the ability to revert its contents back to the state *before* the attempted attack.

Thus some part of the non-volatile memory should be protected. It does not need to be read-proof - but only write-proof. In practice, this implies that an attacker would not be able to reset the flag without physically destroying the device.

It might be better to use a multiple bit sequence instead of a one bit register to guard against attacks where an attacker could flip random bits in in non-volatile memory by irradiation. If multiple bits are used (and only a specific sequence of bits constitute a valid entry) it may be significantly more difficult for an attacker to *precisely* reset *many* bits (without destroying the chip).

As we use a non-volatile register for storing K_{VX}, this itself could be used as the flag. When the device is running K_V is stored in a volatile hidden register. During the boot-up sequence, K_{VX} is erased from the non-volatile register and K_V stored in the volatile register. During normal shut-down process, K_V would be transferred to the non-volatile register as K_{VX}. Any abnormal shutdown would result in an improper value of K_{VX} and render the secrets (and therefore the device) unusable. As K_{VX} is stored in a special register, it may not be difficult to read-proof the register. In other words, K_{VX} cannot be read without physically damaging the chip. If this *weak* read-proof guarantee holds, then the register may not even be *weakly* write-proof. Thus the non-volatile register just needs to be *either* weakly read-proof or weakly write-proof.

5 Choice of KDS for Trusted Devices

There are three basic options for the choice of KDS for trusted devices - a public key infrastructure (PKI), or a Kerberos-like KDS, or KPDs. Privacy and practicality constraints dictate that interactions between any two nodes, for purposes of establishing mutual trust, should not need external mediators - or trust should be established in an *ad hoc* manner. This eliminates Kerberos-like models as a viable alternative. PKI and KPDs however, cater for ad-hoc mutual authentication. We shall discuss the suitability of different KPDs first. We shall then compare KPDs with PKI for this purpose.

6 KPDs with DOWN

The DOWN policy, in general, renders a n-secure KPD nk-secure. Further, as KDS secrets can be stored in unprotected non-volatile memory, the size of k might not be a serious issue. However what might influence the performance is the fact that each cryptographic operation (for instance the process of mutual authentication of two nodes by arriving at a shared secret) may involve an *additional* k cryptographic operations for decrypting the stored secrets. As the cryptographic operation is actually carried out in hardware this may not be a serious issue. However, the overheads imposed by the need for memory access (for each time a secret is needed it should be fetched from non-volatile memory which could be a block based storage device) could be significant. In addition, there may be context switches (to a secure kernel mode) whenever a secret K_i used to encrypt the i^{th}secret is queried.

However, it is not always necessary that *all* the k secrets are used for *each* computation for all KDSes. While this may be true for PKI (all k parts of the private key need to be used for every operation that needs the private key) and some KPDs (for example Blom's KPD [10]) this is not true for random KPDs like HARPS [8]. It might therefore be fair, in order to compare relative merits of different KPD schemes, to use the *number of secrets that actually need to be fetched* from non-volatile memory and decrypted for cryptographic operations, rather than the *number of secrets actually stored* - as unprotected non-volatile memory storage complexity is not a critical issue. We shall however restrict our comparison to two KPD schemes - Blom's scheme and HARPS.

In a n-secure Blom's scheme, the TA chooses $\binom{n+1}{2}$ secrets in $\mathbb{Z}_P$ (where P is a large prime), and generates a polynomial $f(x,y) = \sum_{i=0}^{n}\sum_{j=0}^{n} a_{ij}x^i y^j \bmod P$, where $a_{ij} = a_{ji}$ are $\binom{n+1}{2}$ independent secrets chosen by the TA. Every node is assigned a unique public ID. A node A which has public ID r_A and receives $g_A(x) = f(x, r_A)$ securely ($g_A(x)$ has $n+1$ coefficients, corresponding to $n+1$ secrets of the node A) from the TA. Two nodes A and B can calculate K_{AB} = K_{BA} = $f(r_A, r_B)$ = $f(r_B, r_A)$ = $g_A(r_B)$ = $g_B(r_A)$ independently.

For (P, k, L) HARPS, the TA chooses P secrets. Each node (with a unique ID) is provided with a subset of those secrets (of cardinality k). The k secrets are further hashed repeatedly many times (randomly chosen between 1 and L). The indexes of the secrets assigned to each node, and the number of times each key

is hashed is based on a public one-way function, seeded by the ID of the node. Two nodes, by exchanging their IDs, can execute the public one-way function to determine the secrets they share and the corresponding hash depths. The shared secret between two nodes A and B is derived from all shared keys. For each shared key, the node with the lower hash depth, hashes its key the requisite number of times to reach the same hash depth as the other node.

As a numerical example, for HARPS with $P = 15000$ (number of independent secrets chosen by the TA) and $k = 1000$ (number of secrets stored in each node), the average number of *shared* secrets between two nodes is roughly 67 (or $\frac{k^2}{P} \approx 67$). For Blom's scheme however, as all k secrets are needed for any cryptographic operation (to evaluate the $k-1$ degree polynomial the coefficients of which are the k secrets). It may be therefore be fair to compare HARPS with $k = 1000$ with Blom's scheme with $k = 67$.

For the latter, which needs only $n + 1$ keys to be n-secure, $k = 67$ renders Blom's scheme 66-secure. With DOWN, the scheme is rendered $66 \times 67 = 4422$-secure. In other words, as long as 4422 (or less) nodes are compromised (and one secret exposed from each), Blom's scheme is unconditionally secure. On the other hand, when more that 4422 nodes are compromised the entire KPD is compromised.

For HARPS with $P = 15,000, k = 1000, L = 512$, the number of nodes that an attacker needs to compromise in order to compromise shared secrets between two nodes with some probability p_e is
1. Over 20,000 nodes to have the ability to eavesdrop on one in 2 trillion exchanges (or $p_e = 5 \times 10^{-13}$)
2. Over 40,000 nodes to have the ability to eavesdrop on one in 1 million exchanges ($p_e = 1 \times 10^{-6}$)
3. Over 200,000 nodes to have the ability to eavesdrop on half the exchanges (or $p_e = 0.5$)

However, for some cryptographic operations (for instance authentication of a node to the TA, or encrypting other personal secrets of a user which may in turn be used to encrypt user data), HARPS may employ all its k keys. For HARPS (and all random KPD schemes), it is significantly more difficult for an attacker to compromise all secrets in a node (which is also the shared secrets between a node and the TA) than compromising all secrets that are shared between two *peer* nodes. The ability of the attacker to achieve this (compromise of *all* secrets in a node by exposing keys from *other* nodes) is referred to as "synthesis attack" (while the former attack - determining shared secrets for the purpose of impersonation of a node for the purpose of fooling peer nodes - is referred to as "eavesdropping attack.").

To perform the synthesis attack against HARPS (with probability of success p_s) the attacker has to compromise (expose one secret from)
1. Over 5 million nodes to synthesize one in 10 trillion nodes
2. Over 85 million nodes to synthesize one in a million nodes, and
3. About 300 million nodes to synthesize every other node (or $p_s = 0.5$)

As for Blom's KPD "eavesdropping" and "synthesis" are achieved simultaneously. Blom's KPD with 1000 keys (to be fair, as HARPS uses 1000 keys for this purpose) will succumb to synthesis attack when less than a million nodes have been compromised.

HARPS has many other advantages over Blom's scheme. For instance HARPS can be readily extended to a tree-hierarchical deployment - in which each node acts as a TA for *its* child nodes. A subset of the nodes keys are provided to the child nodes - however at higher hash depths. In such a deployment, compromise of secrets at lower levels of the hierarchy does not affect the security of higher levels (or keys in child nodes do not provide any information about keys in parent node) as long as the hash function is pre-image resistant.

Further, the same secrets used for mutual pair-wise authentication could also be used for discovery of instantaneous group secrets, broadcast authentication, and broadcast encryption (or non-instantaneous establishment of group secrets). In addition, random KPDs also provide satisfactory solutions for periodic renewal. HARPS permits *partial* renewal of secrets - which caters for "seamless" periodic renewal [8].

Furthermore, each elementary cryptographic operation for Blom's KPD scheme involves exponentiation (for evaluation of a $k - 1$ order polynomial) in a prime field, while each operation with HARPS keys involve pure symmetric cryptographic primitives (block cipher or hash function). Thus among KPDs, HARPS may clearly be the better choice - especially for DOWN enabled trusted devices.

6.1 HARPS vs PKI

6.1.1 Computational Complexity

As asymmetric cryptography imposes a large computational burden, this requirement may limit "range" of devices that can take part in the deployment. While computational complexity may not be a serious issue with devices that are expected to be tamper-resistant there may be some highly resource constrained devices with perhaps lower levels of protection.

While technology for miniaturization has improved substantially, and perhaps to a point where any conceivable device would have the ability to perform asymmetric cryptography, battery technology has consistently lagged behind. A chip that is twice as fast in general consumes more than twice the power as the slower one to accomplish the same task - or it takes more energy to do a task faster. While growing confidence and improvements in elliptic curve cryptosystems (ECC) may ease the burden of asymmetric cryptography for signature schemes, they are not very efficient for encryption [41]. Considering that most exchanges between trusted devices would involve encryption rather than signatures, the advantage offered by ECC over RSA may not be substantial.

HARPS needs just a hash function or a block cipher for its implementation. As HARPS permits hierarchical deployments, it is possible to include devices that are not well protected in the deployment by including them in "lower levels" of the hierarchy.

Even under situations where nodes are not resource constrained, it would perhaps be worthwhile to direct the increased complexity necessitated by PKI (compared to KPD) to improve tamper-resistance and read-proof ability of chips, or perhaps the storage complexity (size of k) of KPDs instead.

6.1.2 Bandwidth Overheads

With PKI, two nodes need to exchange *signed public keys* (which may run up to a *few thousand bits*) before they can authenticate each other. For evolving application scenarios, with perhaps thousands of nodes within wireless range, contention for channel access may become a severe problem unless the bandwidth of exchanges between nodes are kept minimal (to ensure that the channel use never reaches close to its capacity, and therefore limiting the problem of contention). Further, for wireless communications the battery power drained by the transmitter may be substantially larger than the power drained by the processor and other peripherals - making bandwidth very expensive. Once again, ECC (and most "faster" public key encryption alternatives) result in *expansion of encrypted data* - which is a severe penalty to pay when bandwidth is at a premium.

For KPDs, on the other hand, two nodes just need to exchange their IDs before they can authenticate each other.

6.1.3 Secure RAM Requirement

In order to achieve effective protection we might need to keep the size of "secure" RAM (the protected cache memory accompanied by guarantee G1) minimal - we might want to limit the areas of the chips to which the attacker cannot have a clear line-of-sight access (to avoid tapping lines using FIB techniques). Asymmetric cryptographic primitives will in general need more RAM, and therefore it may be more expensive to protect the secrets.

6.1.4 Non-volatile Storage Complexity

PKI would need far less non-volatile storage than KPDs (for storing KDS secrets). For example, PKI using 1024-bit RSA, the storage needed is just 1024 bits compared to say HARPS which would need 128,000 bits if each node has 1000 128-bit keys. However, as the non-volatile storage need not be secure if DOWN policy is used (as the secrets are encrypted whenever they are stored in non-volatile memory), the insecure storage complexity may not be a big concern.

6.1.5 Dissemination of Public Keys

Another issue with PKI for huge-scale deployments would be the problem of dissemination of public keys. With billions of devices, certificate chains [42] needed for mutual authentication could become very long.

At the core if this issue is the fact that for public key cryptographic schemes one does not have the freedom to "choose" public keys. The public key is dictated by the choice of the private keys. For example, in RSA, the choice of secret primes p and q would dictate the public value $n = pq$. On the other hand, with KPDs, the "public key" (the ID of a node) could be chosen by hashing a descriptive string (say "First Name, Last Name, Affiliation"). The private secrets are assigned *after* the ID is chosen. This could go a long way in easing the problem of dissemination of public keys.

6.1.6 Multicast Security Primitives

While PKI does not cater for native multicast security primitives (apart from broadcast authentication using digital signatures), HARPS caters for different multicast security primitives like broadcast authentication [43], discovery of conference secrets, and broadcast encryption [44]. Broadcast encryption could be very useful for revocation of devices - for example devices suspected of having been compromised, or just devices that have been lost or stolen. While the ability to perform authenticated broadcasts is sufficient to perform revocation (by broadcasting revocation lists as in PKI), each device would need to *store* the list of currently revoked nodes. However, if broadcast encryption were used instead, nodes just need to store the latest revocation secret (which will not be available to revoked nodes). The current revocation secret would also be used by the nodes in addition to their shared secret (provided by the KDS) for mutual authentication. Especially in large scale deployments where a substantial number of nodes might be revoked at any point in time, storing revocation lists may not be feasible.

However, broadcast authentication using KPDs may have require larger bandwidth than PKI. Typically broadcast authentication with shared keys involves appending HMAC using each of the k keys [43], [45] of the source node. By relaxing security requirements, it could be reduced to one bit per HMAC [45] (for instance just the LSB of the HMAC could be used). This still may imply that the bandwidth needed may be more for KPD (compared to say DSA) for this purpose. Nevertheless, in situations (say mobile ad hoc networks) where broadcast authentication is crucial, techniques using one-way hash chains [46] (and delayed disclosure of pre-images [47] - [49]) could be used.

While strict non-repudiation is not possible with symmetric cryptography, it is indeed possible if the devices are *trusted* - anything that comes out of the trusted device is trusted! For example, for every HMAC a trusted device calculates, it might prefix its ID to the data that is hashed. A trusted device cannot be forced to append the ID of some one else to "forge a signature." The HMACs may be based on shared secrets for cases where the number of verifiers are limited.

6.1.7 Security and Key Renewal

When used in conjunction with the DOWN policy (or the guarantee that only one secret or a part of the secret can be compro-

mised even by a sophisticated attacker), devices protected using PKI (say using RSA) are immune to compromises (except that cryptanalytic attacks are rendered *marginally* easier).

Thus while PKI might not require periodic renewal for reasons of security of keys, the cerificates issued to the nodes will have to be for a limited duration to ensure that revocation lists do not grow *indefinitely* (certificates of revoked devices will not be renewed).

However, with KPDs, the assumption is that it may not be practical for an attacker to tamper with many tens of thousands of devices (in order to achieve temporary compromises) or millions of devices to achieve node synthesis (with which an attacker may be able to effect more damages). Periodic renewal of nodes is also needed for KPDs for reasons of security (to reduce attackers motivation even to attempt compromise of tens of thousands of devices as the secrets would not serve the attacker for a long time - they are rendered useless after key updates).

While the need for periodic renewal may not be of much concern for communication devices or smartcards which are used frequently (renewal would involve an one-on-one interaction of a node with the TA over a public network [8] - say the Internet), it may not be suitable for "unattended" devices. However, the "unattended lifetime" of such devices is more likely to be restricted by their battery life rather than the period between two renewals. In other words, unattended devices might have to be attended to once in a while - for replinishing their batteries. Therefore, if such devices do not have the ability to perform one-on-one interaction with the TA for renewal, their renewal process could be relayed through a longer range communication device when they are attended to.

With a tree-hierarchical deployment of HARPS, nodes only need to approach their parent for key renewal. Typically a manufacturer of a device may possess the parent node for a particular device. For devices at lower levels of the hierarchy (say wireless devices in home equipment) the parent device may be the smartcard of the owner of the device (the smartcard may be plugged into a general purpose communication equipment in order to interact with other home devices).

6.1.8 Key Escrow

The main need for escrowing secrets stored in trusted devices (say in "highly trusted devices") arises due to the fact that trusted devices are expected to self-destruct when tampering attempts are sensed. Obviously, there are bound to be false-alarms.

The owner of the device could have stored all her personal data, encrypted with some key. The encryption key itself could be stored along with the data, after encryption with the KDS secrets. With KPDs for instance, it could mean encryption using *all* k secrets. With her trusted computer (say a smart card) "dead" the user looses access to all her personal data.

If the keys are not escrowed, then there would be no way for the unfortunate user to retrieve her data. Unfortunately, we would like the threshold for detecting tampering attempts to be low to ensure that it is practically impossible for attacker attempts to go unnoticed.

Another fundamental difference between KPDs and PKI, arising once again because of the fact that the keys stored in different nodes are not independent in the former, is the ability of the former to escrow keys. In theory, the ability to escrow keys in all devices by an "authority" implies the ability of the authority to invade on the privacy of the exchanges between nodes. However, this need not be the case if the trusted devices which escrow the keys (the parent device) are trusted not to decrypt content that is not explicitly addressed to it! For instance, the TA's device, though it might know the secrets that are necessary to decipher exchanges between A and B, would just refuse to do so. As devices used by the TA (and say all manufacturers of equipment) is not likely to have any resource constraints their design could be completely open to public scrutiny (unlike Clipper chips [50] for instance) and still perhaps be highly tamper-proof.

With KPDs, the users smart card could simply be replaced with a new one - with the same ID. If trusted devices employ PKI (say the protected secret is the private RSA key), obviously such secrets cannot be escrowed in a device that only the "owner" has acess to (as the point is to protect the secrets even from the owner of the device). Of course, the private keys of each node could also be escrowed in the TA's trusted device - which is a lot less efficient than escrowing KPD keys (where it is needed anyway). In other words, the potential need to escrow keys further reduces the advantage of employing PKI over KPDs for trusted devices.

7 Conclusions

We introduce and analyze a simple security policy - DOWN - that could significantly improve the ability of trusted devices to protect their secrets, and therefore our trust in trusted devices. We argue that DOWN with PUFs may be a good approach for trusted computers.

While the use of DOWN helps any KDS that could be used for mutual authentication of trusted devices, KPDs are rendered particularly attractive in conjuction with DOWN. This is especially true for the very simple random key pre-distribution schemes that have caught the attention of researchers lately due to the growing importance of security in resource constrained environments. However, we argue that random KPDs like HARPS [8] may be well suited for securing trusted devices even when resource constraint is *not* an issue.

One of the main advantages of the DOWN policy is that it reduces the complexity of state transition (from in-use to rest) which is perhaps the most vulnerable period of operation of a trusted device. Further, the DOWN policy also makes it unnecessary extend protection non-volatile memory (even an external storage like an SD card can be used). The primary disadvantage of the DOWN policy is the overheads associated with additional rounds of decryptions, and access from non-volatile

memory needed for decrypting parts of the KDS secrets.

References

[1] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, "Silicon Physical Random Functions," Proceedings of the 9th ACM conference on Computer and communications security, pp 148-160, 2002.

[2] A. Pfitzmann, B. Pfitzmann, M. Schunter, and M. Waidner, "Mobile User Devices and Security Modules: Design for Trustworthiness"; IBM Research Report RZ 2784 (#89262) 02/05/96, IBM Research Division, Zurich, Feb. 1996.

[3] J. Lotspiech, S. Nusser, F. Pestonoi, "Anonymous Trust: Digital Rights Management using Broadcast Encryption," Proceedings of the IEEE, **92** (6), pp 898–909, 2004.

[4] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, T. Rabin, "Tamper Proof Security: Theoretical Foundations for Security Against Hardware Tampering," Theory of Cryptography Conference, Cambridge, MA, February 2004.

[5] M.G. Zapata, "Secure Ad hoc On-demand Distance Vector Routing," Mobile Computing and Communications Review, **6**(3), 2001.

[6] R. Anderson, M. Kahn, "Tamper Resistance - a Cautionary Note," Second USENIX Workshop on Electronic Commerce Proceedings, pp 1-11, Oakland, CA 1996.

[7] Semiconductor Insights Inc., "Tamper Resistance - A Second Opinion," available at http://www.smartcard.co.uk/resources/articles/tamper-res.html.

[8] M. Ramkumar, N. Memon, "An Efficient Random Key Pre-distribution Scheme for MANET Security," IEEE Journal on Selected Areas of Communication, March 2005.

[9] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," Communications of the ACM, 21(12), December 1978.

[10] R. Blom, "An Optimal Class of Symmetric Key Generation Systems," *Advances in Cryptology: Proc. of Eurocrypt 84*, Lecture Notes in Computer Science, **209**, Springer-Verlag, Berlin, pp. 335-338, 1984.

[11] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, M. Yung, "Perfectly-Secure Key Distribution for Dynamic Conferences," Lecture Notes in Computer Science, vol 740, pp 471–486, 1993.

[12] T. Matsumoto, M.E.Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, **IT-22**(6), Dec. 1976, pp.644-654.

[13] D. R. Stinson, T. van Trung, "Some New Results on Key Distribution Patterns and Broadcast Encryption," *Designs, Codes and Cryptography*, **14** (3) pp 261–279, 1998.

[14] C.J. Mitchell, F.C. Piper, "Key Storage in Secure Networks," *Discrete Applied Mathematics*, **21** pp 215–228, 1995.

[15] M. Dyer, T. Fenner, A. Frieze and A. Thomason, "On Key Storage in Secure Networks," *Journal of Cryptology*, **8**, 189–200, 1995.

[16] M. Ramkumar, N. Memon, R. Simha, "Pre-Loaded Key Based Multicast and Broadcast Authentication in Mobile Ad-Hoc Networks," Globecom-2003, San Fransisco, CA, Dec 2003.

[17] T. Leighton, S. Micali, "Secret-key Agreement without Public-Key Cryptography,"*Advances in Cryptology* - CRYPTO 1993, pp 456-479, 1994.

[18] J.P. McGregor, R. B. Lee, "Protecting Cryptographic Keys and Computations via Virtual Secure Coprocessing," ACM SIGARCH Computer Architecture News archive, **33**(1), March 2005.

[19] G. E. Suh, C. W. O'Donnell, I. Sachdev, S. Devadas, "Design and Implementation of a Single-Chip Secure Processor Using Physical Random Functions," MIT CSAIL CSG-TR-483, Nov 2004, available at http://csg.csail.mit.edu/pubs/.

[20] G. E. Suh, C. W. O'Donnell, I. Sachdev, S. Devadas, "Design and Implementation of the AEGIS Secure Processor Using Physical Random Functions," to appear, 32nd International Symposium on Computer Architecture, Madison, Wisconsin, June 2005.

[21] O. Kommerling, M. Kuhn, "Design principles for tamper-resistant smart-card processors," Proceedings of the Usenix Workshop on Smartcard Technology, pp. 9–20, 1999.

[22] R. Anderson, M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices," IWSP: International Workshop on Security Protocols, Paris, April 1997.

[23] S. Skorobogatov, R. Anderson, "Optical Fault Induction Attacks," Cryptographic Hardware and Embedded Systems Workshop (CHES-2002), LNCS 2523, Springer-Verlag, ISBN 3-540-00409-2, pp 2–12.

[24] P. Kocher, "Differential Power Analysis," Advances in Cryptology, CRYPTO 1999, Springer LNCS series, Vol 1666, pp 388–397.

[25] M.G Karpovsky, K. Kulikowski, A. Taubin, " Robust Protection Against Fault-Injection Attacks of Smart Cards Implementing the Advanced Encryption Standard". T Proc. Int. Conference on Dependable Systems and Networks (DNS 2004), July, 2004.

[26] National Computer Security Centre, "A Guide to Understanding Data Remanence in Automated Information Systems," September 1991, available at http://www.fas.org/irp/nsa/rainbow/tg025-2.htm.

[27] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley and Sons, 2001.

[28] P. Gutman, "Secure Deletion of Data from Magnetic and Solid-State Memory," Sixth USENIX Security Symposium, San Jose, California, July 1996.

[29] P. Laackmann, B. Meier, H. Nguyen, Infineon Technologies, "Revolution In The Smart Card Security," available at http://www.epn-online.com/page/11955/revolution-in-the-smart-card-security-physical-attacks.html.

[30] Y. Ishai, A. Sahai, D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks," CRYPTO 2003, Santa Barbara, CA, Aug 2003.

[31] IEEE P1619 - Standard Architecture for Encrypted Shared Storage Media, available at http://siswg.org/docs/.

[32] Application Security Inc., "Encryption of Data at Rest: White Paper," available at http://www.appsecinc.com.

[33] R. Rivest, "All-or-nothing encryption and and the package transform," Fast Software Encryption 1997, LNCS **1267**, E. Biham Ed., Springer-Verlag, 1997.

[34] A. Desai, "The Security of All-or-Nothing Encryption," Advances in Cryptology, Crypto 2000, LNCS **1880**, M. Bellare Ed., Springer-Verlag, 2000.

[35] P.S. Ravikanth, "Physical One-Way Functions," Ph.D Thesis, Department of Media Arts and Sciences, Massachusetts Institute of Technology, 2001.

[36] P. Tuyls, B. Skoric, T. Akkermans, W. Ophey, S. Stallinga, "Security analysis of physical uncloneable functions," 25th Symposium on Information Theory in the Benelux, June 2004.

[37] D. Lim, "Extracting Secret Keys from Integrated Circuits," Masters Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 2004.

[38] F. M. Williams, N. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Press, 1977.

[39] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, "Controlled Physical Random Functions," 18th Annual Computer Security Applications Conference, San Diego, CA, Dec 2002.

[40] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, "Delay-based Circuit Authentication and Applications," Proceedings of the 2003 ACM symposium on Applied Computing, Melbourne, Florida, pp 294 – 301, 2003.

[41] P. Karu, J. Loikkanen, "Practical comparison of Fast Public-key Cryptosystems," Proceedings of the Helsinki University of Technology Seminar on Network Security, 2000.

[42] D. Clarke, J-E Elien, M. Fredette, A. Marcos, R.L.Rivest, "Certificate chain discovery in SPKI/SDSI," Journal of Computer Security, vol. 9, no. 4, pp. 285 – 322(38), 2001.

[43] M. Ramkumar, "Broadcast Authentication with Preferred Verifiers," submitted to CRYPTO 2005.

[44] M. Ramkumar, N. Memon, "A DRM Based on Renewable Broadcast Authentication," to appear, IEEE / SPIE Visual Communications and Image Processing Conference, Beijing, China, July 2005.

[45] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions," INFOCOMM'99, 1999.

[46] L. Lamport, "Password Authentication with Insecure Communication," Communications of the ACM, 24(11):770-772, November 1981.

[47] S. Cheung, "An Efficient Message Authentication Scheme for Link State Routing", Proceedings of the 13th Annual Computer Security Applications Conference, San Diego, California, December 1997, pp. 90-98.

[48] R.J. Anderson, F. Bergadano, B. Crispo, J.H. Lee, C. Manifavas and R.M. Needham, " A New Family of Authentication Protocols," ACM Operating Systems Review, vol. 32, n. 4, pp. 9-20, October 1998, ACM Press.

[49] A. Perrig, R. Canetti, D. Song, D. Tygar, "Efficient and Secure Source Authentication for Multicast," in Network and Distributed System Security Symposium, NDSS '01, Feb. 2001.

[50] W. Diffie, "The Impact of a Secret Cryptographic Standard on Encryption, Privacy, Law Enforcement and Technology," Sun Microsystems, May 1993, available at http://www.epic.org/crypto/clipper/diffie_testimony.html.